Toward memory-centric scheduling for PREM task on multicore platforms, when processor assignments are specified

Ikram Senoussaoui, Mohammed Kamel Benhaoua, Houssam-Eddine Zahaf, Giuseppe Lipari

Toward memory-centric scheduling for PREM task on multicore platforms, when processor assignments are specified

1st Ikram Senoussaoui
Univ-Lille, CRIStAL
Univ-Oran1, LAPECI
Lille, France

2nd Mohammed Kamel Benhaoua
Univ-Mascara
Mascara, Algeria

3rd Houssam-Eddine Zahaf
Univ-Nantes, LS2N
Nantes, France

4th Giuseppe Lipari
Univ-Lille, CRIStAL
Lille, France

***Abstract*—Real-time embedded systems are increasingly being built using commercial-off-the-shelf (COTS) components. Although these components generally offer high performance, they can occasionally incur significant timing delays. Computing precise bounds on timing delays due to contention is difficult without a proper support from the hardware.**

Rather than estimating contention safe delays, this work aims to avoid it. We consider hardware architectures where each core has a scratchpad memory and the task execution is divided into a memory phase and a computation phase (Predictable Execution Model – PREM). Tasks are allocated to cores by a partitioned scheduling scheme. Then we schedule memory phases using a non-preemptive scheduling approach, while computation phases are scheduled using preemptive single core schedulers.

This paper presents a new *artificial deadline* based approach to avoid contention in memory phases, where tasks memory phases are assigned appropriate deadlines and scheduled by a non-preemptive scheduler (EDF). The effectiveness of the proposed method is evaluated using a set of synthetic experiments in terms of schedulability and analysis time.

***Index Terms*—Real-time, Scheduling, PREM, multicore**

I. Introduction

Contention on memory resources was the subject of many research works [1], [14], [21], [27], [24]. Estimating precise bounds on timing delays due to contention is difficult. In general, it is hard to define what are the tasks that might interfere with each other, which likely leads to include scenario's that might never occur, therefore over-estimating the worst-case interference. A second approach tends to prevent interference by enforcing space and time isolation (e.g. time partitioning schemes).

The Acquisition-Execution-Restitution model (AER) [6], and the PRedictable Execution Model (PREM) [17] represent a good alternative. In the PREM model, the task is modeled in two phases: (i) a memory phase where all data are transferred from main memory to the local memory and vice-versa, (ii) and a pure computation phase where the loaded data are processed. Memory-transactions are scheduled so that the memory bus is accessed without interference (time isolation), while pure computations are executed on the platform cores. In this paper, we focus on the PREM model.

Several difficult challenges are encountered when scheduling real-time application modeled by the PREM model on multicore architectures. Finding a proper way to schedule memory phases is the most important difficulty since the bus controllers available on commercial platforms are not dedicated to real-time scheduling, they support very simple policies for example: First-In-First-Out (FIFO) scheduling policy. A second issue is the complexity of the scheduling problem: we need to consider at the same time scheduling of bus memory access and scheduling on the different cores.

In this paper, we propose novel method for assigning artificial deadlines to memory phases of a set of periodic PREM tasks, so to avoid memory contention, in particular, contention between the communication bus and the main memory. We do not address task to core assignment in this paper, therefore we use classical bin-packing heuristics to compute the task to core allocation. Finally, we provide a large set of synthetic experiments to demonstrate the efficiency of the proposed heuristic in terms of schedulability, as well as the required analysis time.

This paper is organized as follows: in the following section we review the state of the art. We present the hardware and task models in Section III. Further, we detail in Section IV how the artificial deadlines are assigned to memory phases. Within Section V, we give a quick overview how tasks can be allocated to cores. Results and simulations are described in Section VI. We draw conclusions in Section VII.

II. Related work

Modern COTS-based embedded systems include multiple active components (CPU cores and I/O peripherals) that can independently initiate access to shared resources (like main memory and inter-connection buses) which cause contention leading to timing degradation [16]. These architectures feature generally, a single port main memory shared among all CPU cores and peripherals, therefore, it is hard to guarantee the absence of memory conflicts during execution. In [19], authors have shown that the worst case execution time (WCET) of a task can increase linearly with the number of suffered cache misses, due to contention for access to main memory.

979-8-3503-9754-3/22/$31.002022 IEEE

To solve this problem, new execution models making use of pre-fetching techniques have been proposed in the literature. These techniques have been investigated first by Rosen et al. [21] in the context of worst-case execution time computation and bus access optimization. The authors show in [8] that pre-fetching techniques improve the cache/scratchpad locality and reduce average execution times.

The PRedictable Execution Model (PREM) from Pellizzoni et al [17] was first proposed to co-schedule both memory requests from CPU and I/O with computations on uniprocessor platforms with multi-level caches. It splits tasks in memory/computation phases. This execution model drastically reduces the variability of memory-contention latency by explicitly controlling memory accesses during memory phases. *PREM* has been extended later in [27] to partitioned multicore/processor platforms where the core isolation is provided through a coarse-grained TDM memory schedule.

Different memory-aware scheduling policies for PREM tasks are evaluated in [3] by simulating synthetic task systems on a multicore platform. The authors of this work used different combinations of TDMA slots (different slot sizes) and different priority-based schedulers for the memory and computation phases on a partitioned multicore platform.

The Acquisition-Execution-Restitution model (*AER*) [6] is a generalization of the *PREM* model where each task is divided into three distinct parts: two memory phases (reading and writing) and a computation phase.

We believe that the memory hierarchies of modern multicores can further boost the effectiveness of the PREM model, and, in general, improve WCET predictability. The reason is that, platform designers replace traditional data caches with explicitly managed memories such as scratchpads (SPMs). The key point is that the behavior of explicitly-managed scratchpads is also much more predictable than that of caches, because access latency is independent of the access pattern [13], [20], [25].

The schedulability problem for globally scheduled PREM tasks using the memory phase prioritizing concept is addressed in [26]. The work in [2] proposes a global fixed-priority scheduling algorithm for a set of sporadic *PREM* tasks, it considers co-scheduling a separate DMA component to perform transfers from main memory to scratchpad and vice-versa. Melani et al. [15] proposed exact response time analysis for fixed-priority scheduling on a single core with a fully preemptive DMA engine.

Much works have been done on the communication and task scheduling problem on single/multi-processor architectures. A good overview of the common real-time scheduling methods is given in [12]. The relevant scheduling approaches can be classified on: event-based scheduling, time-triggered scheduling and hybrid approaches.

One of the most effective techniques to schedule dependent tasks on multicore platforms is to assign artificial deadlines and offsets in order to enforce precedence constraints [9]. The most popular heuristic algorithms are *fair* distribution and *proportional* distribution. We propose in this paper a technique for assigning artificial delays to memory phases to avoid memory contention.

III. System model

A. Architecture model

We consider in this work multicore architectures composed of m cores. Each core has access to a private scratchpad memory. Cores are connected through a single shared bus to a global shared memory. Before starting the computation, the different tasks trigger memory copies between main and scratchpad memories, explicitly by the system designer.

We assume that all memory is directly accessible to all cores via different address spaces. An example of such architecture is the Infineon Aurix TC397 [10].

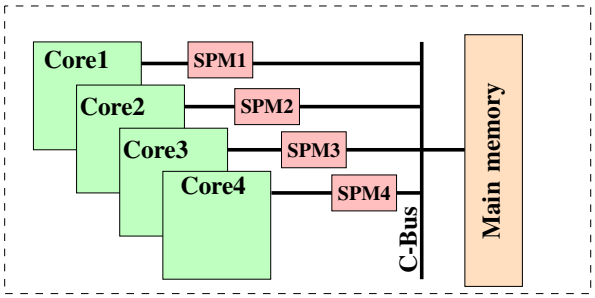


Fig. 1: Multicore target platform.

In Figure 1, we report an architecture compound of 4 cores. Each core is connected to its own scratchpad memory. Different scratchpads are connected using a single bus (C-bus) to the main memory.

B. Task model

We consider a set $\mathcal{T}$ of n independent periodic tasks, i.e., $\mathcal{T} = \{\tau_1, \tau_2, \cdots, \tau_n\}$. According to the semantics defined in *PREM*, each task τ_i is divided in two phases namely: *memory phase* and *computation phase*. The later, can not start before the completion of the *memory phase*. A task τ_i is characterized by tuple $\tau_i = (M_i, C_i, \mathsf{D}_i, \mathsf{T}_i)$: M_i is the task worst case data-prefetch time. It represents the required time that the task τ_i needs to load data from memory and/or I/O devices; C_i is the task worst case computation time. This phase can be preempted contrary to the memory phase; D_i is the task's relative deadline. Each instance of task τ_i must finish its execution no later than D_i time units after its activation; T_i is the task period, it represents the exact time interval between two consecutive activations of τ_i.

The computation phase of tasks does not access the shared bus, thus contentions when accessing the shared bus do not exist between computation phases and memory phases.

We denote by $\mathcal{H}$ the taskset hyperperiod, i.e. the system's period. It is defined as the least common multiple between all periods of tasks $\mathcal{H} = LCM(T_1, T_2, \cdots, T_n)$.

The task utilization is given as:

$$U_i = u_i^m + u_i^c$$

where, $u_i^m = \frac{M_i}{\mathsf{T}_i}$ (resp. $u_i^c = \frac{C_i}{\mathsf{T}_i}$) denotes the memory phase (resp. computation phase) utilization.

The total utilization of taskset $\mathcal{T}$ is computed as follow:

$$U_{\mathcal{T}} = \sum_{i}^{n} U_i$$

IV. Deadline-based memory-processor co-scheduling

We address in this paper the bus contention problem by avoiding memory requests overlap entirely. In this section, we will show how to determine artificial deadlines to memory phases so that they do not overlap during runtime. We consider all the memory requests as non-preemptive tasks having a period and an artificial deadline δ_i and scheduled on the communication bus using Earliest Deadline First (EDF) scheduler. The schedulablity on the different cores will be tested using EDF algorithm. The analysis is based on the processor demand function [4], [18], and considers only the computation phases. The computed artificial deadlines will be used as offsets of the computation phases.

A. Artificial deadline assignment heuristic (ADA)

One strategy to solve the artificial deadline assignment problem could be to search exhaustively among all possible artificial deadline combinations in a Pareto-Front fashion. In a such strategy, the objective would be both to ensure a non-preemptive scheduling of memory phases to avoid the overlap on the communication bus and to minimize their artificial deadlines in order to improve the schedulability on cores. Although the Pareto-Front based approaches are exact and have the advantage to find all the realizable solutions for the considered problem, they suffer from large computational complexity, as the search space may potentially be very large. In fact, any realizable solution can be accepted from real-time perspective. We present in this section a weight-based artificial deadline assignment heuristic for computing deadlines of memory phases. The search is done for each task τ_i in the interval $I_i = \mathsf{D}_i - C_i - M_i$ as shown in figure 2.

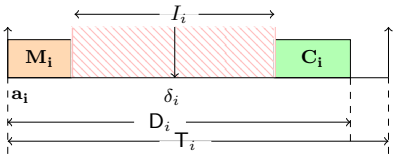


Fig. 2: The task artificial deadline search interval.

We will assume that each task τ_i is characterized by a weight w_i such that $w_i < 1$ (the weight is the same for all the tasks). The idea is to search for the solution by deceasing the memory phase artificial deadline for each task using its weight and to stop the search after the first realizable solution.

Algorithm 1 implements our method for a set of PREM tasks. Lower bound lb_i and upper bounds ub_i are computed for every task. $\mathcal{S}$ is the current solution, it contains the task set with assigned artificial deadlines, it is initialized to empty before the algorithm starts, and $\mathcal{S}_{remained}$ is its remainder, it contains the taskset before the assignment of the artificial deadlines and it is full at the beginning of the algorithm.

Algorithm 1 artificial_deadlines_assignment($\mathcal{T}, \mathcal{S}$,stop)

```
1:  Require: S_realizable
2:  if S_remained ≡ ∅ then
3:     if dbf_analysis_np(S) then
4:        S_realizable ← S
5:        stop ← true
6:     end if
7:     return true;
8:  end if
9:  curr_d ← (d_head_S = 0)?(ub_head_S) : d_head_S
10: repeat
11:    Sol.copy(S)
12:    d_head_Sol ← curr_d
13:    remove(d_head_Sol, Sol_remained)
14:    artificial_deadlines_assignment(T, Sol,stop)
15:    curr_d = curr_d * w_head_Sol
16: until (curr_d < lb_head_S ∧ stop = true )
```

The algorithm computes the artificial deadlines for all the tasks in $\mathcal{S}_{remained}$. The artificial deadline of every task τ_i is equal to ub_i in the beginning. At each iteration, the algorithm selects the task in the head and moves it to $\mathcal{S}$ (line 11-13). Once, $\mathcal{S}_{remained}$ for the current solution is empty, the algorithm performs the non-preemptive EDF schedulability test of [11] to assess the feasibility on the bus. If the solution is feasible on the bus, then the algorithm saves the solution into $\mathcal{S}_{realizable}$, sets the booleen variable *stop* to **true** and exit successfully (line 2-8). The schedulability is tested, further, on the different cores using EDF algorithm (IV-B).

If the schedulablility test fails on the bus for the current solution $\mathcal{S}$, the algorithm ignores the solution, and continues to search in the solution space by decreasing the artificial deadline for the task in the head τ_i using its weight while fixing the artifitial deadline of the other tasks, until reaching the lower bound lb_i.

The main difference between algorithm 1 and a pure Pareto-Front based algorithm when solving our problem is at lines (5 and 15): (i) algorithm 1 uses the task weight w_i, to decrements the task artificial deadline rather than reduce it by 1, and (ii) if a realizable solution is found, then, algorithm 1 stops the research immediately and returns the solution rather than compare it with the Pareto-set using the Pareto dominance concept and adding it to the Pareto-set if not dominated.

In our prototype, we chose np-EDF for scheduling the memory phases on the communication bus and EDF for computation phases on the cores. However, our strategy is general enough that other schedulers can be easily plugged in our framework. As long as offsets are assigned to act as precedence constraints between memory phases and computation phases, the choice of the scheduler is arbitrary.

B. Schedulability analysis on cores

We use in this paper, the Earliest Deadline First algorithm (EDF) to schedule computation phases on cores. Our analysis is based on the processor demand criterion [18] and considers only computation phases with offsets. The processor demand function for computation phases with offsets is defined as follows:

$$df(t_1, t_2) = \sum_{i=1}^{n} \Delta_i(t_1, t_2) \cdot C_i \quad (1)$$

Where:

$$\Delta_i(t_1, t_2) = \left\lfloor \frac{t_2 - \phi(C_i) - \mathsf{D}_i}{\mathsf{T}_i} \right\rfloor - \left\lceil \frac{t_1 - \phi(C_i)}{\mathsf{T}_i} \right\rceil + 1 \quad (2)$$

It is the amount of time demanded by the tasks (computation phases) in interval $[t_1, t_2)$ that the core must execute to ensure that no task misses its deadline. Considering a fully preemptive single core scheduler, a necessary and sufficient condition for a set of tasks (computation phases) to be schedulable by EDF consists in checking that the demand never exceeds the length of the interval.

lemma 1: (Baruah et al. [5]). The taskset $\mathcal{T}$ is feasible on a single core ($U_{\mathcal{T}} \leq 1$) if and only if:

$$\forall 0 \leq t_1 < t_2 \leq \mathcal{H}, \quad df(t_1, t_2) \leq t_2 - t_1 \quad (3)$$

V. Allocation heuristics

The tasks are assumed to be already assigned to the different cores of the platform. In this paper, we use the bin-packing heuristics: Best-Fit (BF) and Worst-Fit (WF) to perform task-core allocation (Algorithm 2) which is likely similar to the one used in [22]. BF and WF sort the cores by capacity. For BF the cores are sorted in a decreasing order of their utilizations, whereas in the case of WF, they are sorted in increasing order of utilization.

First, the algorithm sorts PREM tasks according to their deadlines or their periods, then, for each task, it sorts cores according to parameter $alloc$ (Best-Fit or Worst-Fit order) (line 4). Then, it tests the possibility of allocating the selected task on each core in turn. If the allocation is successful, the next PREM task is tested, otherwise the algorithm tries the next core (line 5-11). If the studied PREM task cannot be allocated on any core, then, the algorithm fails (line 12-14). If all PREM tasks have been allocated, the algorithm returns the tasks allocation.

VI. Results and discussions

In this section, we present the performances of the proposed approach. We conducted experiments with randomly generated workloads to evaluate the proposed approach using different task partitioning heuristic. Consistently with previous results in the literature, WF outperforms BF in all simulated scenario, therefore, we only report the results for WF for clarity of presentation. We study the impact of the workload size on schedulability and the require time to complete the analysis.

Algorithm 2 Heuristics($\mathcal{T}, \mathcal{P}, alloc[BF, WF], order$)

```
1: sort_tasks(T, order)
2: for (∀τ_i ∈ T) do
3:    allocated← false
4:    sort_cores(P, alloc)
5:    for (∀p ∈ P) do
6:       if (schedulable(τ_i, p)) then
7:          allocate task into core p
8:          allocated ← true
9:          break;
10:      end if
11:   end for
12:   if (allocated= false) then
13:      No task allocation is found
14:   end if
15: end for
16: return tasks allocation
```

A. Task set generation

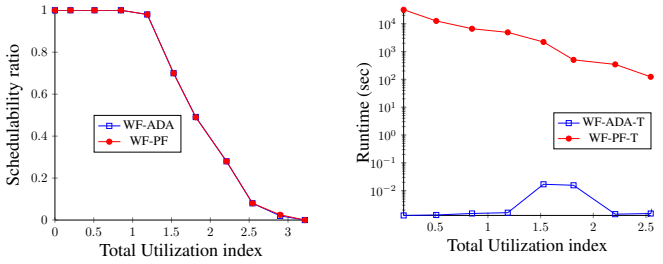
The task set generation process takes as input n the number of tasks and the target total utilization $U_{\mathcal{T}}$. It starts by generating the utilizations of the n tasks by using the UUniFast-Discard [7] algorithm. We varied the baseline utilization from 0.2 to P (number of available cores) with a step of 0.4. For every utilization u_i, the algorithm generates the memory phase utilization u_i^m using a random value stall $= (\frac{M_i}{C_i+M_i})$ that represents the utilization of the memory phase: for example, a task with a memory stall of 0.1 spends 10% of its WCET in the memory phase. The range of random stall in our experience is: $lev = [0.10, 0.20]$. We fixed the task weight to 0.5. We highlight that the total workload comprises also the memory phases, hence, the computation workload on the cores is smaller than the total utilization $U_{\mathcal{T}}$.

For each utilization, we use 50 tasksets per utilization. We generate 10 tasks per taskset. The task period is selected randomly from a predefined list of periods: {80,100,150,200,240,300, 400,500, 600,800}, so to establish an upper bound to the hyperperiod. The task deadline is set to 70% of the task's period.

B. Results of synthetic task set experiments

In this section, we evaluate the performance of our proposed approach against a Pareto-Front based approach which uses the task weight to search **all** the realizable solutions. Tasks are allocated on 4 cores by Worst-fit (WF) heuristic.

Figure 3 shows the schedulability rate and the required analysis time of schedulable tasksets as a function of the utilization. We simulated the memory stall $[0.1 - 0.2)$ since it is the closest to reality. The same performance behavior is noticed for our proposed approach (WF-ADA) and Pareto-Front based approach (WF-PF) since they search for solutions exactly the same way. WF-ADA as well as WF-PF (in this experience) are not exact because they check only a subset of the design space, however, their performances results in term



(a) Schedulability rates

(b) Required analysis time

Fig. 3: Assignment artificial deadlines algorithms performances

of schedulablity are acceptable compared to an exact approach (example: a Pure Pareto-Front based approach or an ILP-based approach [23]).

As expected, WF-PF based approach require more time than our proposed approach WF-ADA, even for small tasksets of 10 tasks and using a task weight equal to 0.5, Unfortunately, it increases exponentially with the number of tasks, and this is why we limited our analysis to 10 tasks per taskset. It is, therefore, unfeasible to run WF-PF (and also the exact approaches) on large tasksets during design time when the parameters of the tasksets may change frequently. The runtime of WF-ADA is still acceptable (a few seconds in average).

VII. Conclusion and Future Work

Multicore systems using ScratchPad Memories (SPMs) are attractive architectures for executing time-critical embedded applications, because they provide both predictability and performance. However, they are likely subject to contention and require a particular attention when supporting hard real-time constraints. In this paper, we proposed technique based on artificial deadline assignment for contention avoidance. Our experiments show a significant improvement in the system performances compared to state-of-the-art (example: FIFO based buses). As future work, we plan to take into account allocation strategies in the analysis phase for AER task model.

References

[1] Ahmed Alhammad and Rodolfo Pellizzoni. Time-predictable execution of multithreaded applications on multicore systems. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6. IEEE, 2014.

[2] Ahmed Alhammad, Saud Wasly, and Rodolfo Pellizzoni. Memory efficient global scheduling of real-time tasks. In *21st IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 285–296. IEEE, 2015.

[3] Stanley Bak, Gang Yao, Rodolfo Pellizzoni, and Marco Caccamo. Memory-aware scheduling of multicore task sets for real-time systems. In *2012 IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 300–309. IEEE, 2012.

[4] Sanjoy K Baruah, Louis E Rosier, and Rodney R Howell. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Systems*, 2:63–119, 1990.

[5] Sanjoy K Baruah, Louis E Rosier, and Rodney R Howell. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-time systems*, 2(4):301–324, 1990.

[6] Guy Durrieu, Madeleine Faugère, et al. Predictable flight management system implementation on a multicore processor. In *Embedded Real Time Software (ERTS'14)*, 2014.

[7] Paul Emberson, Roger Stafford, and Robert I Davis. Techniques for the synthesis of multiprocessor tasksets. In *WATERS*, 2010.

[8] Johannes Adzer Hoogeveen, Jan Karel Lenstra, and Bart Veltman. Preemptive scheduling in a two-stage multiprocessor flow shop is np-hard. *European Journal of Operational Research*, 89(1):172–175, 1996.

[9] Zahaf Houssam-Eddine, Nicola Capodieci, et al. The hpc-dag task model for heterogeneous real-time systems. *IEEE Transactions on Computers*, 70(10):1747–1761, 2021.

[10] AG IT. Aurix 32-bit microcontrollers for automotive and industrial applications. *Infineon Technologies AG*, 1.

[11] Kevin Jeffay, Donald F Stanat, et al. On non-preemptive scheduling of periodic and sporadic tasks. In *IEEE real-time systems symposium*, pages 129–139. US: IEEE, 1991.

[12] Jan Korst, Emile Aarts, and Jan Karel Lenstra. Scheduling periodic tasks. *INFORMS journal on Computing*, 8(4):428–435, 1996.

[13] Ben Lickly, Isaac Liu, et al. Predictable programming on a precision timed architecture. In *Proceedings of the 2008 international conference on Compilers, architectures and synthesis for embedded systems*, pages 137–146, 2008.

[14] Cláudio Maia, Geoffrey Nelissen, Luis Nogueira, Luis Miguel Pinho, and Daniel Gracia Pérez. Schedulability analysis for global fixed-priority scheduling of the 3-phase task model. In *2017 IEEE 23rd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 1–10. IEEE, 2017.

[15] Alessandra Melani, Marko Bertogna, Robert I Davis, Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, and Giorgio Buttazzo. Exact response time analysis for fixed priority memory-processor co-scheduling. *IEEE Transactions on Computers*, 66(4):631–646, 2016.

[16] Rodolfo Pellizzoni, Emiliano Betti, Stanley Bak, Gang Yao, John Criswell, and Marco Caccamo. Predictable execution model: concept and implementation. Technical report, 2010.

[17] Rodolfo Pellizzoni, Emiliano Betti, et al. A predictable execution model for cots-based embedded systems. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 269–279, 2011.

[18] Rodolfo Pellizzoni and Giuseppe Lipari. Feasibility analysis of real-time periodic tasks with offsets. *Real-Time Systems*, 30(1-2):105–128, 2005.

[19] Rodolfo Pellizzoni, Andreas Schranzhofer, Jian-Jia Chen, Marco Caccamo, and Lothar Thiele. Worst case delay analysis for memory interference in multicore systems. In *Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*, pages 741–746, 2010.

[20] Isabelle Puaut and Christophe Pais. Scratchpad memories vs locked caches in hard real-time systems: a quantitative comparison. In *2007 Design, Automation & Test in Europe Conference & Exhibition*, pages 1–6. IEEE, 2007.

[21] Jakob Rosen, Alexandru Andrei, et al. Bus access optimization for predictable implementation of real-time applications on multiprocessor systems-on-chip. In *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*, pages 49–60. IEEE, 2007.

[22] Ikram Senoussaoui, Houssam-Eddine Zahaf, Mohammed Kamel Benhaoua, Giuseppe Lipari, and Richard Olejnik. Allocation of real-time tasks onto identical core platforms under deferred fixed preemption-point model. In *Proceedings of the 28th International Conference on Real-Time Networks and Systems*, pages 34–43, 2020.

[23] Ikram Senoussaoui, Houssam-Eddine Zahaf, Giuseppe Lipari, and Mohammed Kamel Benhaoua. Contention-free scheduling of prem tasks on partitioned multicore platforms. In *Proceedings of the 27th IEEE International Conference on Emerging Technologies and Factory Automation (to appear)*.

[24] Rohan Tabish, Renato Mancuso, et al. A real-time scratchpad-centric os with predictable inter/intra-core communication for multi-core embedded systems. *Real-Time Systems*, 55(4):850–888, 2019.

[25] Saud Wasly and Rodolfo Pellizzoni. A dynamic scratchpad memory unit for predictable real-time embedded systems. In *2013 25th Euromicro Conference on Real-Time Systems*, pages 183–192. IEEE, 2013.

[26] Gang Yao, Rodolfo Pellizzoni, Stanley Bak, Heechul Yun, and Marco Caccamo. Global real-time memory-centric scheduling for multicore systems. *IEEE Transactions on Computers*, 65(9):2739–2751, 2015.

[27] Gang Yao, Rodolfo Pellizzoni, et al. Memory-centric scheduling for multicore hard real-time systems. *Real-Time Systems*, 48(6):681–715, 2012.