CISTER
Research Centre in
Real-Time & Embedded
Computing Systems

Conference Paper

Integrating the Calculation of Preemption and Persistence Related Cache Overhead

Syed Aftab Rashid
Geoffrey Nelissen
Eduardo Tovar

CISTER-TR-161005

Integrating the Calculation of Preemption and Persistence Related Cache Overhead

Syed Aftab Rashid, Geoffrey Nelissen, Eduardo Tovar

*CISTER Research Centre

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail:

http://www.cister.isep.ipp.pt

Abstract

Integrating the Calculation of Preemption and Persistence Related Cache Overhead

Syed Aftab Rashid, Geoffrey Nelissen, Eduardo Tovar
CISTER/INESC TEC, ISEP
Polytechnic Institute of Porto, Portugal

I. Introduction

The increasing gap between processor and main memory operating speeds motivated the introduction of caches in modern processors. Program data and/or instructions that are loaded in caches are readily available to the processor in comparison to fetching it from the main memory, thereby resulting in a faster execution time for the tasks running on the processor. Most commercial-of-the-shelf (COTS) processors use caches to decrease average-case memory access latency. However, as caches have a limited capacity in comparison to the main memory, it results that not all the data and instructions of all tasks can simultaneously reside in the cache. Tasks compete for space in the cache and one task might potentially evict cache content loaded by the others tasks. This can cause big variations in the execution time of the task, depending on whether the instruction and/or data it requires are already loaded in the cache (cache hit) or not (cache miss).

In systems where preemptions are allowed, the preempted tasks may suffer additional cache misses if its useful memory blocks (i.e., blocks that are used more than once by the task during its execution) are evicted from the cache due to the execution of the preempting tasks. These evictions cause extra accesses to the main memory, which result in additional delays during the task execution. This extra cost is usually referred to as cache-related preemption delays (CRPDs).

The literature on CRPD calculation is well developed. Several approaches have been proposed to compute accurate upper-bounds on CRPDs.

In [1], we introduced the notion of *cache persistence* to reduce the pessimism involved in the state-of-the-art worst-case response time (WCRT) analyses for fixed-priority preemptive systems. In the proposed analysis, along with the effect of preemptions on the memory demand of the preempted task τ_i, we also considered the variation in memory demand of different jobs of the preempting tasks executing during the response time of the task τ_i. This variation in memory demand is mainly due to the existence of persistent cache blocks (PCBs). PCBs are the memory blocks of a task τ_j that, once loaded by τ_j, will never be invalidated or evicted from the cache when τ_j executes in isolation [1]. Unless evicted by other tasks running concurrently with τ_j, those cache blocks can thus be reused by subsequent jobs of τ_j resulting in a lower memory demand.

By definition, PCBs of a task τ_j cannot be evicted by the task itself but these PCBs can be evicted by other tasks concurrently running in the system. PCB evictions will result in an extra memory overhead for task τ_j executing during the response time of another lower-priority task τ_i. In [1], we defined this extra memory overhead as the Cache Persistence Reload Overhead (CPRO) and showed how it can be integrated into the WCRT analysis in the context of fixed priority preemptive systems.

The WCRT analysis presented in [1] accounts for both CRPDs and CPRO and dominates the state-of-the-art approaches (e.g., [2]–[5]) that only account for CRPDs. However, it can still result in overestimations due to the fact that it assumes that there is no mutual dependency between the CRPD and CPRO. Therefore, CRPD and CPRO are separately calculated and accounted for in the WCRT analysis. As shown later in this paper, the actual CPRO does in fact depend on what has already been evicted from the cache during earlier preemptions. Therefore, this inaccurate assumption results in double accounting for the same cache block evictions in situations where there is an overlap between persistent and useful cache blocks of a task.

Therefore, in order to compute tighter bounds on the memory overhead, and hence on the WCRT of each task executing in a preemptive system, we propose a novel analysis that captures the cache blocks whose evictions are accounted twice (both in CRPD and CPRO) during the response time of a task τ_i. We further present a first solution to integrate the calculation of CRPD and CPRO and hence ensure that each cache block eviction is considered only once.

II. System Model

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

We consider sporadic tasks with constrained deadlines where each task has a fixed priority. Any priority assignment scheme (e.g., Rate Monotonic or Deadline Monotonic) is acceptable. We also assume that the tasks are independent and do not suspend themselves during their execution. A task τ_i is defined by a triplet (C_i, T_i, D_i), where C_i is the worst-case execution time (WCET) of τ_i, T_i is its minimum inter-arrival time and D_i is the relative deadline of each instance (or job) of τ_i. We assume that the tasks have constrained deadlines, i.e., $D_i \leq T_i$. We further decompose each task's WCET into separate terms for processing and memory demand,

respectively. The worst-case processing demand P_i denotes the worse-case execution time of τ_i considering that every memory access is a cache hit. Consequently, it only accounts for execution requirements of the task and does not include the time needed to fetch data and instructions from the main memory. MD_i is the worst-case memory demand of any job of task τ_i, that is, the maximum time during which any job of τ_i is performing memory operations. The values for C_i, P_i and MD_i are calculated assuming τ_i executes *in isolation*. It is also important to note that the worst-case processing demand and the worst-case memory demand may not necessarily be experienced on the same execution path of τ_i. Therefore, it holds that $C_i \leq P_i + MD_i$.

The worst-case response time (WCRT) of task τ_i is defined as the longest time between the arrival and the completion of any of its jobs.

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

For convenience, we define the following set of tasks:

- $hp(i)$: the set of tasks with a priority higher than that of τ_i.
- $hep(i)$: the set of tasks with priorities higher than or equal to that of τ_i.
- $aff(i,j)$: the set of tasks with priorities higher than or equal to the priority of τ_i (including τ_i), but strictly lower than that of τ_j. This set contains the intermediate priority tasks, that can execute during the response time of τ_i but may also be preempted by τ_j.

III. Useful Concepts

A. Cache Related Preemption Delays

The state-of-the-art is quite extensive with approaches that focus on analyzing the impact of CRPDs on the WCET and WCRT of tasks in preemptive systems. CRPDs caused by a high priority task τ_j executing during the response time of a low priority task τ_i is denoted by $\gamma_{i,j}$.

In one of the earliest works, Lee et al. [4] introduced the concept of *useful cache blocks (UCBs)*, defined as follows.

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

The concept introduced in [4] was later improved by Altmeyer et al. [6]. However, in this work we only need the basic concept provided in [4]. Lee et al. [4] used the notion of UCBs to bound the preemption cost, proving that $\gamma_{i,j}$ is given by the maximum number of UCBs that can be evicted when τ_j preempts τ_i. Busquets et al. [2] and Tomiyama et al. [3] rather introduced the notion of *evicting cache block* (ECBs).

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

The CRPD $\gamma_{i,j}$ caused by τ_j is therefore upper-bounded by the number of ECBs of τ_j.

Other approaches [5], [7]–[9] used both the UCBs of the preempted tasks and ECBs of the preempting tasks in order to come up with more precise bounds on CRPDs. Due to space constraints, we will only discuss the UCB-union approach.

To calculate the preemption cost $\gamma_{i,j}$, the UCB-union approach [7] uses the ECBs of the preempting task τ_j and the UCBs of all tasks in $aff(i,j)$ possibly affected by the preemption caused by τ_j (see Equation (1)).

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

where, UCB_k and ECB_j are the sets of UCBs and ECBs of task τ_k and τ_j, respectively.

B. Cache Persistence

Authors of [1], introduced the notion of cache persistence and defined the concept of *persistent and non-persistent cache blocks (PCBs and nPCBs)* as follows.

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

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

Based on the definition of non-persistent cache blocks (nPCBs), we also introduced the notion of *residual memory demand (MD_i^r)* of a task τ_i.

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

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

$$\hat{MD}_i(t) \stackrel{\text{def}}{=} \min \left\{ \left\lceil \frac{t}{T_i} \right\rceil MD_i \; ; \left\lceil \frac{t}{T_i} \right\rceil MD_i^r + \mid PCB_i \mid \times d_{mem} \right\} \quad (2)$$

It is proved in [1] that $\hat{MD}_i(t)$ upper bounds the memory demand of task τ_i while executing in isolation assuming τ_i starts its execution with an empty cache. Similarly, the notion of CPRO is also formally defined in [1] as:

Definition 6 (Cache-persistence reload overhead)**.** *The Cache-persistence reload overhead, denoted by $\rho_{j,i}$, is the maximum memory overhead of any task τ_j due to evictions of its $PCBs$ resulting from the execution of all tasks in $hp(i) \setminus \tau_j$, while τ_j is executing during the response time of τ_i.*

The cache-persistence reload overhead ($\rho_{j,i}$) can be calculated using any of the three approaches presented in [1]. In this work, we concentrate on the CPRO-union approach, which

uses the PCBs of task τ_j and the union of the ECBs of all tasks in $hp(i) \setminus \tau_j$ to calculate $\rho_{j,i}$:

$$\rho_{j,i} = d_{mem} \times \left| PCB_j \cap \Big(\bigcup_{\forall \tau_k \in hep(i) \setminus \tau_j} ECB_k \Big) \right| \quad (3)$$

For a detailed description on the proof of Equations (2) and (3), readers are referred to [1].

IV. The Problem

The WCRT analysis presented in [1] is given by the following formulation[1]:

$$R_i = C_i + \sum_{\forall j \in hp(i)} \left\{ \left\lceil \frac{R_i}{T_j} \right\rceil (P_j + \gamma_{i,j}) + \hat{MD}_j(R_i) + \hat{\rho}_{j,i}(R_i) \right\} \quad (4)$$

where $\hat{\rho}_{j,i}(R_i)$ is the total CPRO suffered by every high priority task $\tau_j \in hp(i)$, and is given by the following equation:

$$\hat{\rho}_{j,i}(R_i) \stackrel{\text{def}}{=} \left(\left\lceil \frac{R_i}{T_j} \right\rceil - 1 \right) \times \rho_{j,i} \quad (5)$$

For a detailed description on the formulation of Equations (4) and (5), the reader is referred to [1].

The WCRT formulation in Equation (4) separately accounts for both the CRPD $\gamma_{i,j}$ and the CPRO $\rho_{j,i}$ where both these quantities are calculated using Equation (1) and (3), respectively. By definition, CRPD accounts for the evictions of UCBs whereas CPRO accounts for the evictions of PCBs. In situations where we have an overlap between the UCBs and PCBs of some tasks, the formulation in Equation (4) will sometime account for the same evictions twice (both in $\gamma_{i,j}$ and $\rho_{j,i}$) and hence results in an overestimation on the total memory overhead due to both CRPD and CPRO. This situation is illustrated using the example given below.

Example 1. *Consider a task set τ comprising three tasks $\{\tau_1, \tau_2, \tau_3\}$ with τ_1 having the highest priority and τ_3 the lowest. Fig. 1 presents an example schedule together with the evolution of the cache content over time. Cache blocks that have been evicted either due to CRPD or CPRO and must be reloaded from main memory are highlighted in red. The set of persistent cache blocks (PCBs) are highlighted in green.*

Initially, the cache is empty and with τ_3 being the first task to arrive it loads all its ECBs in the cache. When τ_2 preempts τ_3 for the first time, it also loads its ECBs. Similarly, τ_2 is soon preempted by the highest priority task τ_1 which in turn loads all its ECBs into the cache. Note that ECBs of task τ_1 and UCBs/PCBs of task τ_2 are mapped to the same cache sets, i.e., $\{7, 8, 9, 10\}$. Thus, when τ_2 resumes its execution after the completion of the first job of τ_1 it needs to reload all its UCBs, i.e., $UCB_2 = \{7, 8, 9, 10\}$ from the main memory (highlighted in red) as they were evicted during the execution of τ_1. These extra memory accesses will be accounted for as CRPD.

Since, the first job of τ_2 loads all its ECBs (PCBs and nPCBs) into the cache, subsequent jobs of τ_2 may have a lower memory demand due to the existence of persistent cache blocks, i.e., $PCB_2 = \{7, 8, 9, 10\}$. However, each job will also need to reload all the PCBs that may have been evicted due to other tasks executions. This is accounted for as CPRO.

From the execution schedule and cache contents shown in Fig. 1, we conclude that the total memory overhead (i.e., the number of cache blocks that are evicted and need to be reloaded), which accounts for both CRPD and CPRO, during the response time of τ_3 comes out to be equal to 12 (i.e., the number of blocks in red).

Note that, since τ_2 is the only task with useful cache blocks ($UCB_2 = \{7, 8, 9, 10\}$), it is also the only task incurring a CRPD. Using Equation (1), we get

$$\gamma_{3,1} = |(UCB_2 \cup UCB_1) \cap ECB_1| = 4$$

and

$$\gamma_{3,2} = |UCB_3 \cap ECB_2| = 0$$

As three jobs of both task τ_1 and τ_2 execute during the response time of task τ_3, the total CRPD is given by

$$CRPD_{total,3} = 3 \times \gamma_{3,1} = 12$$

which is indeed the number of UCBs of τ_2 evicted by τ_1 during the response time of τ_3 (see Fig. 1).

Similarly, using Equation (3), we get

$$\rho_{3,1} = |(ECB_2 \cup ECB_1) \cap PCB_3| = 0$$

and

$$\rho_{2,1} = |(ECB_3 \cup ECB_1) \cap PCB_2| = 4$$

Consequently, it follows that the only cache persistence reload overhead during the response time of task τ_3 comes from the eviction of PCBs of task τ_2. Furthermore, as three jobs of τ_2 executes during the response time of τ_3, using Equation (5) the total CPRO is given by

$$CPRO_{total,3} = 2 \times \rho_{2,3} = 8$$

Finally, adding CRPD and CPRO, the total memory overhead during the response time of task τ_3, comes out to be $CRPD_{total,3} + CPRO_{total,3} = 20$ cache blocks to be reloaded, which is a clear overestimation over the 12 cache blocks that were actually evicted and reloaded during τ_3's response time.

Example 1 shows that when accounting for both CRPD and CPRO separately, state-of-the-art approaches may consider the eviction of the same cache blocks twice and thus result in an overestimation of the total memory overhead. For instance, in the above example, UCBs of task τ_2, i.e., $\{7, 8, 9, 10\}$ are also its PCBs. Therefore, the eviction of cache block $\{7, 8, 9, 10\}$ are considered twice, once in $\gamma_{3,1}$ and then $\rho_{2,3}$, resulting in an overestimation on the total memory overhead during the response time of τ_3.

V. Integrating the calculation of CRPD and CPRO

Two interesting properties can be observed in the example of Section IV:

O1. Same cache block evictions are accounted twice (in both CRPD and CPRO), only in situations where some PCBs of a task are also its UCBs (as for task τ_2 in Example 1).

[1]We present only a simplified form of the original WCRT formulation presented in [1]

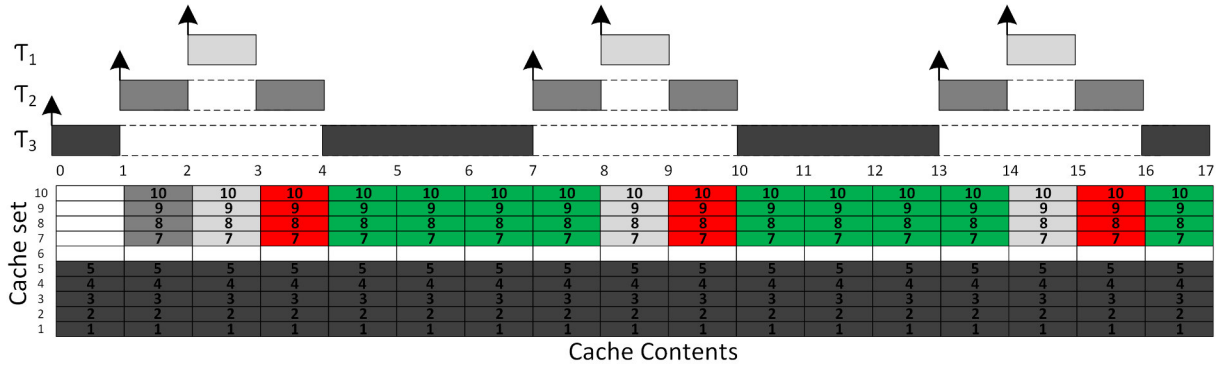

Fig. 1. Schedule and cache contents of $\{\tau_1, \tau_2, \tau_3\}$ with $C_1 = 1$, $C_2 = 2$, $C_3 = 8$, $T_1 = 6$, $T_2 = 6$, $T_3 = 25$, $ECB_1 = \{7, 8, 9, 10\}$, $ECB_2 = \{7, 8, 9, 10\}$, $ECB_3 = \{1, 2, 3, 4, 5\}$, $UCB_1 = \{\}$, $UCB_2 = \{7, 8, 9, 10\}$, $UCB_3 = \{\}$, $PCB_1 = \{\}$, $PCB_2 = \{7, 8, 9, 10\}$ and $PCB_3 = \{\}$

O2. Tasks in $aff(i, j)$ have a lower priority than τ_j and hence cannot preempt τ_j. On the other hand, tasks in $hp(j)$ have a higher priority than τ_j and can preempt τ_j. Therefore, only tasks in $hp(j)$ may cause CRPDs for τ_j and may thus participate to both the CRPD and CPRO of τ_j.

Using the above observations, when calculating the extra memory overhead (specifically CPRO) of a task τ_j executing during the response time of another task τ_i, we can improve the CPRO-union approach (Eq. (3)) by removing the effect of the evictions that have already been considered during the CRPD calculation (Eq. (1)). This leads us to the following theorem.

Theorem 1. *Assuming that CRPDs are calculated with Eq.* (1)*, the cache persistence reload overhead associated to each job of $\tau_j \in hp(i)$ released during the response time of τ_i is upper-bounded by*

$$\rho^{imp}_{j,i} = d_{mem} \times \left| PCB_j \bigcap \left(\left(\bigcup_{\forall k \in aff(i,j)} ECB_k \right) \bigcup \left(\bigcup_{\forall l \in hp(j)} ECB_l \setminus UCB_j \right) \right) \right| \quad (6)$$

Proof sketch. By Observation O2, only the tasks in $hp(j)$ may be the cause of both CRPDs and CPROs over τ_j, and by Observation O1, PCBs of τ_j that are also UCBs of τ_j were already assumed to be evicted when computing CRPDs with Eq. (1). Thus, when computing the effect of tasks in $hp(j)$ (Observation O2) on the CPRO of τ_j (i.e., $\left(PCB_j \cap (\bigcup_{\forall l \in hp(j)} ECB_l)\right)$), UCBs of τ_j that are also PCBs (Observation O1) may be removed so as to avoid to account for evictions that were already considered in the CRPD calculation (i.e., Eq. (1)). □

VI. Conclusion

We highlighted the pessimism of independently calculating CRPDs and CPROs. We proposed a first solution to reduce that pessimism by integrating the calculation of CRPDs and CPROs. This is achieved by considering the cache block evictions that have already been accounted for in the CRPD calculation, when calculating CPROs. However, the proposed result is limited to the UCB-union and CPRO-union approaches. Two methods that are known to be simple but pessimistic [1], [5]. As future work, we will formally prove the correctness of the proposed approach and extend it to more evolved methods for the CRPD and CPRO computation. A thorough evaluation of the proposed result will also be conducted in order to quantify the actual gain over the state-of-the-art, as it is expected that the identified problem and its impact on the response time analysis may be very context dependent.

Acknowledgments. This work was partially supported by National Funds through FCT/MEC (Portuguese Foundation for Science and Technology) and co-financed by ERDF (European Regional Development Fund) under the PT2020 Partnership, within project UID/CEC/04234/2013 (CISTER); also by FCT/MEC and the EU ARTEMIS JU within project ARTEMIS/0001/2013 - JU grant nr. 621429 (EMC2).

References

[1] S. A. Rashid, G. Nelissen, D. Hardy, B. Akesson, I. Puaut, and E. Tovar, "Cache-persistence-aware response-time analysis for fixed-priority preemptive systems," in *2016 28th Euromicro Conference on Real-Time Systems*. IEEE, 2016, pp. 262–272.

[2] J. V. Busquets-Mataix, J. J. Serrano, R. Ors, P. Gil, and A. Wellings, "Adding instruction cache effect to schedulability analysis of preemptive real-time systems," in *RTAS'96*. IEEE, 1996, pp. 204–212.

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

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

[5] S. Altmeyer, R. Davis, C. Maiza *et al.*, "Cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems," in *RTSS'11*. IEEE, 2011, pp. 261–271.

[6] S. Altmeyer and C. M. Burguière, "Cache-related preemption delay via useful cache blocks: Survey and redefinition," *Journal of Systems Architecture*, vol. 57, no. 7, pp. 707–719, 2011.

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

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

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