CISTER
Research Centre in
Real-Time & Embedded
Computing Systems

Conference Paper

ResilienceP Analysis: Bounding Cache Persistence Reload Overhead for Set-Associative Caches

Syed Aftab Rashid
Geoffrey Nelissen
Eduardo Tovar

CISTER-TR-190512

2019/07/09

ResilienceP Analysis: Bounding Cache Persistence Reload Overhead for Set-Associative 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, grrpn@isep.ipp.pt, emt@isep.ipp.pt

https://www.cister-labs.pt

Abstract

ResilienceP Analysis: Bounding Cache Persistence Reload Overhead for Set-Associative Caches

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

I. Motivation and Introduction

In modern systems, the latency of an access to the main memory is much higher than the latency of an individual computation on the processor. Cache memory bridge this performance gap between the main memory and processor by holding frequently required data and instructions. Intuitively, caches are used to decrease average-case memory access latency; however, due to their limited capacity in comparison to main memory the use of caches can also cause large variations in the execution times of tasks. Due to limited space, not all data/instructions of all tasks can simultaneously reside in the cache. Hence, tasks may compete for cache space, with the execution of one task potentially evicting memory blocks previously loaded into the cache by other tasks. This may result in increasing the worse-case execution/response time (WCET/WCRT) of tasks depending on whether the instructions/data needed by the tasks are already present in the cache (i.e. cache hit) or not (i.e. cache miss).

The impact of caches on the WCET/WCRT of tasks is more evident under preemptive scheduling. In preemptive scheduling, tasks may suffer additional execution delays depending on the state of the cache, namely, *Cache Related Preemption Delays* (CRPDs) and *Cache Persistence Reload Overheads* (CPROs). CRPDs are delays suffered by the preempted tasks in reloading *useful cache blocks* (UCBs) (blocks cached before the preemption and potentially reused after) that were evicted from the cache during the execution of preempting tasks. On the other hand, CPROs result from the eviction of *persistent cache blocks* (PCBs) (memory blocks that, once loaded into cache by the task, will never be invalidated or evicted by the task itself and hence always available for fast access) due to the interleave or preemptive execution with other tasks. Many different approaches have been presented in the state-of-the-art (SoA) to bound CRPDs [2] and CPROs [3], [4]. However, most of these approaches focus on CRPD/CPRO calculation assuming a *direct-mapped* cache. In a direct-mapped cache, each cache set can hold at most one memory block and in case of a cache conflict between two tasks τ_i and τ_j, each cache block used by τ_j during its execution (i.e., called an evicting cache block (ECB)) can evict at most one UCB/PCB of τ_i and vice versa. However, today most processor architectures rely on *set-associative* caches. In set-associative caches, each cache set may hold more than one memory block depending on the number of available cache *ways* (also called cache *associativity*). Hence, one cache access by a task τ_j to the same cache set used by another task τ_i, may lead to multiple cache misses for τ_i (i.e., known as the cascading effect).

The few analyses in the literature that consider set-associative caches only focus on CRPD computation. However, it has been shown in recent works [3], [4] that only considering CRPDs for tasks scheduled under fixed-priority preemptive scheduling may result in largely pessimistic WCRT bounds and that the analyses that considers both CRPD and CPRO [3], [4] dominate the WCRT analyses that only consider CRPD [1]. Considering that the existing approaches for CPRO calculation only consider direct-mapped caches, in this paper we present different approaches to bound CPRO for set-associative caches. First, we present the PCB-ECB approach that considers PCBs of the task under analysis and ECBs of all other tasks in the system to calculate CPROs. We then introduce the resilienceP analysis that removes some of the pessimism in the PCB-ECB approach by considering the *resilience* of PCBs when calculating CPRO. Finally, we present a multi-set alike resilienceP analysis that considers variation in the resilience of PCBs over different job executions of a task in order to have an even tighter CPRO bound.

II. Notations and Background

We focus on set-associative caches using the Least-Recently-Used (LRU) replacement policy, i.e., on a cache miss the least recently used memory block within a cache set is evicted. The number of memory blocks each cache set can store is known as the number of ways or the associativity of the cache and is denoted by k. The total number of sets in the cache is denoted by cs. We use d_{mem} to denote the time needed to load one cache block from the main memory into the cache. As we consider fixed priority preemptive scheduling (FPPS), we use $\mathrm{hep}(i)$ to denote the set of tasks with priorities higher than or equal to that of τ_i (hence including τ_i).

Evicting and Useful Cache Blocks (ECBs and UCBs). All cache blocks used by the task during its execution are called ECBs [5] and an ECB m is also a UCB at a program point P, if m is cached at P and may be reused at program point Q that may be reached from P without eviction of m [6].

Cache Related Preemption Delay (CRPD). when a task τ_i is preempted by a higher priority task τ_j, ECBs of τ_j may evict UCBs of τ_i that are to be reloaded from the main memory after τ_i resumes. The additional execution time incurred by τ_i due to these extra cache reloads is termed as CRPD.

For set-associative caches, the resilience analysis [7] dominates all other method in the SoA to compute CRPD. It uses the notion of *resilience* to bound the CRPD of task τ_i due to preemptions by a higher priority task τ_j.

Resilience [7]. The Resilience of a memory block m at program point P is the largest l such that all possible next accesses to m (i) would be cache hits if there is no preemption, and (ii) would still be cache hits if there is a preemption at

program point P with l cache accesses to the same cache set as m. The Resilience of a cache block m at a program point P is given by

$$res_P(m) = (k-1) - max\text{-}age_P(m) \quad (1)$$

where $max\text{-}age_P(m)$ is the *maximum* LRU-age of m at program point P, i.e., the maximum number of accesses to the same cache set as m from the last use of m before or at program point P to the next access to m after P [7]. In resilience analyses, the CPRD of task τ_i due to a single preemption by a higher priority task τ_j in a cache set s is given by $\gamma^{res,s}_{i,j}$;

$$\gamma^{res,s}_{i,j} = d_{mem} \times |UCB^s_i \setminus \{m_i | res(m_i) \geq |ECB^s_j|\}| \quad (2)$$

where $|UCB^s_i|$ and $|ECB^s_j|$ denote the number of UCBs/ECBs of τ_i and τ_j in cache set s. Effectively, the total CRPD over all cache sets $s \in cs$ is given by $\gamma^{res}_{i,j}$, where

$$\gamma^{res}_{i,j} = \sum_{s=0}^{cs} \gamma^{res,s}_{i,j} \quad (3)$$

Note that since the number of UCBs and the resilience is calculated for each program point, $\gamma^{res}_{i,j}$ is given by the program point that maximize Eq. (3) over all program points. Details on the formulation of Eq. (1)-(2) can be found in [7].

Persistent Cache Block (PCB) [3]. A memory block m_i of task τ_i is a PCB if, once loaded by τ_i, m_i will never be invalidated or evicted from the cache when τ_i executes in isolation.

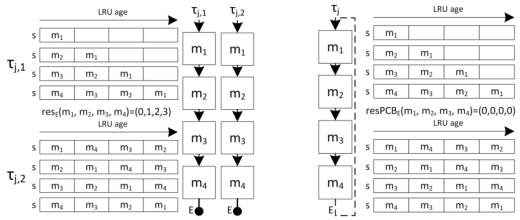
Cache Persistence Reload Overhead (CPRO) [3]. The CPRO of a task τ_j executing during the response time of a task τ_i is denoted by $\rho_{j,i}$ and is formally defined as the maximum memory reload overhead suffered by task τ_j due to evictions of its PCBs by tasks in $\mathrm{hep}(i) \setminus \tau_j$.

III. PCB-ECB Approach for CPRO calculation

Existing approaches for CPRO calculation [3], [4] cannot be used as is for set-associative caches. This is due to the cascading effect in set-associative LRU caches which may result in evicting several PCBs of task τ_j due to a single ECB of tasks in $\mathrm{hep}(i) \setminus \tau_j$. This effect does not happen in a direct-mapped cache where each ECB of tasks $\in \mathrm{hep}(i) \setminus \tau_j$ can evict at most one PCB of τ_j. Before presenting our solution for set-associative caches, we first recall the CPRO-union approach [3] that calculates the CPRO $\rho^{dir}_{j,i}$ of a task τ_j executing during the response time of another task τ_i considering a direct-mapped cache,

$$\rho^{dir}_{j,i} = d_{mem} \times \left| PCB_j \cap \Big(\bigcup_{\forall \tau_k \in \mathrm{hep}(i) \setminus \tau_j} ECB_k \Big) \right| \quad (4)$$

where PCB_j is the set of PCBs of τ_j and $\bigcup_{\forall \tau_k \in \mathrm{hep}(i) \setminus \tau_j} ECB_k$ is the set of ECBs of all tasks $\in \mathrm{hep}(i) \setminus \tau_j$. For a formal proof of Eq. (4) see [3]. It is proved in [3] that the set of PCBs of τ_j, i.e., PCB_j, upper bound the CPRO τ_j may suffer. We can easily extend that concept to set-associative caches by observing that the number of PCBs of τ_j in a cache set s, i.e., $|PCB^s_j|$, upper bounds the CPRO τ_j may suffer due to s. Let $CPRO^s_j$ denote the CPRO τ_j may suffer in cache set s, then $CPRO^s_j = |PCB^s_j|$, i.e., the total number of PCBs of τ_j in cache set s.



(a) SoA resilience Analysis (b) ResilienceP analysis

Fig. 1: Overestimation in the SoA resilience analysis

From [3], we also know that the worst-case impact of all tasks in $\mathrm{hep}(i) \setminus \tau_j$ on PCBs of τ_j is bounded by the set of all ECBs of all tasks in $\mathrm{hep}(i) \setminus \tau_j$ (See Eq. 4). Hence, the worst-case impact of all task in $\mathrm{hep}(i) \setminus \tau_j$ on PCBs of τ_j in a cache set s can be upper-bounded by $CPRO^s_{\mathrm{hep}(i)\setminus\tau_j}$, where

$$CPRO^s_{\mathrm{hep}(i)\setminus\tau_j} = \begin{cases} k & \text{if } \bigcup_{\forall \tau_k \in \mathrm{hep}(i)\setminus\tau_j} ECB^s_k \neq \emptyset \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

Consequently, the CPRO of task τ_j in cache set s is bounded by $\rho^{set,s}_{j,i}$, where

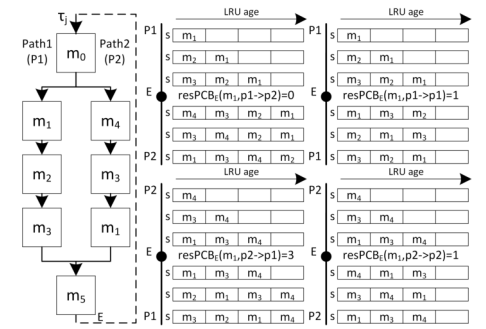
$$\rho^{set,s}_{j,i} = d_{mem} \times \min\Big(CPRO^s_j, CPRO^s_{\mathrm{hep}(i)\setminus\tau_j}\Big) \quad (6)$$

and the total CPRO one job of τ_j may suffer during the response time of τ_i is thus given by $\rho^{set}_{j,i} = \sum_{s=0}^{cs} \rho^{set,s}_{j,i}$.

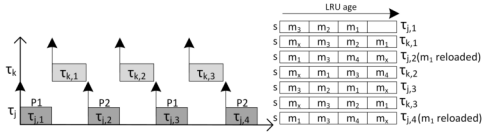
IV. ResilienceP Analysis

The PCB-ECB approach presented in Section III assumes that if one ECB of any task $\tau_k \in \mathrm{hep}(i) \setminus \tau_j$ is mapped to a cache set s then all the PCBs of τ_j in s will be evicted. This assumption is safe but very pessimistic. Therefore, to have a tighter bound on the CPRO, in this section we determine the set of PCBs of task τ_j that may remain cached even after preemptions/executions of tasks $\in \mathrm{hep}(i) \setminus \tau_j$ thanks to the resilience of τ_j's PCBs. However, we first note that the SoA resilience analysis [7] cannot be used as is to calculate the resilience of PCBs. To illustrate, see Fig. 1a showing the control-flow graph (CFG) and mapping of memory blocks of two jobs of task τ_j, i.e., $\tau_{j,1}, \tau_{j,2}$, in a 4-way set associative cache. We assume that $\{m_1, m_2, m_3, m_4\}$ are all PCBs of τ_j. Using the SoA resilience analysis that only considers the execution of one job of τ_j, i.e., $\tau_{j,1}$, it results that the resilience of PCB m_1, m_2, m_3 and m_4 is 0, 1, 2, and 3 respectively (see Fig. 1a). However, these resilience bounds are not sound considering that PCB m_1, m_2, m_3 and m_4 are reused only during the execution of the next job of τ_j, i.e., $\tau_{j,2}$. In fact, the maximum-age of all these PCBs across two jobs of τ_j is 3 which leads to a resilience of 0 for all the PCBs (See Eq. (1)).

The ResilienceP analysis accounts for the overestimated resilience of PCBs in the existing resilience analysis by calculating the maximum-age of PCBs over all job executions of τ_j. This is done by assuming that τ_j is cyclic, i.e., a loop between the end point E and start point S of τ_j (e.g., see Fig. 1b). The cyclic assumption ensures that the maximal number of different cache accesses between the last use of m_j in one job of τ_j and the first access of m_j in the next job of τ_j are considered when determining the maximum-age of m_j. Moreover, knowing that PCBs are calculated at task



(a) Variation in the resilience of PCBs of task τ_i



(b) Different job executions of τ_j and τ_k

Fig. 2: Highlighting the pessimism in ResilienceP analysis

level [3] in contrast to UCBs (calculated per program point) and the evictions of cache blocks in $UCB \cap PCB$ are already accounted for in the CRPD cost, the resilienceP analysis only calculates the maximum-age of PCBs at the end point E of a task τ_j using the same approach as proposed in [7]. Formally, under the resilienceP analysis the maximum-age of a PCB m_j is given by $max\text{-}age(m_j) = max\text{-}age_E(m_j)$, and the resilience of PCB m_j is given by $res_{PCB}(m_j) = (k-1) - max\text{-}age(m_j)$. Consequently, the total CPRO of one job of task τ_j executing during the response time of τ_i is bounded by $\rho^{res}_{j,i} = \sum_{s=1}^{cs} \rho^{res,s}_{i,j}$, where

$$\rho^{res,s}_{j,i} = d_{mem} \times \left| PCB^s_j \setminus \left\{ m_j | res_{PCB}(m_j) \geq \sum_{\forall \tau_k \in \mathrm{hep}(i) \setminus \tau_j} |ECB^s_k| \right\} \right| \quad (7)$$

V. Multiset alike ResilienceP Analysis

The resilienceP analysis always considers the worst-case (i.e., minimum) resilience of PCBs for all jobs of τ_j that may execute in a time interval of length t. This is true if τ_j only has a single execution path as in Fig. 1b. However, if τ_j has multiple execution paths, the resilience of PCBs may vary depending on the actual execution paths taken by two successive jobs of τ_j. Therefore, always considering the minimum resilience of PCBs over all job executions of τ_j may overestimate the total CPRO τ_j may suffer. To illustrate this, see Fig. 2a that shows the CFG of a task τ_j with two execution paths and four possible execution flows between two jobs of τ_j, i.e., $p1 \rightarrow p2$, $p2 \rightarrow p1$, $p1 \rightarrow p1$ and $p2 \rightarrow p2$. The cache contents along each execution flow are also shown in Fig. 2a. We assume that all memory blocks of τ_j except m_0 and m_5 map to the same cache set s of a 4-way set-associative cache. For clarity, we only focus on PCB m_1.

We can see in Fig. 2a that the resilience of m_1 is minimum, i.e., $res_{PCB}(m_1) = 0$, if first job of τ_j follows path $p1$ and the next job follow path $p2$. Now consider the example schedule shown in Fig. 2b showing four jobs of τ_j along with three jobs of a task $\tau_k \in \mathrm{hep}(i) \setminus \tau_j$ such that $ECB^s_k = \{m_x\}$. Fig. 2b also shows the contents of cache set s after the execution of every job of τ_j and τ_k.

TABLE I: CPRO-table for every PCB m_j of task τ_j

Disturbance (D) \ Number of jobs of τ_j (J)	2	3	...	$\left\lceil \frac{t}{T_j} \right\rceil$
1	$\min(1,x)$	$\min(2,x)$	...	$\min(\left\lceil \frac{t}{T_j} \right\rceil - 1, x)$
2	$\min(1,x)$	$\min(2,x)$	...	$\min(\left\lceil \frac{t}{T_j} \right\rceil - 1, x)$
...	...	...	...	...
$\geq k$	1	2	...	$\left\lceil \frac{t}{T_j} \right\rceil - 1$

As the minimum resilience of m_1 is 0 and $|ECB^s_k| > res_{PCB}(m_1)$, the resilienceP analysis (i.e., Eq. (7)) implies that every time τ_k preempts τ_j or executes between two subsequent jobs of τ_j, m_1 will be evicted. This results in a CPRO of $d_{mem} \times 3$. However, we can see in Fig. 2b that this is not true. In fact even in the worst-case when we maximize jobs of τ_j following the execution flow with the minimum resilience (i.e., $p_1 \rightarrow p_2$), m_2 is evicted and reloaded only two times resulting in a CPRO of $d_{mem} \times 2$.

The multi-set alike ResilienceP analysis reduces the pessimism in the ResilienceP analysis by considering the variation in the resilience of PCBs across different job execution of a task τ_j. For each PCB m_j of τ_j we create a CPRO-table (See Table I) to determine how many times m_j can be evicted in an interval of length t considering a given disturbance D, i.e., the total number of ECBs of tasks in $\mathrm{hep}(i) \setminus \tau_j$. Given the value of D and J, one entry in Table I (i.e., x) for a PCB m_j tells us how many times m_j may be evicted and must therefore be reloaded.

In future, we will investigate how to efficiently build Table I and evaluate our solutions.

Acknowledgments. This work was partially supported under PhD grant SFRH/BD/119150/2016, by National Funds through FCT/MCTES (Portuguese Foundation for Science and Technology), within the CISTER Research Unit (UID/CEC/04234); 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 POCI-01-0145-FEDER-029119 (PREFECT); by the European Union through the Clean Sky 2 Joint Undertaking, under the H2020 Framework Programme (H2020-CS2-CFP08-2018-01), grant agreement nr. 832011 (THERMAC).

References

[1] 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.

[2] M. Lv, N. Guan, J. Reineke, R. Wilhelm, and W. Yi, "A survey on static cache analysis for real-time systems," *Leibniz Transactions on Embedded Systems*, vol. 3, no. 1, pp. 05–1, 2016.

[3] S. A. Rashid, G. Nelissen, D. Hardy, B. Akesson, I. Puaut, and E. Tovar, "Cache-persistence-aware response-time analysis for fixed-priority pre-emptive systems," in *ECRTS*, 2016, pp. 262–272.

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

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

[6] 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.

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