Cache Persistence-Aware Memory Bus Contention Analysis for Multicore Systems

Syed Aftab Rashid

Geoffrey Nelissen

Eduardo Tovar

CISTER-TR-191102

Cache Persistence-Aware Memory Bus Contention Analysis for Multicore Systems

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:

https://www.cister-labs.pt

Abstract

Memory bus contention strongly relates to the numberof main memory requests generated by tasks running ondifferent cores of a multicore platform, which, in turn, depends onthe content of the cache memories during the execution of thosetasks. Recent works have shown that due to cache persistence thememory access demand of multiple jobs of a task may not alwaysbe equal to its worst-case memory access demand in isolation.Analysis of the variable memory access demand of tasks due tocache persistence leads to significantly tighter worst-case responsetime (WCRT) of tasks.In this work, we show how the notion of cache persistence canbe extended from single-core to multicore systems. In particular,we focus on analyzing the impact of cache persistence on thememory bus contention suffered by tasks executing on a multicoreplatform considering both work conserving and non-workconserving bus arbitration policies. Experimental evaluationshows that cache persistence-aware analyses of bus arbitrationpolicies increase the number of task sets deemed schedulableby up to 70 percentage points in comparison to their respectivecounterparts that do not account for cache persistence.

Cache Persistence-Aware Memory Bus Contention Analysis for Multicore Systems

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

***Abstract*—Memory bus contention strongly relates to the number of main memory requests generated by tasks running on different cores of a multicore platform, which, in turn, depends on the content of the cache memories during the execution of those tasks. Recent works have shown that due to *cache persistence* the memory access demand of multiple jobs of a task may not always be equal to its worst-case memory access demand in *isolation*. Analysis of the *variable* memory access demand of tasks due to cache persistence leads to significantly tighter worst-case response time (WCRT) of tasks.**

In this work, we show how the notion of cache persistence can be extended from single-core to multicore systems. In particular, we focus on analyzing the impact of cache persistence on the memory bus contention suffered by tasks executing on a multicore platform considering both work conserving and non-work conserving bus arbitration policies. Experimental evaluation shows that cache persistence-aware analyses of bus arbitration policies increase the number of task sets deemed schedulable by up to 70 percentage points in comparison to their respective counterparts that do not account for cache persistence.

I. Introduction

Ensuring the timing correctness of multicore systems is considerably more complex than for single-core ones. This is mainly because of the difficulty to bound the amount of *cross-core interference* on shared resources, e.g., last-level caches (LLC), common interconnect (i.e., memory bus) and DRAM memory, generated by tasks concurrently running on different cores of a multicore system. For example, the latency of a read operation from the main memory in Freescale P4080 varies between 40 to 600 cycles depending on the number of tasks competing to access that memory [1]. Since data/instructions are transferred from the main memory to the requesting core over a memory bus and because the memory bus is shared between cores, requests by a task τ_i running on one core may be delayed by tasks executing on other cores, thereby increasing its WCRT. This increase in the WCRT depends on many factors such as (i) the number of main memory requests generated by τ_i and all other tasks running on the same core, (ii) the number of main memory requests generated by all tasks executing on different cores than τ_i and (iii) the memory bus arbiter. One of the main aspects that impact (i) and (ii) is the number of *cache misses* suffered by each task during its execution. Indeed, the number of main memory requests generated by a task strongly depends on whether the instructions and data it requires are available in the cache memory (cache hit) or not (cache miss). If all data/instructions used by a task τ_i are already available in the cache, τ_i may not access main memory, effectively suffering no memory bus contention. However, if the data/instructions required by τ_i are not available in the cache (i.e., due to limited cache space or evictions by other tasks), τ_i will access main memory and hence τ_i may cause and be subjected to bus contention.

Under fixed-priority preemptive scheduling (FPPS), the number of main memory accesses by a task τ_i may increase due to preemptions by higher priority tasks. This happens when cache blocks used by τ_i are evicted from the cache due to preemptions by higher priority tasks. Upon resumption, τ_i may need to reload the evicted cache blocks from the main memory which results in increasing the number of main memory accesses. The delay caused by these additional main memory accesses is called Cache-Related Preemption Delay (CRPD). There exist approaches in literature that account for CRPDs [2] when bounding the memory bus contention in multicore systems. Recent works [3], [4] have also shown that the use of caches may reduce the worst-case main memory access demand of tasks due to *cache persistence*. Cache persistence refers to the re-use of cache blocks between different job executions of the same task which may result in reducing its number of accesses to the main memory.

In this work, we extend the analysis in [3], [4] from single-core to multicore platforms. The three main contributions of this work are: 1) we extend the notion of cache persistence between different jobs of a task to multicore systems. This allows us to capture the *reduction* in the number of main memory accesses by tasks executing on different cores due to re-use of cache content between multiple executions of the same task. 2) We evaluate the impact of cache persistence on memory bus contention in multicore platforms considering both work conserving and non-working conserving bus arbitration policies. 3) We present an experimental evaluation that shows that cache persistence-aware analyses of bus arbitration policies increase the number of task sets deemed schedulable by up to 70 percentage points in comparison to their counterparts that do not consider cache persistence.

II. System Model

We consider a multicore platform with m identical timing-compositional cores π_1 to π_m. By timing-compositional we mean that it is safe to separately account for interference from different sources such as cores, caches and memory bus [5]. Each core has a local direct-mapped instruction cache using the Least-Recently-Used (LRU) replacement policy. The cache is connected via a shared bus to the global main memory. Note that we assume that data accesses to memory are performed via a separate bus. Moreover, since we only consider single-level caches the proposed analysis is independent of the cache inclusion policy. The worst-case time for one access to the main memory is denoted by d_{mem}. We consider a set of n

sporadic constrained deadline tasks $\{\tau_1, \tau_2, ...\tau_n\}$. Each task τ_i is defined by a quadruple (PD_i, MD_i, D_i, T_i) where PD_i is the worse-case execution time of a job 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 main memory. MD_i is the worst-case memory access demand of a job of τ_i, i.e., the maximum number of main memory request generated by any job of τ_i. Note that the values of PD_i and MD_i are calculated assuming τ_i executes in *isolation*. D_i is the relative deadline of τ_i and T_i is the minimum-inter arrival time between two jobs of τ_i. We assume that tasks are scheduled with a partitioned task-level fixed priority scheduling algorithm where each task is statically assigned to a core at design time. Tasks assigned to a core π_x are denoted by Γ_x. Tasks can be assigned priorities using any fixed priority assignment scheme (e.g., Rate or Deadline Monotonic [6]). Furthermore, we assume that the priority of each task is unique thus providing a global priority order such that τ_1 has the highest priority and τ_n the lowest. R_i denotes the WCRT of task τ_i and is defined as the longest time between the arrival and the completion of any of its jobs. For notational convenience, we use $\mathrm{hp}(i)$ and $\mathrm{lp}(i)$ to denote the set of tasks with priorities higher, respectively lower, than that of τ_i. We use $\mathrm{hep}(i)$ and $\mathrm{aff}(i,j)$ as short notations for $\mathrm{hep}(i) = \mathrm{hp}(i) \cup \{i\}$, and $\mathrm{aff}(i,j) = \mathrm{hep}(i) \cap \mathrm{lp}(j)$. 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.

III. Background

The total number of bus accesses that may affect the execution of a task $\tau_i \in \Gamma_x$ in a time interval of length t depends mainly on two factors; (i) the total number of bus accesses $BAS_i^x(t)$ that can occur due to task τ_i and all higher priority tasks in $\mathrm{hp}(i)$ executing on core π_x during that time interval, and (ii) the total number of bus accesses generated by all tasks running on other cores than π_x during the same time interval of length t. It is proved in [2] that

$$BAS_i^x(t) \leq MD_i + \sum_{\forall \tau_j \in \Gamma_x \cap \mathrm{hp}(i)} \left\lceil \frac{t}{T_j} \right\rceil \times (MD_j + \gamma_{i,j,x}) \qquad (1)$$

where $\gamma_{i,j,x}$ denotes the CRPD suffered by task τ_i due to preemptions by higher priority task τ_j executing on the same core π_x. In this work, $\gamma_{i,j,x}$ is calculated as in [2] using the ECB-union approach [7]. The ECB-union approach provides a reasonably precise bound on the CRPD using the notion of *useful* and *evicting* cache blocks (i.e., UCBs and ECBs). Formally, "a memory block m is called a useful cache block at program point P, if it is cached at P and will be reused at program point Q that may be reached from P without eviction of m [8]". Similarly, all cache blocks used by the task during its execution are called evicting cache blocks (ECBs) [8]. The ECB-union approach uses the set of UCBs of tasks that may be preempted by τ_j during the response time of τ_i, i.e., $\mathrm{aff}(i,j)$ and the union of the ECBs of all tasks in $\mathrm{hep}(j)$, i.e., it assumes τ_j itself is preempted by all higher priority tasks. Under the ECB-union approach, $\gamma_{i,j,x}$ is then given by

$$\gamma_{i,j,x} = \max_{\forall g \in \Gamma_x \cap \mathrm{aff}(i,j)} \left\{ \left| UCB_g \cap \left(\bigcup_{\forall h \in \Gamma_x \cap \mathrm{hep}(j)} ECB_h \right) \right| \right\} \qquad (2)$$

Detailed description and proof for Eq. (2) can be found in [7].

When bounding the total number of bus accesses generated by all tasks running on cores other than π_x, no assumption can be made about the synchronization of tasks w.r.t the release of τ_i on core π_x. Therefore, for a task τ_l executing on some core $\pi_y \neq \pi_x$, the worst-case number of bus accesses generated by τ_l in a time interval of length t are obtained by assuming that the first job of τ_l executes as late as possible, i.e., just before its WCRT, while all subsequent jobs of τ_l execute as early as possible. Let $BAO_k^y(t)$ be an upper bound on the total number of bus accesses due to all tasks having priority k or higher executing on core π_y. It is proven in [2] that

$$BAO_k^y(t) \leq \sum_{\tau_l \in \Gamma_y \cap \mathrm{hep}(k)} W_{k,l}^y(t) + W_{k,l,cout}^y(t) \qquad (3)$$

with

$$W_{k,l}^y(t) = N_{k,l}^y(t) \times (MD_l + \gamma_{k,l,y}) \qquad (4)$$

$$W_{k,l,cout}^y = \min \left(\left\lceil \frac{t + R_l - (MD_l + \gamma_{k,l,y}) \times d_{mem} - N_{k,l}^y(t) \times T_l}{d_{mem}} \right\rceil \right.$$
$$\left. ; MD_l + \gamma_{k,l,y} \right) \qquad (5)$$

and where $N_{k,l}^y(t)$ is an upper bound on the maximum number of jobs of τ_l that may fully execute in an interval of length t on core π_y, i.e.,

$$N_{k,l}^y(t) = \left\lfloor \frac{t + R_l - (MD_l + \gamma_{k,l,y}) \times d_{mem}}{T_l} \right\rfloor \qquad (6)$$

Using Eq. (1) and (3), the total number of bus accesses BAT_i^x that may delay the execution of τ_i on core π_x can be bounded for Fixed-Priority (FP), Round-Robin (RR) and TDMA bus arbitration policies as presented below.

For a FP bus, where bus accesses inherit the priority of the task that generate them, $BAT_i^x(t)$ is given by

$$BAT_i^x(t) = BAS_i^x(t) + \sum_{\forall \pi_y \neq \pi_x} BAO_i^y(t) + 1 \qquad (7)$$
$$+ \min \left(BAS_i^x(t); \sum_{\forall \pi_y \neq \pi_x} BAO_{i,low}^y(t) \right)$$

where $BAO_{i,low}^y(t) = \sum_{\forall \tau_l \in \Gamma_y \cap \mathrm{lp}(i)} W_{i,l}^y(t) + W_{i,l,cout}^y$, i.e., to account for bus accesses from tasks having a lower priority than τ_i.

Similarly, for a RR bus, $BAT_i^x(t)$ is upper bounded by

$$BAT_i^x(t) = BAS_i^x(t) + \sum_{\forall \pi_y \neq \pi_x} \min \left(BAO_n^y(t); s \times BAS_i^x(t) \right) + 1 \qquad (8)$$

where s denotes the number of memory access slots per core and n is the index of the task with the lowest priority. Under a TDMA bus, $BAT_i^x(t)$ is given by

$$BAT_i^x(t) = BAS_i^x(t) + ((L-1) \times s) \times BAS_i^x(t) + 1 \qquad (9)$$

where the length of one TDMA cycle is $L \times s$. Detailed description of Eq. (1) and Eq. (3)-(9) can be found in [2].

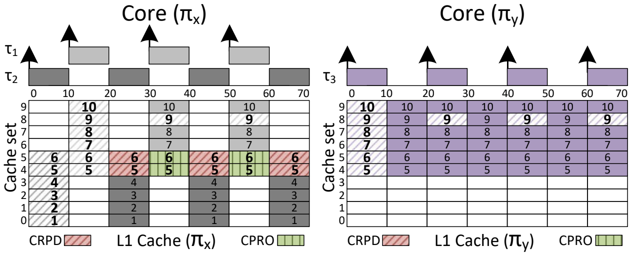


Fig. 1: Execution of task τ_1 and τ_2 on core π_x and task τ_3 on core π_y. Task parameters of interest are: PD_1=PD_3 = 4, PD_2= 32, MD_1=MD_3 = 6, MD_2 = 8, MD_1^r=MD_3^r = 1, ECB_1=ECB_3 = $\{5, 6, 7, 8, 9, 10\}$, ECB_2 = $\{1, 2, 3, 4, 5, 6\}$, PCB_1=PCB_3 = $\{5, 6, 7, 8, 10\}$ and UCB_2 = $\{5, 6\}$.

IV. Persistence-aware Bus Contention analysis

The analysis presented in [2] (i.e., Eq. (1) and (3)) provides a safe upper bound on the memory bus delay suffered by tasks executing on a multicore platform. However, since the analysis in [2] does not consider the variation in the memory access demand of tasks due to cache persistence, it may overestimate the actual number of accesses that compete for the bus during the response time of the task under analysis. Cache persistence refers to the re-use of *persistent cache blocks* (PCBs) between different job executions of the same task. PCBs are defined as [3] "memory blocks used by a task that, once loaded in the cache, will never be evicted or invalidated by the task itself" (see [3] for details on how to find PCBs of a task). If all PCBs of a task τ_i were loaded in the cache by a previous job of τ_i, the memory access demand of all subsequent jobs of τ_i can be much lower than the worst-case memory access demand of τ_i in isolation. This type of memory demand is called the *residual memory access demand* MD_i^r of τ_i and is defined as [3] "the worst-case memory access demand of any job of a task when assuming all its PCBs are already loaded in the cache". According to this definition, a PCB is loaded only once from the main memory when a task τ_i executes in isolation. Therefore, the total number of bus accesses generated by n_i jobs of τ_i executing in isolation is given by

$$\hat{MD}_i(n_i) = \min\left(n_i \times MD_i \;; n_i \times MD_i^r + |PCB_i|\right) \tag{10}$$

We now consider the schedule and task parameters shown in Fig. 1 to show how Eq. (10) may be used to reduce the pessimism of [2]. We have three tasks τ_1, τ_2 and τ_3 with τ_1 and τ_2 executing on core π_x and τ_3 executing on core π_y. We assume τ_1 has the highest priority and τ_3 the lowest. The worst-case main memory access demands MD_1, MD_2 and MD_3 are 6, 8 and 6 respectively. Memory blocks in Fig. 1 that are pattern filled are those that are loaded/reloaded from the main memory during the task executions. We focus on τ_2 and use $BAT_2^x(R_2)$ to denote the total number of bus accesses that may be generated during its response time. Assuming the memory bus arbitration policy is RR with a slot size s equal to 1, $BAT_2^x(R_2)$ can be bounded using Eq. (8) such that

$$BAT_2^x(R_2) = BAS_2^x(R_2) + \min\left(BAO_3^y(R_2); BAS_2^x(R_2)\right) \tag{11}$$

where

$$BAS_2^x(R_2) = MD_2 + 3 \times (MD_1 + \gamma_{2,1,x}) = 8 + 3 \times (6 + 2) = 32 \tag{12}$$

$$BAO_3^y(R_2) = N_{3,3}^y(R_2) \times MD_3 = 4 \times (6) = 24 \tag{13}$$

Note that $\gamma_{2,1,x}$ in Eq. (12) is derived using Eq. (2). Moreover, since τ_2 is the lowest priority task on core π_x, Eq. (12) does not have a trailing +1 as in Eq. (8) (see [2] for details).

However, by comparing the result of Eq. (12) with the cache contents of core π_x in Fig. 1 we can see that Eq. (12) overestimates the value of $BAS_2^x(R_2)$. Fig. 1 shows that only the first job of τ_1 needs to load all its ECBs from the main memory and hence has a worst-case memory access demand $MD_1 = 6$. Moreover, since all PCBs of τ_1 were loaded in the cache by the first job of τ_1, the memory access demand of the next two jobs of τ_1 only corresponds to the reloading of memory block $\{9\}$, i.e., $MD_1^r = 1$. Consequently, the actual number of memory accesses made by the three jobs of τ_1 executing during the response time of τ_2 are respectively $MD_1 + MD_1^r + MD_1^r = 6 + 1 + 1 = 8$, which is much lower than $3 \times MD_1 = 18$ accounted for in Eq. (12).

Fig. 1 also shows an overlap between PCBs $\{5, 6\}$ of τ_1 and ECBs $\{5, 6\}$ of τ_2 in cache. This overlap may lead to evictions of PCBs $\{5, 6\}$ between two subsequent executions of τ_1. By definition a task can not evict its own PCBs. However, PCBs of a task can be evicted due to interleaved or preemptive execution of other tasks in the system, leading to what is called *cache persistence reload overhead* (CPRO). CPRO of a task $\tau_j \in \mathrm{hp}(i)$ executing during the response time of τ_i on core π_x is formally defined as "the overhead suffered by a task τ_j during the response time of another task τ_i due to evictions of its PCBs by tasks in $\mathrm{hep}(i) \setminus \tau_j$ [3]". CPRO can be calculated using any of the approaches presented in [3], [4]. In this work we use the CPRO-union approach [3]. The additional number of bus accesses generated by n_j successive jobs of τ_j due to CPRO is upper bounded by

$$\hat{\rho}_{j,i,x}(n_j) = (n_j - 1) \times \left|PCB_j \cap \Big(\bigcup_{\forall \tau_s \in \Gamma_x \cap \mathrm{hep}(i) \setminus \tau_j} ECB_s\Big)\right| \tag{14}$$

where PCB_j is the set of PCBs of task τ_j (that may be evicted) and $\bigcup_{\forall \tau_s \in \mathrm{hep}(i) \setminus \tau_j} ECB_s$ is the union of ECBs of all tasks that may evict PCBs of τ_j. See [3] for a detailed description of Eq. (10) and (14).

Consequently, for the schedule in Fig. 1, the additional bus accesses due to CPRO suffered by task τ_1 during the response of task τ_2 are 4, i.e., $\hat{\rho}_{1,2,x}(3) = 2 \times 2 = 4$. Therefore, due to cache persistence, the actual number of bus accesses during the response time task τ_2 on core π_x are given by

$$MD_2 + MD_1 + 2 \times MD_1^r + \hat{\rho}_{1,2,x}(3) + 3 \times \gamma_{2,1,x} = 26 \tag{15}$$

which is much lower than the value of $BAS_2^x(R_2) = 32$ calculated using Eq. (12). This leads to the following lemma.

Lemma 1. The total number of bus accesses by a single job of task $\tau_i \in \Gamma_x$ and all higher priority tasks in $\Gamma_x \cap \mathrm{hp}(i)$ executing in a time interval of length t are upper bounded by $\hat{BAS}_i^x(t)$, where

$$\hat{BAS}_i^x(t) = \sum_{\forall \tau_j \in \Gamma_x \cap \mathrm{hp}(i)} \min\Big(E_j(t) \times MD_j; \hat{MD}_j(E_j(t)) + \hat{\rho}_{j,i,x}(E_j(t))\Big) + \sum_{\forall \tau_j \in \Gamma_x \cap \mathrm{hp}(i)} E_j(t) \times \gamma_{i,j,x} + MD_i \text{ with } E_j(t) = \left\lceil \frac{t}{T_j} \right\rceil \tag{16}$$

Proof. We prove that in a time interval of length t, $B\hat{A}S_i^x(t)$ is an upper bound on the total number of bus accesses generated by task $\tau_i \in \Gamma_x$ and all higher priority tasks in $\Gamma_x \cap \mathrm{hp}(i)$.
1. By assumption, only one job of τ_i must be considered. Hence, the total number of bus accesses generated by τ_i are upper bounded by its worst-case memory access demand MD_i.
2. Any task $\tau_j \in \Gamma_x \cap \mathrm{hp}(i)$ can release at most $E_j(t) = \left\lceil \frac{t}{T_j} \right\rceil$ jobs in a time window of length t. Therefore, it follows from Eq. (1) that $E_j(t) \times MD_j$ is an upper bound on the total number of bus accesses generated by task $\tau_j \in \Gamma_x \cap \mathrm{hp}(i)$ in isolation. Moreover, the additional bus accesses due to preemptions of task τ_i by task τ_j in a time interval of length t are upper bounded by $E_j(t) \times \gamma_{i,j,x}$ (see Eq. 1). Hence, the sum $MD_i + E_j(t) \times MD_j + E_j(t) \times \gamma_{i,j,x}$ is an upper bound on the total number of bus accesses generated by task $\tau_i \in \Gamma_x$ and any higher priority task τ_j in $\Gamma_x \cap \mathrm{hp}(i)$ in a time interval of length t.
3. Recall from Eq. (10) and (14) that $\hat{MD}_j(E_j(t))$ is an upper bound on the total number of bus accesses due to $E_j(t)$ jobs of τ_j executing in isolation and $\hat{\rho}_{j,i,x}(E_j(t))$ is an upper bound on the additional bus accesses due to CPRO suffered by all those jobs of τ_j. Therefore, the sum $MD_i + \hat{MD}_j(E_j(t)) + \hat{\rho}_{j,i,x}(E_j(t)) + E_j(t) \times \gamma_{i,j,x}$ is also an upper bound on the total number of bus accesses generated by task $\tau_i \in \Gamma_x$ and any higher priority task τ_j in $\Gamma_x \cap \mathrm{hp}(i)$ in a time window of length t considering both CRPD and CPRO. Thus, the minimum between $MD_i + E_j(t) \times MD_j + E_j(t) \times \gamma_{i,j,x}$ and $MD_i + \hat{MD}_j(E_j(t)) + \hat{\rho}_{j,i,x}(E_j(t)) + E_j(t) \times \gamma_{i,j,x}$ is also an upper bound. The lemma follows. □

Continuing the example depicted in Fig. 1, we can see that Eq. (13) also overestimates the value of $BAO_3^y(R_2)$. In fact, due to cache persistence, the actual number of bus accesses generated by task $\tau_3 \in \Gamma_y$ that may contend for the bus during the response time of task $\tau_2 \in \Gamma_x$ are: $MD_3 + 3 \times MD_3^r = 6 + 3 \times 1 = 9$, which is much lower than the value of $BAO_3^y(R_2) = 24$ calculated using Eq. (13).

Lemma 2. The total number of bus accesses by all tasks $\in \Gamma_y$ with priority k or higher that may contend for bus access with task $\tau_i \in \Gamma_x$ during a time window of length t is upper bounded by

$$B\hat{A}O_k^y(t) = \sum_{\forall \tau_l \in \Gamma_y \cap \mathrm{hep}(k)} \hat{W}_{k,l}^y(t) + W_{k,l,cout}^y \tag{17}$$

where

$$\hat{W}_{k,l}^y(t) = \min\Big(N_{k,l}^y(t) \times MD_l ; \hat{MD}_l(N_{k,l}^y(t)) + \hat{\rho}_{k,l,y}(N_{k,l}^y(t))\Big) + N_{k,l}^y(t) \times \gamma_{k,l,y} \tag{18}$$

and $W_{k,l,cout}^y$ and $N_{k,l}^y(t)$ are given by Eq. (5) and (6).

Proof. Since Eq. (5) (i.e., $W_{k,l,cout}^y$) is proved in [2], we only need to prove that $\hat{W}_{k,l}^y(t)$ is an upper bound on the total number of bus accesses by jobs of $\tau_l \in \Gamma_y \cap \mathrm{hep}(k)$ that fully execute in a time interval of length t.
1. It follows from Eq. (6) that $N_{k,l}^y(t)$ is an upper bound on the number of jobs that may be fully executed in an interval of length t by any task $\tau_l \in \Gamma_y \cap \mathrm{hep}(k)$. Therefore, $N_{k,l}^y(t) \times MD_l$ upper bounds the total number of bus accesses generated by τ_l in a time interval of length t in isolation. Moreover, $N_{k,l}^y(t) \times \gamma_{k,l,y}$ is an upper bound on the additional bus accesses due to CRPD suffered by task $\tau_l \in \Gamma_y \cap \mathrm{hep}(k)$ in a time window of length t. Therefore, the sum $N_{k,l}^y(t) \times MD_l + N_{k,l}^y(t) \times \gamma_{k,l,y}$ is an upper bound on the total number of bus accesses by jobs of $\tau_l \in \Gamma_y \cap \mathrm{hep}(k)$ that fully execute in a time interval of length t.
2. Using $N_{k,l}^y(t)$ in Eq. (10) and (14) we get $\hat{MD}_l(N_{k,l}^y(t))$ which is an upper bound on the total number of bus accesses due to $N_{k,l}^y(t)$ successive jobs of τ_j executing in isolation and $\hat{\rho}_{k,l,y}(N_{k,l}^y(t))$ which is an upper bound on the additional bus accesses due to CPRO suffered by all those jobs. Hence, the sum $\hat{MD}_l(N_{k,l}^y(t)) + \hat{\rho}_{k,l,y}(N_{k,l}^y(t)) + N_{k,l}^y(t) \times \gamma_{k,l,y}$ is also an upper bound on the total number of bus accesses by jobs of $\tau_l \in \Gamma_y \cap \mathrm{hep}(k)$ that fully execute in a time interval of length t considering both CRPD and CPRO. Consequently, the minimum between $N_{k,l}^y(t) \times MD_l + N_{k,l}^y(t) \times \gamma_{k,l,y}$ and $\hat{MD}_l(N_{k,l}^y(t)) + \hat{\rho}_{k,l,y}(N_{k,l}^y(t)) + N_{k,l}^y(t) \times \gamma_{k,l,y}$ is also an upper bound. The lemma follows. □

The worst-case response time R_i of a task $\tau_i \in \Gamma_x$ is given by the smallest possible solution of the following expression

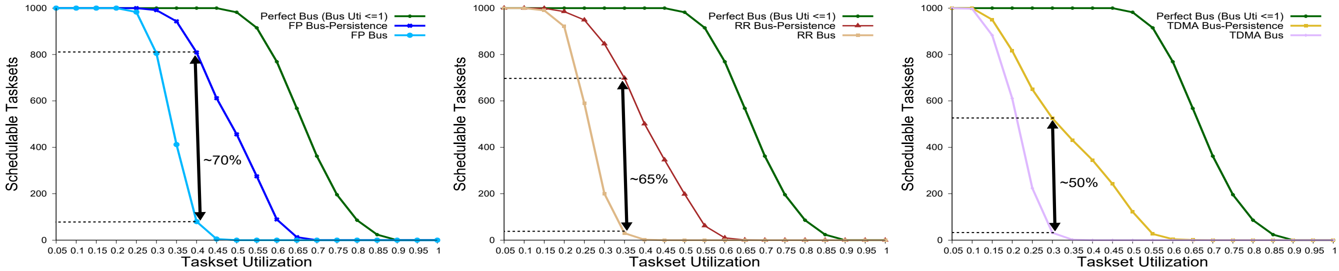
$$R_i = PD_i + \sum_{\forall \tau_j \in \Gamma_x \cap \mathrm{hp}(i)} \left\lceil \frac{R_i}{T_j} \right\rceil \times PD_j + BAT_i^x(R_i) \times d_{mem} \tag{19}$$

The term $\sum_{\forall \tau_j \in \Gamma_x \cap \mathrm{hp}(i)} \left\lceil \frac{R_i}{T_j} \right\rceil \times PD_j$ upper bounds the total core interference suffered by task τ_i due to preemptions by higher priority tasks executing on the same core, whereas the total memory bus interference that τ_i may suffer during R_i is upper bounded by $BAT_i^x(R_i) \times d_{mem}$. Depending on the bus arbitration policy, $BAT_i^x(R_i)$ can be calculated using Eq. (7), (8) or (9) with $BAS_i^x(t)$ and $BAO_k^y(t)$ computed using Lemma 1 and Lemma 2, respectively. Note that since the response time of each task may depend on the response times of other tasks, we use an outer loop around a set of fixed-point iterations to compute the response times of all the tasks, and so deal with the apparent circular dependency. The iteration on R_i starts by initiating R_i to $PD_i + MD_i \times d_{mem}$ and stops as soon as R_i does not evolve anymore or $R_i > D_i$.

V. Experimental Evaluation

In this section, we compare the performance of FP, RR and TDMA bus arbitration policies in terms of task set schedulability with and without considering cache persistence. We model a multicore platform with 4 cores each having a private L1 instruction cache with 256 cache sets and a block size of 32 Bytes. The default value of d_{mem} is $5\mu s$. All experiments were performed using the Mälardalen benchmark suite [9] with parameters PD_i, MD_i, MD_i^r, UCB_i, ECB_i and PCB_i extracted using the Heptane static WCET analysis tool [3], [10]. Due to space constraints, Table I only shows details of some of the benchmarks, with PD_i, MD_i and MD_i^r given in clock cycles. A table with all benchmarks can be found in [4]. Note that our simulator is available on demand.

The default task set size was 32 with 8 tasks per core. Each task within the task set is randomly assigned parameters from one of the benchmarks of the Mälardalen benchmark suite. Task utilizations were generated using UUnifast [11] assuming an equal utilization for each core. Task periods and



(a) FP Bus with/without cache Peristence

(b) RR Bus with/without cache Peristence

(c) TDMA Bus with/without cache Peristence

Fig. 2: Schedulability ratio of different bus arbitration policies by varying total core utilizations

TABLE I: Task parameters for a selection of benchmarks from the Mälardalen Benchmark Suite [9]

Name	$\boldsymbol{PD_i}$	$\boldsymbol{MD_i}$	$\boldsymbol{MD_i^r}$	$\boldsymbol{ECB_i}$	$\boldsymbol{PCB_i}$	$\boldsymbol{UCB_i}$
lcdnum	984	1440	192	20	20	20
bsort100	710289	89893	88907	20	20	18
ludcmp	27036	8607	3545	98	98	98
fdct	6550	6017	819	106	22	58
nsichneu	22009	147200	147200	256	0	256
statemate	10586	18257	3891	256	36	256

deadlines were set such that $T_i = D_i = (PD_i + MD_i)/U_i$. Task priorities were assigned according to deadline monotonic.

We randomly generated task sets and determined their schedulability using Eq. (19) for FP, RR, and TDMA buses under different settings, i.e., by varying core utilizations, number of cores, memory reload time d_{mem}, cache size and RR/TDMA slot size s that has a default value of 2.

1. Core Utilizations: In this experiment, we varied the per core utilization between 0.05 to 1 in steps of 0.05. For every value of core utilization, 1000 task sets were generated. Fig. 2 shows the number of task sets that were deemed schedulable by FP, RR and TDMA bus arbitration policies with and without considering cache persistence. Fig. 2 also shows a line marked as "perfect bus" which assumes that there is no interference on the memory bus when the bus utilization ≤ 1. That line provides an upper bound on the actual number of schedulable task sets at a particular core utilization. We can see in Fig. 2 that bus arbitration policies that account for cache persistence dominate their counterparts that do not consider cache persistence. This improved performance mainly results from a tighter bound on the number of bus request generated by tasks executing on different cores. We can see in Fig. 2a that for a FP bus, up to 70% more task sets were schedulable when considering cache persistence. Similarly, we can also see huge improvements for both RR (up to 65% more schedulable task sets) and TDMA (up to 50% more schedulable task sets). Note that the FP bus outperforms the RR and TDMA buses since it provides a tightly bounded bus latency for single accesses which is not the case with RR and TDMA.

2. Number of Cores: In this experiment, we varied the number of cores between 2 and 10 in steps of 2 with all other parameters set to the default values. We have used the weighted schedulability measure defined in [12] to plot the results in Fig. 3a. The weighted schedulability measure reduces what would otherwise be a 3-dimensional plot to 2-dimensions. We can see that by increasing the number of cores the total number of schedulable task sets decreases. This is mainly because by increasing the number of cores the number of tasks also increases. This leads to an increases in the interference on the memory bus. However, we can see that analyses that account for cache persistence always dominate their counterparts that do not account for cache persistence.

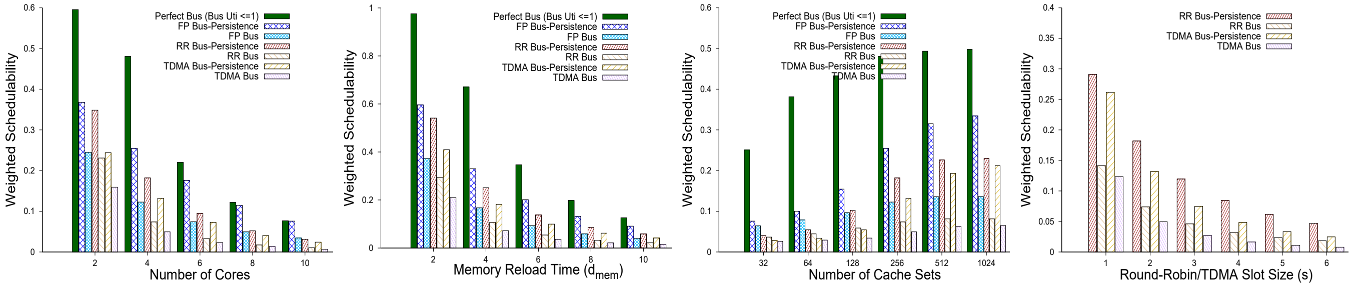
3. Memory Reload Time d_{mem}: For this experiment, we varied the value of memory reload time d_{mem} from $2\mu s$ to $10\mu s$ in step of $2\mu s$. The results are presented in Fig. 3b. We can see that for lower values of d_{mem} the difference between the weighted schedulability of cache persistence aware analyses and their respective counterparts is higher. However, for higher value of d_{mem} the time spent by tasks in performing memory operations increases and hence the schedulability of all approaches decreases.

4. Cache Size: To evaluate the impact of cache size on the performance of the analyses, we varied the size of the L1 cache of each core from 32 to 1024 cache sets, keeping default values for all other parameters. The results are shown in Fig. 3c. We can see that by increasing the cache size, the number of schedulable task sets under bus arbitration policies that account for cache persistence also increases. This is mainly because by increasing the cache size the number of PCBs of each task also increases, which results in gradually improving the performance of the analyses that account for cache persistence. Note that the increase in cache size also reduce CRPD and thus increase the task set schedulability for analyses that do not account for cache persistence, but at a slower rate than for persistence aware analyses.

5. RR/TDMA slot size s: For RR/TDMA buses the number of memory access slots per core, i.e., s, can greatly affect the memory bus contention suffered by tasks. To evaluate this, we varied the value of s between 1 to 6 and plotted the results in Fig. 3d. The results show that for smaller values of s, the difference between the weighted schedulability of cache persistence aware analyses and their respective counterparts is much higher. However, by increasing the value of s the task set schedulability of all approaches decrease, which is intuitive, considering the formulation of Eq. (8) and (9).

VI. Related Work

Schliecker et al. [13] used event arrival curves to bound the memory bus contention suffered by tasks. However, [13] only supports an unspecified work-conserving bus arbitration policy. Kelter et al. [14] and Chattopadhyay et al. [15] also proposed WCRT analysis techniques to bound memory bus contention in multicore systems considering a TDMA bus and a L1 instruction cache. However, their methods have limited



(a) Varying Number of Cores

(b) Varying d_{mem}

(c) Varying Cache Size

(d) Varying slot size (s)

Fig. 3: Weighted schedulability of different bus arbitration policies by varying different parameters

applicability since they assume non-preemptive scheduling. Dasari et al. [16] presented an analysis to bound the maximum number of bus requests that can be made by a task in a given time interval using performance counters. Consequently, the bus contention suffered by the tasks is modeled as an additional term in the WCRT analysis. This work was later extended in Dasari et al. [17]. Although the analysis in [16], [17] may be more accurate when it estimates the memory access demand of tasks but, it uses non-preemptive scheduling and assumes partitioned caches and therefore does not take cache related effects into account. This makes the analysis in [16], [17] less general than the one presented in this work. Huang et al. [18] presented a WCRT analysis that applies to multicore systems with one shared resource (e.g., memory bus) using a fixed-priority arbitration. The WCRT analysis in [18] has a speedup factor of 7 when used with a simple task-to-core allocation algorithm. However, their model does not consider the impact of caches on the shared resource access demand of tasks which may potentially lead to optimistic results. Davis et al. [2] explicitly modeled interference on cores, caches, memory bus and the main memory in a multicore system. The analysis in [2] accounts for CRPDs when bounding memory bus delay suffered by the tasks under different bus arbitration policies. However, since the bus contention analysis in [2] does not account for cache persistence it may overestimate the memory bus delay suffered by the tasks.

VII. Conclusion

In this work, we extend the notion of cache persistence from single core to multicore systems. We built on the idea that due to re-use of cache content between multiple executions of the same task, the number of bus accesses generated by the task may not always be equal to its worst-case memory access demand. We show how the memory bus contention suffered by tasks executing on a multicore platform can be upper bounded in the presence of cache persistence. We then evaluated the performance of cache persistence aware bus arbitration policies against their respective counterparts that do not account for cache persistence. Experimental results show that cache persistence aware bus contention analyses was able to deem up to 70% more task sets schedulable.

In future work, we plan to extend the proposed analysis to multilevel shared caches.

Acknowledgments. This work was partially supported 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(s) POCI-01-0145-FEDER-029119 (PREFECT); also by FCT and the ESF (European Social Fund) through POPH (Portuguese Human Potential Operational Program), under PhD grant SFRH/BD/119150/2016.

References

[1] J. Nowotsch, M. Paulitsch, D. Bühler, H. Theiling, S. Wegener, and M. Schmidt, "Multi-core interference-sensitive wcet analysis leveraging runtime resource capacity enforcement," in *ECRTS*. IEEE, 2014, pp. 109–118.

[2] R. I. Davis, S. Altmeyer, L. S. Indrusiak, C. Maiza, V. Nelis, and J. Reineke, "An extensible framework for multicore response time analysis," *Real-Time Systems*, vol. 54, no. 3, pp. 607–661, 2018.

[3] 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 *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] S. Hahn, J. Reineke, and R. Wilhelm, "Towards compositionality in execution time analysis–definition and challenges," in *CRTS*, 2013.

[6] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *JACM*, vol. 20, no. 1, pp. 46–61, 1973.

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

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

[9] J. Gustafsson, A. Betts, A. Ermedahl, and B. Lisper, "The Mälardalen WCET benchmarks – past, present and future," B. Lisper, Ed. Brussels, Belgium: OCG, Jul. 2010, pp. 137–147.

[10] S. A. Rashid, G. Nelissen, and E. Tovar, "Trading between intra-and inter-task cache interference to improve schedulability." in *RTNS*, 2018, pp. 125–136.

[11] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems*, vol. 30, no. 1-2, pp. 129–154, 2005.

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

[13] S. Schliecker and R. Ernst, "Real-time performance analysis of multiprocessor systems with shared memory," *TECS*, vol. 10, no. 2, p. 22, 2010.

[14] T. Kelter, H. Falk, P. Marwedel, S. Chattopadhyay, and A. Roychoudhury, "Static analysis of multi-core tdma resource arbitration delays," *Real-Time Systems*, vol. 50, no. 2, pp. 185–229, 2014.

[15] S. Chattopadhyay, A. Roychoudhury, and T. Mitra, "Modeling shared cache and bus in multi-cores for timing analysis," in *SCOPES*. ACM, 2010, p. 6.

[16] D. Dasari, B. Andersson, V. Nelis, S. M. Petters, A. Easwaran, and J. Lee, "Response time analysis of cots-based multicores considering the contention on the shared memory bus," in *TrustCom*. IEEE, 2011, pp. 1068–1075.

[17] D. Dasari, V. Nelis, and B. Akesson, "A framework for memory contention analysis in multi-core platforms," *Real-Time Systems*, vol. 52, no. 3, pp. 272–322, 2016.

[18] W.-H. Huang, J.-J. Chen, and J. Reineke, "Mirror: symmetric timing analysis for real-time tasks on multicore platforms with shared resources," in *DAC*. ACM, 2016, p. 158.