CACHE-RELATED PREEMPTION DELAY COMPUTATION FOR SET-ASSOCIATIVE CACHES
PITFALLS AND SOLUTIONS[1]

Claire Burguière, Jan Reineke, Sebastian Altmeyer[2]

Abstract
In preemptive real-time systems, scheduling analyses need—in addition to the worst-case execution time—the context-switch cost. In case of preemption, the preempted and the preempting task may interfere on the cache memory. These interferences lead to additional reloads in the preempted task. The delay due to these reloads is referred to as the cache-related preemption delay (CRPD). The CRPD constitutes a large part of the context-switch cost. In this article, we focus on the computation of upper bounds on the CRPD based on the concepts of useful cache blocks (UCBs) and evicting cache blocks (ECBs). We explain how these concepts can be used to bound the CRPD in case of direct-mapped caches. Then we consider set-associative caches with LRU, FIFO, *and* PLRU *replacement. We show potential pitfalls when using UCBs and ECBs to bound the CRPD in case of* LRU *and demonstrate that neither UCBs nor ECBs can be used to bound the CRPD in case of* FIFO *and* PLRU. *Finally, we sketch a new approach to circumvent these limitations by using the concept of relative competitiveness.*

1. Introduction

Preemption introduces a new dimension of complexity into worst-case execution time (WCET) analysis: The possible interference of preempting and preempted task—especially on the cache—has to be taken into account. The additional execution time due to preemption is referred to as the context-switch cost (CSC), the part of the context switch cost due to cache interferences as cache-related preemption delay (CRPD). One approach to soundly deal with cache interferences due to preemption is to totally avoid them by cache partitioning. When cache partitioning is not an option, one can distinguish two other approaches: 1) to incorporate cache interferences within WCET analysis, or 2) to perform a separate analysis of the number of additional misses due to preemption. Whereas only the first alternative is applicable to processors exhibiting timing anomalies, the second one is probably more precise but relies on "timing compositional" processors.

Lee et al. [2] laid the foundation for the separate analysis of the cache-related preemption delay by introducing the notion of *useful cache block* (UCB). A cache block of the preempted task is called useful at program point P, if it may be cached at P and if it may be reused at some program point reached from P. Memory blocks that meet both criteria may cause additional cache misses that only occur in case of preemption. Hence, an upper bound on the number of UCBs gives an upper bound

[1]This work was supported by ICT project PREDATOR in the European Community's Seventh Framework Programme, by Transregional Collaborative Research Center AVACS of the German Research Council (DFG) and by ARTIST DESIGN NoE.
[2]Compiler Design Lab, Saarland University, 66041 Saarbrücken, Germany
{burguiere,reineke,altmeyer}@cs.uni-saarland.de

on the number of additional misses and, multiplied by the cache miss penalty, an upper bound on the cache-related preemption delay.

As Tan et al. [7] and, later, Staschulat et al. [6] have shown, the CRPD bound can be reduced by taking into account the preempting task. Only those cache blocks that are actually evicted due to preemption may contribute to the CRPD. So, the preempting task is analyzed in order to derive the set of *evicting cache blocks* (ECBs) and to bound the effect of the preempting task on the useful cache blocks. These concepts have been introduced for direct-mapped and set-associative caches with LRU replacement. Although widely used in practice, other replacement policies, such as PLRU or FIFO, have not been considered so far—except for [6] which suggests to adapt the CRPD computation for LRU to PLRU.

We first review how the CRPD can be bounded for direct-mapped caches. Then, we discuss CRPD computation for set-associative caches. We show that neither the number of UCBs nor the number of ECBs can be used to bound the CRPD for set-associative caches. Even for LRU, the previously proposed combination of ECBs and UCBs may underestimate the CRPD. We provide a correct formula for LRU and sketch a new approach to CRPD computation for FIFO, PLRU, and other policies.

1.1. Background: Cache Memory

Caches are fast and small memories storing frequently used memory blocks to close the increasing performance gap between processor and main memory. They can be implemented as data, instruction or combined caches. An access to a memory block which is already in the cache is called a *cache hit*. An access to a memory block that is not cached, called a *cache miss*, causes the cache to load and store the data from the main memory.

Caches are divided into *cache lines*. *Cache lines* are the basic unit to store *memory blocks*, i.e., *line-size* l contiguous bytes of memory. The set of all memory blocks is denoted by M. The cache size s is thus given by the number of cache lines c times the line-size l. A set of n cache lines forms one *cache set*, where n is the *associativity* of the cache and determines the number of cache lines a specific memory block may reside in. The number of sets is given by c/n and the cache set that memory block b maps to is given by $b \textit{ mod}(c/n)$.

Special cases are direct-mapped caches ($n = 1$) and fully-associative caches ($n = c$). In the first case, each cache line forms exactly one cache set and there is exactly one position for each memory block. In the second case, all cache lines together form one cache sets and all memory blocks compete for all positions. If the associativity is higher than 1 a *replacement policy* has to decide in which cache line of the cache set a memory block is stored, and, in case all cache lines are occupied, which memory block to remove. The goal of the replacement policy is to minimize the number of cache misses.

We will investigate CRPD computation in the context of the following three widely-used policies:

- Least-Recently-Used (LRU) used in INTEL PENTIUM I and MIPS 24K/34K
- First-In, First-Out (FIFO or Round-Robin) used in MOTOROLA POWERPC 56X, INTEL XSCALE, ARM9, ARM11
- Pseudo-LRU (PLRU) used in INTEL PENTIUM II-IV and POWERPC 75X

We will explain these policies in Section 3

2. Bounding the CRPD for direct-mapped caches

The cache-related preemption delay denotes the additional execution time due to cache misses caused by a single preemption. Such cache misses occur, if the preempting task evicts cache blocks of the preempted task that otherwise would later be reused. Therefore upper bounds on the CRPD can be derived from two directions: bounding the worst-case effect on the preempted task or bounding the effect of the preempting task. Note that the schedulability analysis might have to account for multiple preemptions. In that case, it has to multiply the CRPD bound by the number of preemptions.

For the analysis of the preempted task, Lee et al. [2] introduced the concept of useful cache blocks:

Definition 1 (Useful Cache Block (UCB))
A memory block m is called a useful cache block at program point P, if

a) m *may be cached at* P

b) m *may be reused at program point* Q *that may be reached from* P *without eviction of* m *on this path.*

In case of preemption at program point P, only the memory blocks that a) are cached and b) will be reused, may cause additional reloads. Hence, the number of UCBs at program point P gives an upper bound on the number of additional reloads due to a preemption at P. A global bound on the CRPD of the whole task is determined by the program point with the highest number of UCBs.

The worst-case effect of the preempting task is given by the number of cache blocks the task may evict during preemption. Obviously, each memory block possibly cached during the execution of the preempting task may evict a cache block of the preempted one:

Definition 2 (Evicting Cache Blocks (ECB))
A memory block of the preempting task is called an evicting cache block, if it may be accessed during the execution of the preempting task.

Accessing an ECB in the preempting task may evict a cache block of the preempted task. Tomiyama and Dutt [8] proposed to use only the number of ECBs—in a more precise manner—to bound the CRPD. Negi et al. [3] and Tan et al. [7] proposed to combine the number of ECBs and UCBs to improve the CRPD bounds; only useful cache blocks that are actually evicted by an evicting cache block may contribute to the CRPD.

Hence, the following three formulas can be used to bound the CRPD for direct-mapped caches (where c denotes the number of cache sets):

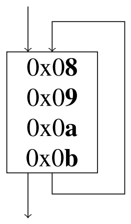
$$\text{CRPD}_{\text{UCB}} = \text{CRT} \cdot |\{s_i \mid \exists m \in \text{UCB} : m \bmod c = s_i\}| \tag{1}$$

The CRPD is bounded by the cache reload time (CRT) times the number of sets, which at least one UCB maps to [2].

$$\text{CRPD}_{\text{ECB}} = \text{CRT} \cdot |\{s_i \mid \exists m \in \text{ECB} : m \bmod c = s_i\}| \tag{2}$$

The CRPD is bounded by the cache reload time times the number of sets, which at least one ECB maps to [8].

$$\text{CRPD}_{\text{UCB\&ECB}} = \text{CRT} \cdot |\{s_i \mid \exists m \in \text{UCB} : m \bmod c = s_i \wedge \exists m' \in \text{ECB} : m' \bmod c = s_i\}| \tag{3}$$



ECBs $= \{\mathbf{e}\}$

Without preemption:

$[b, a, 9, 8] \xrightarrow{8} [8, b, a, 9] \xrightarrow{9} [9, 8, b, a] \xrightarrow{a} [a, 9, 8, b] \xrightarrow{b} [b, a, 9, 8]$ 0 misses

With preemption:

$[\mathbf{e}, b, a, 9] \xrightarrow{8^*} [8, \mathbf{e}, b, a] \xrightarrow{9^*} [9, 8, \mathbf{e}, b] \xrightarrow{a^*} [a, 9, 8, \mathbf{e}] \xrightarrow{b^*} [b, a, 9, 8]$ 4 misses

Figure 1. Evolution of the cache contents for LRU replacement. The first row shows the evolution of the cache contents accessing $8, 9, a, b$ without preemption. The second row shows the evolution of the cache contents on the same sequence with preemption. Preempting task accesses block e that maps to this cache set. Each miss is marked by *. Blocks 8, 9, a and b are useful before this access sequence.

The CRPD is bounded by the cache reload time times the number of sets, which at least one UCB and one ECB map to [3, 7].

The computation of UCBs and ECBs is out of the scope of this paper. Detailed information can be found in the original paper [2], as well as in [3, 6].

3. Bounding the CRPD for set-associative instruction caches

The definitions of UCBs and ECBs apply to all types of cache architectures, including set-associative caches with any replacement policy. Of course, whether a block is useful or not depends on the particular cache architecture, i.e., its associativity and replacement policy. So, a UCB analysis needs to be tailored to a specific cache architecture. In addition, for set-associative caches, the CRPD computation based on UCBs and ECBs differs from one replacement policy to another. As we show in this section, several pitfalls may be encountered on the way to the CRPD computation for the most common replacement policies LRU, FIFO, and PLRU.

3.1. LRU – Least-Recently-Used

Least-Recently-Used (LRU) policy replaces the least-recently-used element on a cache miss. It conceptually maintains a queue of length k for each cache set, where k is the associativity of the cache. In the case of LRU and associativity 4, $[b, c, e, d]$ denotes a cache set, where elements are ordered from most- to least-recently-used. If an element is accessed that is not yet in the cache (a miss), it is placed at the front of the queue. The last element of the queue, i.e., the least-recently-used, is then removed if the set is full. In our example, an access to f would thus result in $[f, b, c, e]$. Upon a cache hit, the accessed element is moved from its position in the queue to the front, in this respect treating hits and misses equally. Accessing c in $[f, b, c, e]$ results in $[c, f, b, e]$.

Since we concentrate on the use rather than on the computation of UCBs, we refer to Lee et al. [2] and Staschulat et al. [6] in case of LRU. For the derivation of the CRPD, there are again three possibilities: UCBs, ECBs and the combination of both.

The cache state after preemption may cause additional cache misses compared with the cache state without preemption. However the difference in the number of misses, and thus, the number of additional reloads, per cache set is always bounded by the associativity n: This is because any two initially different cache-set states converge within n accesses for LRU [4]. In particular, this holds for the cache states after preemption and the cache state without preemption. Furthermore, the number of additional misses is bounded by the number of reused memory blocks. Thus, the number of

UCBs bounds the additional cache misses.

However, due to control flow joins in the UCB analysis, the number of UCBs per set may exceed the associativity of the cache. Therefore, we have to limit the number of UCBs per set to n:

$$\text{CRPD}^{\text{LRU}}_{\text{UCB}} = \text{CRT} \cdot \sum_{s=1}^{c} min(|\text{UCB}(s)|, n) \tag{4}$$

where UCB(s) denotes the set of UCBs mapping to cache set s.
By using solely the number of ECBs to bound the CRPD, a more complex formula is needed. As shown in Figure 1, a single ECB may cause up to n additional cache misses in the set it maps to. Therefore, the number of additional cache misses for set s (CRPD$^{\text{LRU}}_{\text{ECB}}(s)$) is zero, in case no ECB maps to set s, and, otherwise, bounded by the associativity (n) of the cache.

$$\text{CRPD}^{\text{LRU}}_{\text{ECB}} = \sum_{s=1}^{c} \text{CRPD}^{\text{LRU}}_{\text{ECB}}(s) \tag{5}$$

where

$$\text{CRPD}^{\text{LRU}}_{\text{ECB}}(s) = \begin{cases} 0 & \text{if ECB}(s) = \emptyset \\ \text{CRT} \cdot n & \text{otherwise} \end{cases} \tag{6}$$

where ECB(s) denotes the set of ECBs mapping to cache set s.
To bound the number of additional reloads for cache set s using both UCBs and ECBs, Tan et al. [7] proposed the minimum function on the number of UCBs, the number of ECBs and the number of ways:

$$\text{CRPD}^{\text{LRU}}_{\text{MIN}}(s) = \text{CRT} \cdot \min(|\text{UCB}(s)|, |\text{ECB}(s)|, n) \tag{7}$$

where UCB(s) and ECB(s) denote the sets of UCBs and ECBs, respectively, mapping to cache set s.

However, this function gives an underestimation on the number of misses in several cases. Consider the CFG of Figure 1. All memory blocks map to the same cache set. Therefore, at the end of the execution of this basic block, the content of the 4-way set is given by $[b, a, 9, 8]$. As this basic block forms the body of a loop, these memory blocks are useful. One block of the preempting task maps to this set and evicts only one useful cache block: Using the minimum function, only one additional miss is taken into account for this memory set ($min(4, 1, 4) = 1$). However, the number of additional misses, four, is greater than the number of ECBs, one: All useful cache blocks are evicted and reloaded. In this example, the number of UCBs and the number of ways are upper bounds on the number of misses, but the minimum function results in an underestimation of the CRPD.

Instead of using the formula by Tan et al., the results from the CRPD computation via UCB and via ECB can be combined in a straight-forward manner:

$$\text{CRPD}^{\text{LRU}}_{\text{UCB\&ECB}} = \sum_{s=1}^{c} \text{CRPD}^{\text{LRU}}_{\text{UCB\&ECB}}(s) \tag{8}$$

where

$$\text{CRPD}^{\text{LRU}}_{\text{UCB\&ECB}}(s) = \begin{cases} 0 & \text{if ECB}(s) = \emptyset \\ \text{CRT} \cdot \min(|\text{UCB}(s)|, n) & \text{otherwise} \end{cases} \tag{9}$$

Again, UCB(s) and ECB(s) denote the sets of UCBs and ECBs, respectively, mapping to cache set s.

If no ECB maps to cache set s, no additional cache misses occur—as in Equation 6. Otherwise, the number of additional cache misses is bounded by the number of UCBs and the associativity of the cache—as in Equation 4.

3.2. FIFO – First-In, First-Out

First In, First Out (FIFO, also known as Round-Robin) bases its replacement decisions on *when* an element entered the cache, *not* on the time of its most-recent use. It replaces the element which has been resident in the cache for the longest time. FIFO cache sets can also be seen as a queue: new elements are inserted at the front evicting elements at the end of the queue. This resembles LRU. In contrast to LRU, hits do *not* change the queue. In our representation of FIFO cache sets, elements are ordered from last-in to first-in: Assume an access to f. In $[b, c, e, d]$, d will be replaced on a miss, resulting in $[f, b, c, e]$.

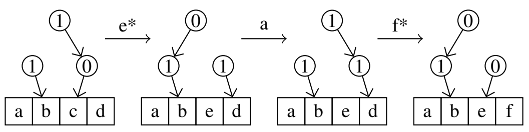
$$\begin{tikzcd}
\text{Without preemption:} & & {[b,a]} \arrow[r, "a"] & {[b,a]} \arrow[r, "e^*"] & {[e,b]} \arrow[r, "b"] & {[e,b]} \arrow[r, "c^*"] & {[c,e]} \arrow[r, "e"] & {[c,e]} & \text{2 misses} \\
& \text{ECBs} = \{\mathbf{x}\} \arrow[ur] \arrow[dr] & & & & & & & \\
\text{With preemption:} & & {[\mathbf{x},b]} \arrow[r, "a^*"] & {[a,\mathbf{x}]} \arrow[r, "e^*"] & {[e,a]} \arrow[r, "b^*"] & {[b,e]} \arrow[r, "c^*"] & {[c,b]} \arrow[r, "e^*"] & {[e,c]} & \text{5 misses}
\end{tikzcd}$$

Figure 2. Evolution of the cache contents for FIFO replacement. The first row shows the evolution of the cache contents accessing a, e, b, c, e without preemption. The second row shows the evolution of the cache contents on the same sequence with preemption. Each miss is marked by *. Blocks a and b are useful before this access sequence.

The definitions of UCBs and ECBs also apply to FIFO caches. However, there is no correlation between the number of additional misses due to preemption and the number of UCBs, the number of ECBs or the number of ways. We illustrate this using the example of Figure 2. Consider a 2-way set-associative FIFO cache. The preemption occurs before the presented sequence: Blocks a and b are useful, memory block a is evicted and the final content of this set after preemption is $[x, b]$. The number of misses in the case without preemption is two and it is five in the case of preemption. The number of additional misses, three, is greater than the number of UCBs, two, the number of ways, two, and the number of ECBs, one. So, these numbers cannot be used as an upper bound on the number of additional misses when FIFO replacement is used.

3.3. PLRU – Pseudo-LRU

Pseudo-LRU (PLRU) is a tree-based approximation of the LRU policy. It arranges the cache lines at the leaves of a tree with $k - 1$ "tree bits" pointing to the line to be replaced next; a 0 indicating the left subtree, a 1 indicating the right. After every access, all tree bits on the path from the accessed line to the root are set to point away from the line. Other tree bits are left untouched. Consider the following example of 3 consecutive accesses to a set of a 4-way set-associative PLRU cache:



The first access in the sequence, to e, is a miss. It replaces c, which is pointed to by the tree bits. The tree bits on the path from e to the root of the tree are flipped to protect e from eviction. The second access in the sequence, to a, is a hit. The left tree bit already points away from a, so only the root tree bit is flipped. Another access to a would not change the tree bits at all. Finally, the access to f, is another miss, replacing d.

As for FIFO replacement, the definition of UCB applies to the PLRU replacement strategy; but the number of UCBs, the number of ECBs and the number of ways are not upper bounds on the number of additional misses due to preemption. This is shown in Figure 3. There are two accesses to this set during the execution of the preempting task: Block b and d are evicted by x and y. The number of UCBs, four, the number of ECBs, two, and the number of ways, four, are lower than the number of additional misses, five. Thus, to compute an upper bound on the CRPD, one cannot simply use these numbers to derive an upper bound on the number of additional cache misses.

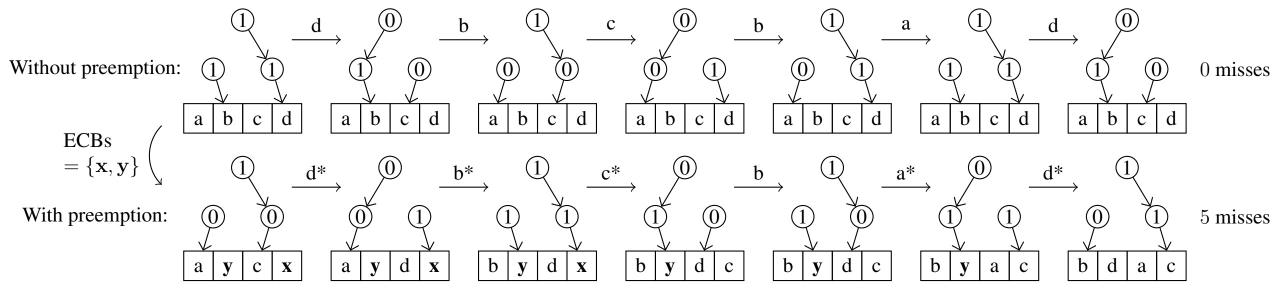


Figure 3. Evolution of the cache contents for PLRU replacement. The first row shows the evolution of the cache contents accessing d, b, c, b, a, d without preemption. The second row shows the evolution of the cache contents on the same sequence with preemption. Each miss is marked by *. Blocks a, b, c and d are useful before this access sequence.

For both FIFO and PLRU, we can extend the access sequences of our examples in Figures 2 and 3 such that the number of additional misses grows arbitrarily. In general, policies whose miss-sensitivity is greater than 1 exhibit such problems [4].

4. A new approach for FIFO, PLRU, and other policies

In the previous section, we have seen that the number of UCBs and the number of ECBs cannot be used to bound the CRPD for FIFO and PLRU. In this section, we *sketch* a new approach to the CRPD computation for policies other than LRU, including FIFO and PLRU. Due to space limitations, we cannot provide correctness proofs of the proposed approaches. Our basic idea is to transform guarantees for LRU to guarantees for other policies. To this end, we make use of the relative competitiveness of replacement policies:

4.1. Relative Competitiveness

Relative competitive analyses yield upper bounds on the number of misses of a policy P relative to the number of misses of another policy Q. For example, a competitive analysis may find out that policy P will incur at most 30% more misses than policy Q in the execution of any task.

For the definition of relative competitiveness we need the following notation: $m_P(p, s)$ denotes the number of misses of policy P on access sequence $s \in M^*$ starting in state $p \in C^P$. M denotes the set of memory blocks and C^P the set of states of policy P. For the precise definition of *compatible*, we refer the reader to [5]. Intuitively, it ensures that the policies are not given an undue advantage by the starting state.

Definition 3 (Relative Miss-Competitiveness)
A policy P is (k, c)-miss-competitive relative to policy Q, if

$$m_P(p, s) \leq k \cdot m_Q(q, s) + c$$

for all access sequences $s \in M^*$ and compatible cache-set states $p \in C^P, q \in C^Q$.

In other words, policy P will incur at most k times the number of misses of policy Q plus a constant c on any access sequence. For a pair of policies P and Q, there is a smallest k, such that P is k-miss-competitive relative to Q. This is called the *relative competitive ratio* of P to Q. Relative competitive ratios can be computed automatically for a pair of policies [5][2].

The competitiveness of FIFO and PLRU relative to LRU has been studied in [4, 5]. The most important results in our context are (the numbers in parentheses denote the associativities of the policies):

- PLRU(k) is $(1, 0)$-miss-competitive relative to LRU$(1 + log_2 k)$.
- FIFO(k) is $(\frac{k}{k-l+1}, l)$-miss-competitive relative to LRU(l).

This means that a PLRU-controlled cache of associativity k always performs at least as good as an LRU-controlled cache of associativity $1 + log_2 k$ with the same number of cache sets. In contrast to PLRU, it is not possible to achieve $(1, 0)$-miss-competitive relative to LRU for FIFO. However, the greater the associativity of FIFO is relative to the associativity of LRU, the closer $\frac{k}{k-l+1}$ gets to 1.

The relative competitiveness results mentioned above are for fully-associative caches. However, they can easily be lifted to set-associative caches, which can be seen as the composition of a number of fully-associative sets. If P is (k, c)-miss-competitive relative to Q in the fully-associative case, it is $(k, c \cdot s)$-miss-competitive relative to Q for a set-associative cache with s cache sets.

4.2. WCET and CRPD computation based on Relative Competitiveness

We have seen in the previous section that neither the number of UCBs nor the number of ECBs can be used to bound the CRPD for FIFO and PLRU. Our approach relies on the following observation: WCET and CRPD are always used in combination by schedulability analysis. For schedulability analysis it is sufficient, if (WCET bound + n·CRPD estimate) is greater than the execution time of the task including the cost of n preemptions. In that case, for safe schedulability analysis, the CRPD estimate alone does not have to bound the real CRPD. Recent work on CRPD computation provides only this weaker guarantee [1]. In addition, all sound existing approaches [2, 6] for LRU naturally also fulfill this weaker guarantee. For FIFO and PLRU, this allows us to compute a CRPD estimate that is not necessarily a bound on the CRPD itself: We will compute the WCET bound and CRPD estimate in such a way that together they provide the guarantee described above. The idea behind the approach we will present is to adapt WCET and CRPD analyses, which were originally designed for LRU, to other policies.

The CRPD accounts for all additional reloads due to preemption. Memory accesses that cause additional reloads due to preemption might have been considered to be hits by the WCET analysis, which assumes uninterrupted execution. The sequence of memory accesses along the execution of the preempted task can be split into two subsequences: the sequence of memory accesses before the

[2]See `http://rw4.cs.uni-sb.de/~reineke/relacs` for an applet that computes relative competitive ratios.

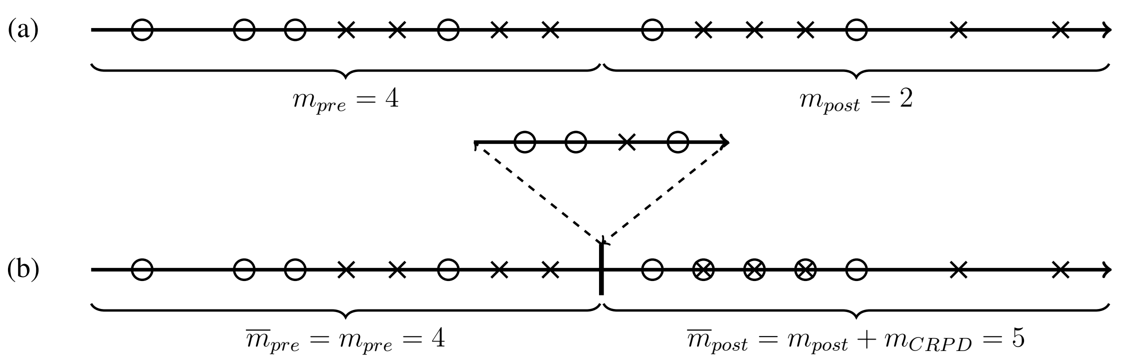

(a)
$m_{pre} = 4$
$m_{post} = 2$
(b)
$\overline{m}_{pre} = m_{pre} = 4$
$\overline{m}_{post} = m_{post} + m_{CRPD} = 5$

Figure 4. Sequence of memory accesses during task execution without preemption (a) and with preemption (b). $\bigcirc$ represents a miss, $\times$ a hit and $\otimes$ an additional miss due to preemption.

preemption point and the sequence after it. Figure 4 illustrates this situation. First we consider an uninterrupted task execution. Then, m_{pre} and m_{post} denote the number of misses on the sequences before and after the preemption point. For the interrupted execution, $\overline{m}_{pre}$ and $\overline{m}_{post}$ denote the number of misses on the same sequences including the effects of the preemption. As the execution of the first sequence is not influenced by the preemption, $\overline{m}_{pre}$ and m_{pre} are equal. m_{CRPD} is the number of additional misses due to preemption. So, $\overline{m}_{post}$ is the sum of m_{post} and m_{CRPD}. The total number of misses on the sequence is given by: $\overline{m} = \overline{m}_{pre} + \overline{m}_{post} = m_{pre} + (m_{post} + m_{CRPD}) = m + m_{CRPD}$.

For each of the two sequences, relative competitiveness allows to transfer the number of misses for one policy to a bound of a number of misses for another policy. Let policy $\mathrm{P}(t)$ be (k, c)-miss-competitive relative to $\mathrm{LRU}(s)$, and let $m^{\mathrm{P}(t)}$ and $m^{\mathrm{LRU}(s)}$ denote the number of misses of the two policies. Then,

$$\begin{aligned} \overline{m}_{pre}^{\mathrm{P}(t)} &\leq k \cdot \overline{m}_{pre}^{\mathrm{LRU}(s)} + c \\ &= k \cdot m_{pre}^{\mathrm{LRU}(s)} + c \end{aligned} \qquad \text{and} \qquad \begin{aligned} \overline{m}_{post}^{\mathrm{P}(t)} &\leq k \cdot \overline{m}_{post}^{\mathrm{LRU}(s)} + c \\ &= k \cdot (m_{post}^{\mathrm{LRU}(s)} + m_{CRPD}^{\mathrm{LRU}(s)}) + c. \end{aligned}$$

So, the total number of misses of policy $\mathrm{P}(m)$ can also be bounded by the number of misses of $\mathrm{LRU}(s)$:

$$\begin{aligned} \overline{m}^{\mathrm{P}(t)} &= \overline{m}_{pre}^{\mathrm{P}(t)} + \overline{m}_{post}^{\mathrm{P}(t)} \\ &\leq k \cdot m_{pre}^{\mathrm{LRU}(s)} + c + k \cdot (m_{post}^{\mathrm{LRU}(s)} + m_{CRPD}^{\mathrm{LRU}(s)}) + c \\ &= (k \cdot (m_{pre}^{\mathrm{LRU}(s)} + m_{post}^{\mathrm{LRU}(s)}) + c) + (k \cdot m_{CRPD}^{\mathrm{LRU}(s)} + c) \\ &= (k \cdot m^{\mathrm{LRU}(s)} + c) + (k \cdot m_{CRPD}^{\mathrm{LRU}(s)} + c). \end{aligned}$$

We can split this bound into $m^{\mathrm{P}(t)} = k \cdot m^{\mathrm{LRU}(s)} + c$ and $m_{CRPD}^{\mathrm{P}(t)} = k \cdot m_{CRPD}^{\mathrm{LRU}(s)} + c$, such that $\overline{m}^{\mathrm{P}(t)} \leq m^{\mathrm{P}(t)} + m_{CRPD}^{\mathrm{P}(t)}$. Note that while we *do* obtain a bound on the number of misses for $\mathrm{P}(t)$ including the preemption, namely $m^{\mathrm{P}(t)} + m_{CRPD}^{\mathrm{P}(t)}$, we do *not* necessarily obtain a bound on the cache-related preemption delay for $\mathrm{P}(t)$, i.e., $m_{CRPD}^{\mathrm{P}(t)}$ itself does not bound the real CRPD.

WCET and CRPD analyses take into account all possible access sequences. Our calculations above considered a single access sequence. However, the result can be lifted to all possible access sequences by considering bounds on the number of additional misses (CRPD) and bounds on the number of misses during task execution. Also, we have considered exactly one preemption in our calculations. However, it can be verified rather easily, that $m^{\mathrm{P}(t)} + n \cdot m_{CRPD}^{\mathrm{P}(t)}$ is a bound on the number of misses for $\mathrm{P}(t)$ including up to n preemptions by similar calculations as above.

What does all this mean for the WCET and CRPD analysis of FIFO and PLRU? Due to its $(1, 0)$-miss-competitive relative to $\mathrm{LRU}(1 + log_2 k)$, $\mathrm{PLRU}(k)$ is particularly easy to treat: One can simply perform WCET and CRPD analyses assuming an $\mathrm{LRU}(1 + log_2 k)$ cache. The resulting bounds will also be valid for $\mathrm{PLRU}(k)$. The situation becomes more difficult for FIFO. It is not $(1, 0)$-miss-competitive relative to $\mathrm{LRU}(k)$ for any associativity. $\mathrm{FIFO}(k)$ is $(\frac{k}{k-l+1}, l)$-miss-competitive relative to $\mathrm{LRU}(l)$. To handle FIFO correctly, both the WCET and the CRPD analyses need to be adapted. $m_{CRPD}^{\mathrm{FIFO}(k)} = \frac{k}{k-l+1} \cdot m_{CRPD}^{\mathrm{LRU}(l)} + l$ can be used as a CRPD estimate for $\mathrm{FIFO}(k)$ given that $m_{CRPD}^{\mathrm{LRU}(l)}$ is a sound estimate for $\mathrm{LRU}(l)$. Likewise, the WCET analysis for $\mathrm{FIFO}(k)$ needs to account for $\frac{k}{k-l+1}$ as many misses as it would have to for $\mathrm{LRU}(l)$ plus l misses. For WCET analyses that explicitly compute an upper bound on the number of cache misses, like ILP-based approaches, this can be rather easily. Other WCET analyses that take cache misses into account as part of the execution time of basic blocks it maybe more difficult to account for the higher number of misses.

5. Conclusions and Future Work

Prior useful cache block analyses mainly focus on direct-mapped caches. They use the number of UCBs and/or ECBs to derive an upper-bound on the CRPD. As we have shown in this paper, considering set-associative caches, the CRPD computation is much more complex. In case of LRU replacement, the computation of the CRPD solely based on the number of UCBs is correct, whereas a previously proposed formula combining UCBs and ECBs may underapproximate the CRPD. In contrast to LRU, for PLRU and FIFO policies, neither UCBs nor ECBs can be used to bound the CRPD.

For LRU, we introduce a new CRPD formula based on UCBs and ECBs. For other policies, we sketch a new approach to bound the CRPD. This approach is based on the concept of relative competitiveness: Bounds obtained for LRU are transformed into bounds for other policies. For future work we plan to implement and experimentally evaluate this approach.

References

[1] ALTMEYER, S., AND BURGUIÈRE, C. A new notion of useful cache block to improve the bounds of cache-related preemption delay. In *Proceedings of the 21st Euromicro Conference on Real-Time Systems (ECRTS '09)* (July 2009), IEEE Computer Society, pp. 109–118.

[2] LEE, C.-G., HAHN, J., MIN, S. L., HA, R., HONG, S., PARK, C. Y., LEE, M., AND KIM, C. S. Analysis of cache-related preemption delay in fixed-priority preemptive scheduling. In *RTSS'96* (1996), IEEE Computer Society, p. 264.

[3] NEGI, H. S., MITRA, T., AND ROYCHOUDHURY, A. Accurate estimation of cache-related preemption delay. In *CODES+ISSS'03* (2003), ACM.

[4] REINEKE, J. *Caches in WCET Analysis*. PhD thesis, Universität des Saarlandes, , Saarbrücken, November 2008.

[5] REINEKE, J., AND GRUND, D. Relative competitive analysis of cache replacement policies. In *LCTES'08* (June 2008), ACM, pp. 51–60.

[6] STASCHULAT, J., AND ERNST, R. Scalable precision cache analysis for real-time software. *ACM Trans. on Embedded Computing Sys. 6*, 4 (2007), 25.

[7] TAN, Y., AND MOONEY, V. Integrated intra- and inter-task cache analysis for preemptive multi-tasking real-time systems. In *SCOPES'04* (2004), pp. 182–199.

[8] TOMIYAMA, H., AND DUTT, N. D. Program path analysis to bound cache-related preemption delay in preemptive real-time systems. In *CODES'00* (2000), ACM.