IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received February 11, 2022, accepted February 12, 2022, date of publication February 15, 2022, date of current version March 3, 2022.

*Digital Object Identifier 10.1109/ACCESS.2022.3151891*

# A Survey of Techniques for Reducing Interference in Real-Time Applications on Multicore Platforms

**TAMARA LUGO[1], (Member, IEEE), SANTIAGO LOZANO[2], JAVIER FERNÁNDEZ[1], (Member, IEEE), AND JESUS CARRETERO[1], (Senior Member, IEEE)**

[1]Computer Science and Engineering Department, University Carlos III of Madrid, 28911 Leganés, Spain
[2]SENER Aeroespacial, Tres Cantos, 28760 Madrid, Spain

Corresponding author: Jesus Carretero (jcarrete@inf.uc3m.es)

This work was supported in part by the Comunidad de Madrid Government "Nuevas Técnicas de Desarrollo de Software de Tiempo Real Embarcado Para Plataformas. MPSoC de Próxima Generación" under Grant IND2019/TIC-17261.

**ABSTRACT** This survey reviews the scientific literature on techniques for reducing interference in real-time multicore systems, focusing on the approaches proposed between 2015 and 2020. It also presents proposals that use interference reduction techniques without considering the predictability issue. The survey highlights interference sources and categorizes proposals from the perspective of the shared resource. It covers techniques for reducing contentions in main memory, cache memory, a memory bus, and the integration of interference effects into schedulability analysis. Every section contains an overview of each proposal and an assessment of its advantages and disadvantages.

**INDEX TERMS** Real-time systems, architecture, multicore, timing analysis, schedulability analysis, WCET, co-runner interference.

## I. INTRODUCTION

In a real-time system, the application's response time to external stimuli must be within a specified time interval. Real-time systems can be classified into hard real-time systems and soft real-time systems. Hard real-time systems have a strict response time, and failure to meet a single response time is enough for the system to fail. Examples of these types of systems are aeronautics, satellite control, automotive, etc. In soft real-time systems, the response time is flexible; the occasional loss of its deadlines can cause the system to have a degraded operation during an interval that must be limited and allow the system to recover regular operation quickly.

One of the main problems in developing a real-time system is ensuring that it always meets the required functionality within the specified time margins. Through schedulability analysis, the use restrictions of each of the available resources can determine the response time of a running task. This analysis requires obtaining the Worst-Case Execution Time (WCET) as accurately as possible [1], [2]. WCET is the maximum time it takes for a task to run on a specific hardware platform. The predictability of the response time determines whether the system will offer a correct response to the arrival of a particular stimulus in a limited time.

The associate editor coordinating the review of this manuscript and approving it for publication was Barbara Masini.

Traditionally, aerospace systems have used very conservative architectures to ensure correct execution, reliability, and predictability. Platforms traditionally used in space applications are embedded systems with a single-core CPU and several specialized devices. Moreover, most of them are FPGA-based, as many examples have demonstrated the increased performance and efficiency of reconfigurable systems over traditional programmable processor approaches [3]. A well-known example is LEON3, an FPGA-based architecture with a single-core RISC microprocessor initially designed by the European Space Agency [4].

In the last decade, the arrival of multicore processors has provided a solution in embedded systems to allow the execution of mixed-criticality application workloads comprised of various hard real-time (HRT) and non-HRT (NHRT) applications and to provide integrated architectures [5]. Figure 1 shows, for example, the multicore architecture designed for LEON4.

However, in multicore systems, the predictability of response time and WCET analysis is more complex because of the parallel execution of applications, which creates interferences among the tasks due to simultaneous accesses to shared resources and causes variations in the execution time of the tasks depending on the execution order [6].

Interference is a challenge in real-time systems running on multicore platforms due to the complexity of the systems

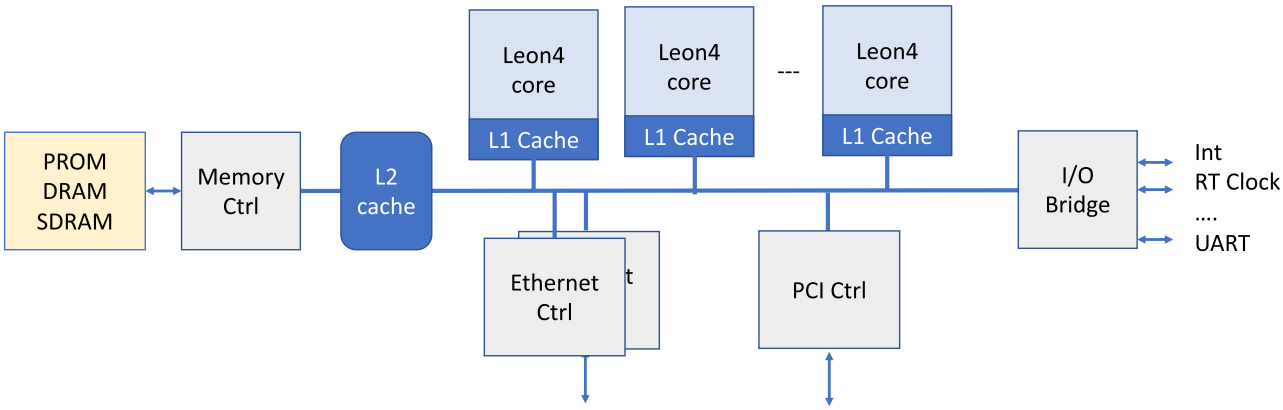


**FIGURE 1.** Example of a multiprocessor system on a chip (MPSoC) architecture.

and the impossibility of completely controlling all hardware resources. In recent years, a robust research effort has been made to avoid, or at least alleviate, the interferences so that the execution time could be more deterministic. Users who design and implement real-time systems on multicore platforms need to identify the main sources of interference and the techniques to obtain more deterministic runtime behavior.

This survey provides an overview of the scientific literature on reducing interference in real-time applications on multicore platforms. It also includes the proposals that use interference reduction techniques without considering the predictability issue. Although our survey focuses on the context of real-time applications, we consider that the works focused on achieving a better performance are also relevant for researchers and engineers dealing with interferences in real-time systems. The proposals are categorized from the point of view of the shared resource. The survey includes an independent section for the discussion and analysis of the proposals that integrate the effects of interference into schedulability, taking into account that of the total of publications reviewed in this period, they show the highest incidence. The proposals in each section are categorized and concluded with a discussion of the advantages and disadvantages.

### A. RESEARCH SCOPE AND PAPER ORGANIZATION

Following research carried out [7]–[9], there are two major topics in this aspect: main sources of interference in multicore systems and how to integrate interference effects into schedulability analysis. Thus, we made a bibliographic search on those topics. After a first refinement of the results found, three significant sources of interferences were identified: the cache, the memory bus, and the main memory.

We made a second more focused bibliographic search on those three categories plus schedulability. Figure 2 illustrates the behavior of the publications made in each of the four main categories, in 2-year periods, from 2015/16 to 2019/20. Reviewed papers are compiled from repositories such as

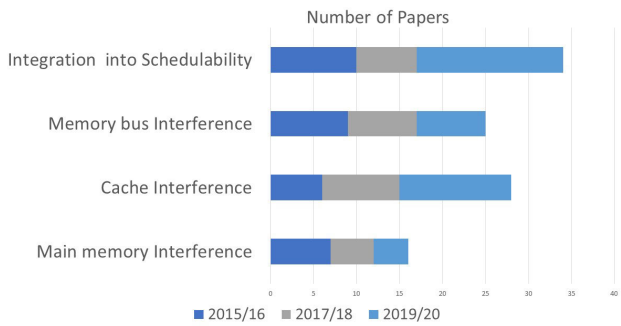


**FIGURE 2.** Distribution of classified papers over the past 5 years.

IEEE Xplore, ACM Digital Library, arXiv, and Google Scholar. The distribution of the publications reviewed in this period shows a greater incidence in the proposals that integrate the effects of interference into schedulability, representing 33 percent of the total publications reviewed in this period. It is followed in order of incidence by the proposals focused on reducing the effects of interference in the shared cache, which shows a growing trend and represents 27 percent of all publications. To a lesser extent, but equally significant, were the proposals for interference reduction techniques in the shared main memory and the shared memory bus, representing 16 percent and 24 percent, respectively.

Thus, in this survey, we have grouped the techniques for reducing interference in shared resources of multicore real-time systems using those three fundamental categories.

The structure of the survey is described below.

Section II presents information about the main *sources of interference* due to the simultaneous use of shared resources in multicore systems. The causes of interference from those shared resources and their effect on the execution time of a task are initially exposed.

Section III shows the proposed techniques for interference reduction. They are grouped according to the source of interference on which it is focused. *Techniques focused*

*on main memory*. Proposals in this category focus on interference reduction techniques in shared main memory divided into three fundamental categories: Spatial memory resource isolation, Temporal memory resource isolation, and Predictable DRAM controllers. *Techniques focused on cache memory*. This section presents the techniques for reducing interference in shared cache memory divided into four fundamental categories: cache-partitioning techniques, cache-locking techniques, designing predictable cache coherence protocols, and other approaches. *Techniques focused on memory bus*. In this category, the approaches are divided into four fundamental categories: Memory Bandwidth Regulator, Phased Execution Model, Offline Scheduling, and Hardware isolation. Each section begins with an overview of the proposals and ends with a discussion of the advantages and disadvantages of each approach. Each one also includes a discussion of interference reduction techniques without considering the issue of predictability.

Section IV presents *techniques to integrate the effects of interference into schedulability analysis*. Proposals in this category focus on integrating interference effects due to the use of shared hardware resources into schedulability analysis. The investigations carried out are grouped into different categories considering the shared resource they consider and the integration of the effect of multiple shared resources.

Finally, Section V shows major conclusions from this survey.

The overall organization of this article is shown in Figure 3.

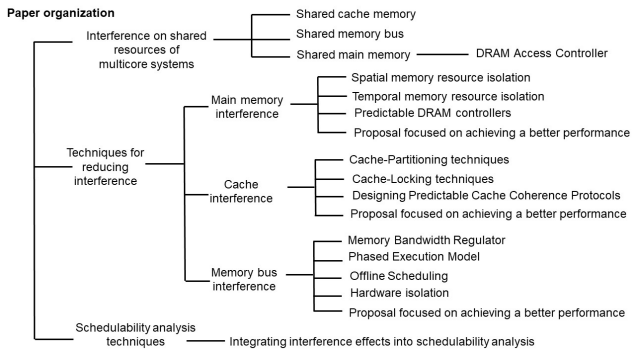


**FIGURE 3.** **Paper organization.**

## II. INTERFERENCE ON SHARED RESOURCES OF MULTICORE SYSTEMS

In general, the problem with the use of shared resources in multicore systems lies in the unpredictable delays in the execution time of the tasks. Two requests issued to a hardware resource that can only handle one request at the same time, for example, causes one of the requests to wait for the resource to become available. Arbitration mechanisms are used to assign shared resources to tasks. The delays caused by the implemented arbitration mechanism increase with the increase in the number of simultaneous requests [10].

Research carried out [7]–[9] indicates the main sources of interference in multicore systems. Dasari *et al.* [7] and Löfwenmark and Nadjm-Tehrani [8] identified the main sources of unpredictability in multicore systems based on shared resource interference. Shared caches, shared interconnect networks, and shared main memory are identified as the main sources of interference. Also, in 2016, Nagalakshmi and Gomathi [9] identified and evaluated the main sources of unpredictability in a critical security domain such as the aeronautical industry. The sources of interference identified are the main memory and shared memory bus, DRAM Access Controller, Cache Memories, Logical units pipelines, and addressable peripheral, considering the most significant interference in memories and shared bus.

This section considers shared cache memory, a memory bus, and main memory as significant sources of interference. The causes of interference from these shared resources in multicore systems and their effect on the execution time of a task are explained below.

### A. SHARED CACHE MEMORY

Cache memory is located between the CPU and main memory to allow the system to operate faster. For this reason, the areas of main memory most likely to be referenced are stored in the cache. A mechanism for updating the cache with the most likely data is implemented by dividing the main memory into blocks of several bytes and the cache into lines of equal size. A cache hit occurs if the content of the physical address is found in a block located on a cache line, and a cache miss occurs when the content of the physical address is not found in a block located on any line of cache. When a cache miss occurs, it is necessary to access the main memory to obtain the block containing the physical address, bring the content of the physical address to the CPU, and load the main memory block into the allocated cache frame.

Generally, the architecture of a multicore system has a shared last-level cache (LLC) and one or more levels of private caches. This architecture poses challenges when analyzing the effect on the execution time of the shared cache due to non-deterministic replacement algorithms and the use of preemptive scheduling, which is more severe in multicore systems.

The shared cache level between tasks that could preempt each other on the same core causes additional delays with complex computational analysis [7]. When a task starts executing by expelling another, the cache contents will be modified, loading the code and data of the other running task. When the task that has been expelled resumes its execution, it will find a different cache state than it had before its expulsion. This change in cache content will cause a cache miss burst until the cache reloads the instructions and data of the ejected task. As a result, the ejected task will increase its response time because it must wait for the execution time of the ejection task to finish. In addition, the ejected task must restore the cache contents it had before the ejection. The delay

caused by this effect is called the cache-related preemption delay [11].

In multicore systems, the co-execution of tasks in multiple cores leads to an increase in the probability of replacing cache lines in the LLC. This increase in the replacement of the contents of the cache lines shared between all cores causes an increase in cache misses, and consequently, an increase in requests to main memory [7].

Shared cache interference occurs when tasks on the same core replace their private cache lines (intratask interference) or replace the private cache lines of another task (intertask interference). Inter-core interference occurs when a task replaces the contents of a shared cache line that is used by a different task on another core [8].

Other necessary aspects to consider are write cache policy, cache-coherence issues, and the use of scratchpads as an alternative to caches.

There are two standard options for the write policy: write through and write back. The data is written directly to the main memory in a write-through cache. On the other hand, in a write-back, the data is stored in the cache line marked as dirty and is only written to the main memory when the dirty cache is evicted. The lack of WCET analysis support, because the write backs are decoupled in time from the corresponding stores in the program, prevents its use on hard real-time systems [12]. Benedicte *et al.* [13] presented an extensive analysis of write policies.

Cache coherence allows high programmability by moving data and keeping them coherent between all caches in the system. However, this generates power consumption and an increasing amount of coherence traffic. Scratchpad memories [14] are an alternative to cache hierarchies; they allow access as fast as a cache, without coherence traffic and more efficient energy consumption. Still, unlike shared coherent cache memory, they have poor programmability [15]. Also, scratchpad memories support is rare in commercially available platforms [16], [17]. Several proposed approaches focus on using scratchpads instead of caches [18], [19]. Gracioli *et al.* [17] presented a summary of work concerning scratchpads and compared them with the caches. Puaut and Pais [20] presented a quantitative comparison of scratchpad memories vs. locked caches in hard real-time systems.

### B. SHARED MEMORY BUS

In 2016, Nagalakshmi and Gomathi [9] described the system bus as an interconnection structure for data transfer, establishing that only one system component can access the bus at a time. The shared bus to access main memory also affects task execution times due to contentions [21]. When multiple components attempt to access a bus simultaneously, they cause conflicts. Arbitration mechanisms such as Round Robin or First Come First Served (FCFS) policies control access to the bus. Thus, the additional delay in execution time is a function of the time required for the bus to be available, the speed and data width of the bus, and the arbitration mechanism implemented.

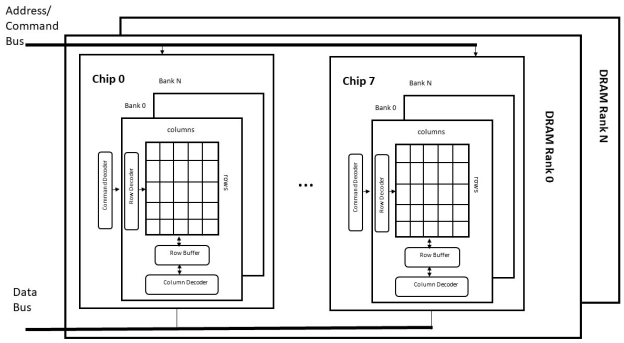


**FIGURE 4.** DRAM Device Organization.

### C. SHARED MAIN MEMORY

Each DRAM chip is made up of banks, and these are made up of a matrix of rows and columns of memory locations. Each DRAM bank has a row buffer that can store the content of one row [7]. Figure 4 shows the organization of the DRAM device. Access to the content of a memory location is done through the row buffer. A row stored in the row buffer by a previous activation command is considered open, and access to an open row is considered a row hit. If the desired row is not stored in the row buffer, it is considered closed (row miss). A row hit allows executing read/write commands to access the data; however, a row miss requires first executing a command to write the row to the array and executing the activation command to load the desired row into the row buffer.

DRAM access latency varies depending on the row stored in the requested bank's row buffer. The access latency is lower if the memory request corresponds to the row already in the row buffer. If the memory request does not correspond to the row that is in the row buffer, the access latency is higher since it is required to close the currently open row and load the requested row into the row buffer [22].

Interleaved accesses to main memory cause additional delays due to the operation of the cores in different memory pages that require the controller to continuously open and close new memory pages [23].

#### 1) DRAM ACCESS CONTROLLER

The memory controller schedules memory accesses based on system requirements. The memory controller consists of a request buffer that contains the status information for each memory request, read/write buffers that contain the data read or to be written, and a memory scheduler that determines the order of attention of the memory requests. The memory scheduler is composed of one level of priority queues per memory bank and bank schedulers, and a second level consisting of a channel scheduler that issues the highest priority command of all ready commands from the bank schedulers. Figure 5 shows the Logical Structure of DRAM Controllers.

Memory controllers typically use First Ready First-Come First-Serve (FR-FCFS) [24] as their base scheduling policy.

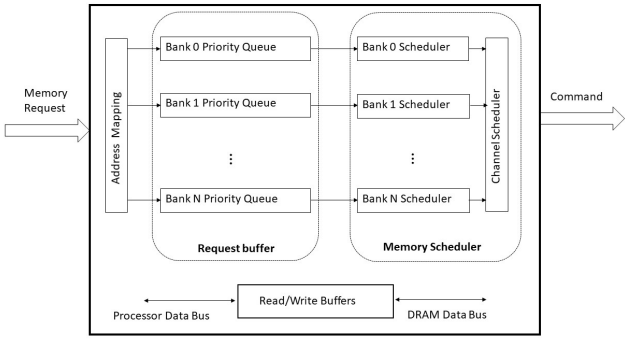


**FIGURE 5.** Logical Structure of DRAM Controllers.

FR-FCFS reorders memory requests so that row hit requests have higher priority than row miss requests, and in the case of a tie, older requests have higher priority. The scheduling algorithm implemented in the memory controller will influence the response time of memory requests [7].

The interference delay for a memory request can be classified into *interbank interference* and *intrabank interference*. In an architecture with dedicated bank partitions for each core, interbank interference occurs when a memory request command is sent to the channel scheduler. Memory request commands from other banks delay serving the request because the FR-FCFS policy issues ready commands. The intrabank interference occurs in an architecture where the cores share bank partitions when the request buffer for a memory request gets slowed down due to other, higher priority requests the bank's scheduler is servicing. [22].

## III. TECHNIQUES FOR REDUCING INTERFERENCE ON SHARED RESOURCES OF MULTICORE SYSTEMS

Several techniques have been proposed to reduce interference in shared resources of multicore systems. The proposed techniques select an isolation architecture early in the design life cycle, using other isolation mechanisms combining various media and software partitioning. Proposed techniques for interference analysis in software partitions include the division and allocation of the software in partitions according to its functionality, the use of an arbitration mechanism to guarantee that all the requirements are fulfilled when using a shared communication channel, the analysis of the potential for interference due to the use of a shared resource and the management of shared resources through the use of control mechanisms that restrict the number of resources that each partition can use, or monitoring mechanisms that employ corrective actions if needed [25], [26].

Interference reduction techniques allow a more predictable estimation of the WCET of programs in a multicore system [27]. Other approaches [28], [29] focus on limiting contentions, although they do not ensure the identification of the worst-case. Bin *et al.* [28] use two different measurement techniques (hardware Performance Monitoring Counters and the execution of a set of stressing benchmarks dedicated to stressing a particular shared hardware resource) to carry out a study of co-running real-time avionic applications on multicore COTS architectures. The fundamental objective is to characterize the application workload at the system level and obtain resource contention models at the hardware level. Evaluating these results allows identifying applications that can be executed together without modifying the WCET. Iorga *et al.* [29], explore the design and evaluation of empirical techniques to estimate the upper limits of the interference to which a program that runs on a multicore processor could be subjected from the implementation of enemy programs accessing shared resources to maximize contention. The application of these techniques allows comparisons between processors to evaluate their capabilities in scenarios that imply real-time restrictions.

In 2019, Maiza *et al.* [30] presented a survey of timing verification techniques for real-time multicore systems in the period from 2006 to 2018. The authors categorize the works from the point of view of the approach and divide the proposals discussed into four fundamental categories: full integration, temporal isolation, integrating interference effects into schedulability analysis, and mapping and allocation. Our survey categorizes the works from the point of view of the shared resource: main memory, cache memory, and shared memory bus. In this sense, our survey adds the discussion of new and relevant proposals not previously discussed, such as [31]–[58].

The following subsections focus on proposals for interference reduction techniques implemented in real-time multicore systems and proposals that use interference reduction techniques without considering the predictability issue. The proposals are divided according to the shared resource they focus on: i) main memory, ii) cache memory, and iii) memory bus.

### A. MAIN MEMORY INTERFERENCE

Modern multicore systems consist of various components, such as processing cores, prefetchers, and Direct Memory Access (DMA) engines, generating memory requests with different characteristics and priorities. Memory requests from multiple components on multicore systems with shared DRAM cause interference and system performance degradation.

Various approaches have been proposed to reduce contention due to shared resources. For example, the use of scheduling algorithms to allocate threads [59], approaches that use channel partitioning [60], and approaches that adjust their scheduling policy based on memory access behavior [61], [62].

Proposals focused on solving the problem of application execution delay due to memory access contention can be classified into spatial memory resource isolation and temporal memory resource isolation depending on how memory resources are isolated. Spatial memory resource isolation is performed by physically partitioning memory hardware resources. Temporal memory resource isolation

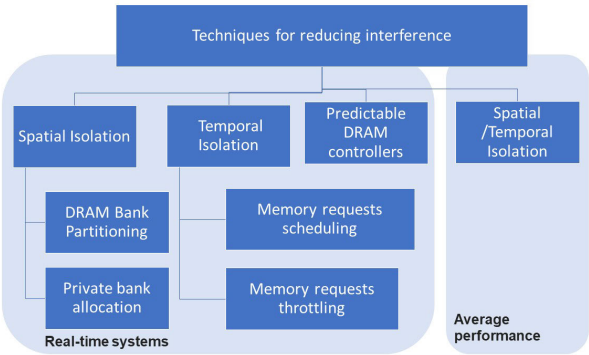


**FIGURE 6.** Distribution of proposals focused on shared main memory.

avoids interference by allocating a specific amount of time for applications to use memory hardware resources.

The proposals for interference reduction techniques in shared main memory are discussed below, divided into three fundamental categories: Spatial memory resource isolation, Temporal memory resource isolation, and Predictable DRAM controllers. In addition, the main proposals that use interference reduction techniques without considering the predictability issue are discussed. A taxonomy of the proposals focused on shared main memory is illustrated in Figure 6.

### 1) SPATIAL MEMORY RESOURCE ISOLATION

Proposals in this sub-category reduce interference in main memory by physically partitioning memory hardware resources. Several of the proposed approaches employ DRAM bank partitioning and private bank allocation techniques to make memory access more predictable [63]–[68].

A Single Core Equivalence (SCE) technology that implements a set of techniques at the OS level to provide isolation in obtaining the results of the schedulability analysis for each core was introduced in 2015 by Mancuso *et al.* [63]. Access to shared memory is regulated by applying Colored Lockdown - Cache Assignment, Memory Bandwidth Partitioning, and DRAM Bank Partitioning techniques. The WCET is estimated given the knowledge of the behavior of the task in isolation. Later, in 2017, Mancuso *et al.* [64] proposed a more precise analysis for obtaining the WCET considering the exact distribution of memory budgets to cores.

Yun *et al.* [65] introduced PALLOC, a DRAM bank-aware memory allocator. With PALLOC, the designer can assign private DRAM banks without requiring any hardware modification.

These approaches [63]–[65] do not consider non-uniform memory access (NUMA). In a NUMA system, the multicore architecture is divided into nodes of sets of cores with a memory controller. Pan *et al.* [66], [67] proposed a controller/node-aware memory coloring (CAMC) allocator. It divides the memory space into different sets (Colors) and assigns each memory bank a different color. The implemented mechanism allows assigning a private memory space for each task in its local memory node. In contrast to the previous TintMalloc proposal made by Pan *et al.* [68], the entire memory space is "colored" without requiring application modifications.

### 2) TEMPORAL MEMORY RESOURCE ISOLATION

Papers in this sub-category reduce interference by allocating a specific amount of time for applications to use memory hardware resources. Temporal memory resource isolation approaches can be divided into memory requests scheduling and memory requests throttling.

Memory requests scheduling determines the order of attention of memory requests, and memory request throttling varies the frequency of non-critical application memory requests to ensure proper operation of critical applications [69].

Alhammaed *et al.* [70] proposed an approach that reduces the timing interference when accessing the main memory by using a DMA component. The DMA operations scheduling algorithm allows that while a task is executing from one of the partitions of the core's local memory, the next task is placed in the other partition of the same size, avoiding time interference in the access to the main memory.

Kim *et al.* [69] proposed an operating system-level mechanism to reduce memory interference latency in dual-criticality systems dynamically. Using the cgroup of the Linux Kernel mechanism, critical execution tasks are dynamically identified and grouped. A memory request rate controller is added to the original cgroup mechanism. The memory interference experienced by the group of critical tasks is estimated from the number of pending memory requests in the memory controller. Based on the interference estimates experienced by the critical task group, the memory request rate controller reduces the number of memory requests from the non-critical task group. The implemented mechanism reduces latency in executing a critical application by reducing contentions in shared memory.

Other scheduling proposals are discussed in the subsection Integrating interference effects into schedulability analysis [22], [71]–[73].

### 3) PREDICTABLE DRAM CONTROLLERS

Papers in this sub-category focus on designing predictable DRAM controllers to produce tight WCET bounds [74]–[77].

Hassan *et al.* [74] focuses on reduced latency DRAM (RLDRAM) for multicore safety-critical real-time systems. The use of an RLDRAM capable of complying with stricter latency limits is proposed, and a memory controller design that predictably manages accesses to the RLDRAM is presented.

Ecco and Ernst [75] proposed a real-time memory controller that reorders read and write commands to obtain improved DRAM timing bounds. In 2017, Guo and Pellizzoni [76] proposed a DRAM Controller for Mixed-Criticality Systems. The proposal focuses on the one hand

on guaranteeing a tight upper-bounded HRT request latency through the use of a close-page policy for HRT requestors. As a result, the HRT requestor is not delayed due to interbank interference through a private bank scheme for HRT and SRTs and reordering of reading and write requests. On the other hand, it focuses on a configurable guarantee for SRT requests bandwidth, employing an open page policy for SRT banks and an FR-FCFS scheduling algorithm.

A comparative study of predictable DRAM controllers was presented in 2018 by Guo *et al.* [77]. The authors carry out a comparison of the proposed designs based on crucial configuration parameters such as the impact of the number of requestors on the analytical and simulated worst-case latency per memory request, the impact of row hit ratio, effects of the variation of data bus width, the impact of memory device frequency, etc. In addition, a discussion of the advantages and disadvantages of different controller architectures is presented, and the evaluation of tradeoffs between latency bounds provided to real-time tasks and average memory bandwidth offered to non-real-time tasks.

#### 4) PROPOSAL FOCUSED ON ACHIEVING A BETTER PERFORMANCE

The proposals in this subsection employ interference reduction techniques to achieve better performance on a multicore platform without considering predictability issues.

*Spatial memory resource isolation*: Different approaches have been proposed to limit the hardware resources shared by different cores. In 2010, Mi *et al.* [78], proposed for the first time a DRAM bank partitioning scheme to reduce intracore and intercore interference. The proposed approach uses the classic OS page-coloring technique [79] to partition DRAM banks combined with XOR cache mapping. In 2011, Muralidhara *et al.* [60], aimed to reduce interference by dividing application data into different channels based on application memory usage and implementing per-channel page allocation and memory request scheduling. Xie *et al.* [80] considered memory usage intensity and spatial locality of the applications to guide page policy assignments, and Liu *et al.* [81] implemented a bank-level partitioning mechanism based on page coloring, which assigns specific DRAM banks to each core.

In 2016, Jia *et al.* [82] proposed PUMA, a method that achieves isolation by partitioning memory banks and allocating memory banks and private bandwidth for each core. The threads are divided into groups, and 16 unique memory banks are assigned to each core. The bandwidth is allocated equally to each thread running simultaneously.

However, modifying the operating system in systems such as avionics would imply costs for security verification. To avoid this problem, Shen *et al.* [83] proposed the use of a row-based policy, but it does not consider the inter-thread memory interference. In 2015, Fang *et al.* [84] introduced an approach that reduces inter-thread interference without modifying the operating system. It proposed partitioning the DRAM memory banks and using an adaptive page policy that varies dynamically depending on the flow of memory access and the row buffer total hits or misses of each bank.

Other hardware designs have also been proposed to ensure that each core has access only to its assigned DRAM banks [85]. For example, a Decoupled Direct Memory Access hardware-software mechanism was proposed in 2015 by Lee *et al.* [85] to avoid the increasingly data-intensive applications cause contentions in the main memory channel by increasing the transfer of data from Input/Output (IO) devices to main memory, and the access of the CPU to main memory. The mechanism separates CPU and IO requests using two hardware components: dual data port DRAM and off-the-processor-chip control logic to reduce the contention problem.

*Temporal memory resource isolation*: In 2020, Fang *et al.* [42] proposed a memory access scheduling strategy to solve the problem of shared memory contention in systems where GPUs are used. The method consists of three steps: initially, the memory requests are separated into two queues in the memory controller, avoiding GPU memory access requests interfering with CPU requests. Then, depending on the memory access behavior of the applications, they are divided into classes and assigned dynamic bank partitions that allow eliminating interference between applications without affecting parallelism at the bank level. Finally, for GPU requests, a core criticality-aware scheduling is implemented.

In 2019, Rashid [86] modeled contention in main memory using a server-based approach. The modeling uses the concepts of notional processors/servers [87]. The main memory contention problem is modeled as a task assignment problem, based on the work proposed by [88]. Initially, a simple First Fit Decreasing Memory Demand (FFDM) heuristic is used; servers are ordered by non-increasing memory demands and assigned to cores using the First-Fit heuristic. Later, two algorithms were proposed to improve the initial solution: select the best neighborhood solutions and use a simulated annealing-based meta-heuristic approach. Experimental results show that feasible solutions with these approaches require a total number of servers per core below a certain threshold.

#### 5) SUMMARY

Interference reduction techniques in multicore systems with shared main memory focus on achieving isolation of the shared resource: temporal or spatial isolation. However, spatial memory resource isolation approaches can cause memory resources to be underutilized. Furthermore, if multiple applications simultaneously are assigned the same memory partition, the interference in accessing the shared resource will not be eliminated. Temporal memory resource isolation techniques can be divided into memory request scheduling and memory request throttling. Techniques that focus on memory request scheduling require a hardware modification of the memory controller.

**TABLE 1. Comparative Table of Interference Reduction Techniques due to Shared Main Memory in Multicore Systems.**

| Approaches | Technique | Spatial/Temporal/Predictable Controller | Soft/Hard/none | HRT/SRT/AVG |
|---|---|---|---|---|
| [63], [64] | Single Core Equivalence | Spatial | Software | SRT/HRT |
| [65] | Bank-Aware Memory Allocator | Spatial | Software | SRT |
| [66]–[68] | Controller Aware Memory Coloring | Spatial | None | SRT |
| [70] | Scheduling algorithms | Temporal | Hardware | HRT |
| [69] | Memory request throttling | Temporal | Software | SRT/HRT |
| [74]–[77] | Predictable DRAM Controllers | Predictable Controller | Hardware | SRT/HRT |
| [78], [81], [82] | Bank Partitioning | Spatial | Software | AVG |
| [60] | Channel Partitioning | Spatial | Soft/Hard | AVG |
| [80] | Page Policy Control | Spatial | Soft/Hard | AVG |
| [83] | Page Policy Control | Spatial | None | AVG |
| [84] | Bank Partitioning | Spatial | None | AVG |
| [85] | Decoupled Direct Access | Spatial | Hardware | AVG |
| [42] | Scheduling algorithms | Temporal | None | AVG |
| [86] | Task allocation | Temporal | Software | AVG |

Table 1 shows a summary of the proposals discussed above, indicating in each case the implemented technique, if it allows temporal or spatial isolation, whether it requires software or hardware modifications and its use in hard real-time systems, soft real-time systems, or average-case if it is a proposal focused on achieving better performance on the multicore platform and are not proposed for real-time systems.

Initial proposals focus on scheduling algorithms to allocate threads, channel partitioning, and scheduling policy based on memory access behavior. However, although thread scheduling alleviates contention, it does not eliminate bank interference among threads, channel partition does not provide optimal bandwidth usage, and thread-aware memory scheduling requires a hardware modification.

In the 2015-2020 period, approaches to handling interference in real-time multicore systems have mainly focused on DRAM bank partitioning techniques, private DRAM banks, memory request scheduling, and memory request throttling. To estimate the WCET as accurately as possible, SCE [63] employs Memory Bandwidth Partitioning and DRAM Bank Partitioning techniques. In addition, the authors provide a methodology for response time analysis; however, they assume the worst-case of the memory access pattern, and the WCET estimate shows a certain degree of pessimism. Adding to the research, the exact knowledge of the budget to core assignments allows a more accurate estimate.

The allocation of private DRAM banks implemented in PALLOC [65] improves isolation and real-time performance but requires exact knowledge of the DRAM controller address mapping.

CAMC [66], [67] considers multiple memory controllers and assess its performance on a NUMA architecture. It does not require any code modifications for applications and improves performance and timing predictability through bank isolation techniques. On the other hand, it cannot provide a Single Core Equivalence for multicore executions per controller and does not guarantee compositionality for policies with interference.

The memory request scheduling technique proposed by Alhammaed *et al.* [70] manages contention among cores for access to main memory, but its implementation requires hardware modifications. On the other hand, the memory request throttling technique proposed by Kim *et al.* [69] does not require hardware modifications, and its application in a mixed critical system allows to reduce the memory interference latency of critical applications dynamically. Finally, design proposals for DRAM Controllers are presented to make main memory access more predictable.

The most relevant proposals for interference reduction techniques focused on achieving a better performance include Memory Bandwidth Partitioning and DRAM Bank Partitioning involves or not modifying the Operating System or implementing hardware designs. Techniques centered on memory request scheduling, memory request throttling, dynamically assigned page policy, and assigning tasks to cores based on memory resource demand have also been proposed.

### B. CACHE INTERFERENCE

One of the main factors that cause unpredictability in multicore systems is shared cache. Cache partitioning and locking are the most commonly used approaches to guarantee more deterministic behavior. The surveys [17] and [89] presented the main cache partitioning and cache locking techniques proposed from 1990 to 2014. Our survey adds proposals for cache partitioning and cache locking techniques not previously discussed, such as [90]–[113]. In addition, proposals for cache management that implement new cache replacement algorithms and predictable cache coherence protocols are discussed.

Proposals to reduce interference due to shared cache in real-time multicore systems are discussed below, divided into four fundamental categories: Cache-Partitioning techniques, Cache-locking techniques, Designing Predictable Cache Coherence Protocols, and Other approaches. In addition, the main proposals that use interference reduction techniques without considering the predictability issue are discussed. Finally, a taxonomy of proposals focused on shared cache is illustrated in Figure 7.

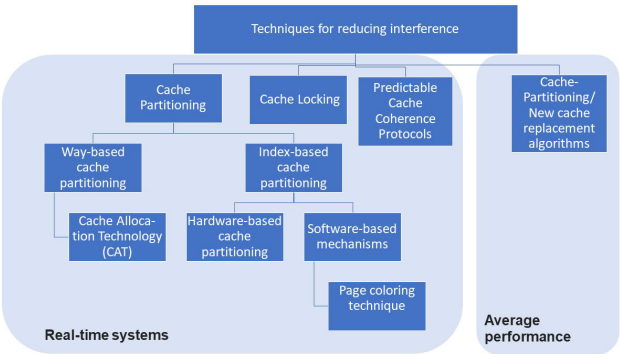


**FIGURE 7.** Distribution of proposals focused on shared cache memory.

### 1) CACHE-PARTITIONING TECHNIQUES

Papers in this sub-category reduce shared cache interference by employing the cache partitioning technique. The central idea of the cache partitioning techniques is the segmentation of cache space and the allocation of a memory partition to a particular task or core to avoid interference.

Cache partitioning techniques can be characterized as static or dynamic depending on whether memory allocation is offline or runtime. [89]. Static partitioning techniques have greater predictability since each task has a fixed cache partition. However, fixed-size partitions cause low cache utilization and consequently reduced performance. On the other hand, in dynamic partitioning techniques, the size of the allocated cache partitions vary during runtime, giving high cache utilization and causing lower predictability [98].

Furthermore, cache partitioning techniques can be characterized as index-based partitioning and way-based partitioning based on the structure of a set-associative cache. The way-based cache partitioning methods use a specific hardware implementation. Index-based cache partitioning methods are divided into hardware and software-based techniques. The hardware methods of index-based cache partitioning require special hardware support, and the most widely used of the software-based techniques is page coloring [17].

*Way-based cache partitioning methods*: Way-based cache partitioning has the advantage of low hardware cost; however, it is limited by the number of partitions and granularity of allocations due to cache associativity.

In 2015, Intel Corporation [90] introduced Cache Allocation Technology (CAT). This technology allows the reservation of cache portions for individual cores. Xu *et al.* [91] present a design for dynamic shared cache management on multicore virtualization platforms based on CAT, and Satka *et al.* [92] evaluated the performance of CAT configured in different ways based on the purpose of the usage. In 2018, Pons *et al.* [93] proposed a Critical-Aware Partitioning Approach (CA) on multicore processors. The proposed technique uses CAT to partition the cache and assign the partitions to applications. The central idea of the proposed partitioning mechanism is to divide the LLC into two partitions: a larger one for critical applications and a smaller one for the rest. CA resizes partitions dynamically based on critical application behavior.

*Index-based cache partitioning methods*: For real-time systems, in 2005, Chousein and Mahapatra [114] propose a hardware-based cache partitioning mechanism for a fully associative cache architecture, and in 2008, Suhendra and Mitra [115] suggest the use of a combination of cache partitioning and locking mechanisms.

Lee *et al.* [98] presented a dead-block-based cache partitioning technique on a new shared LLC architecture. The proposal focuses on reducing the deadline miss rate of Time-sensitive tasks (TSTs). The method requires an extension of the shared cache by adding a time-sensitivity field and a decay counter to each cache block. The first one indicates whether the cache block belongs to a TSTs, and the second is used to detect dead blocks. TATS divides the cache space into as many segments as TSTs, assigns each of the tasks to each segment, and then assigns each General Task(GT) a segment. It also implements a matching algorithm to decide which GT shares the same cache segment with each TST. Experimental results show lower TST deadline miss rates than the values obtained with existing shared cache methods.

Software-based cache partitioning approaches have also been implemented [100], [103]–[105], [116]. Most of them are based on a page coloring technique.

Ward *et al.* [116] applied several cache management schemes based on page coloring in a mixed-criticality scheduling framework. They consider the colors assigned to physical memory pages as shared resources and use cache scheduling and locking techniques to arbitrate access to these resources. Kim *et al.* [100] presented an extension to the MC2 framework (mixed-criticality on multicore) [116]–[118]. The proposed variant employs page coloring to eliminate interference within the LLC and memory banks, providing LLC and DRAM-bank isolation. It also provides operating system isolation of highly critical tasks in the LLC via way-based partitioning. Depending on their criticality level, DRAM memory banks and LLC areas are assigned to certain task groups.

Kim *et al.* [103] proposed a cache management framework for multicore virtualization. The problem of assigning cache to each task in a virtual machine is solved by using two new hypervisor-level techniques: vLLC and vColoring. vLLC allocates a portion of the host machine's LLC cache in the form of a virtual LLC to a virtual machine running a guest operating system with page coloring support. vColoring allows the hypervisor to directly assign a cache portion of the host machine's LLC to a task on a virtual machine running a guest operating system without support for page coloring. These cache management schemes focus on homogeneous multicore processors. Lim *et al.* [104] presented a cache management scheme using partitioning hypervisors for clustered multicore embedded platforms. The proposal considers the cache underutilization present in cluster-unaware page coloring-based techniques and the

cache interference caused by shared memory regions in inter-VM (virtual machine) communication.

Kloda *et al.* [106] used the page coloring technique to partition cache and DRAM through virtualization support. The proposed implementation considers dynamic coloring and the analysis of coloring side effects such as the reduction of available memory space and the undesired partitioning of other cache levels, among others. Finally, they propose a strategy for cache allocation based on the principles of the cache replacement policy. The technique consists of invalidating a series of lines that are expected to be used by the task that starts or resumes its execution.

Panchamukhi *et al.* [107] implemented a new heap allocator that uses page coloring to provide software partitioning of the translation lookaside buffer (TLB). This approach enables task isolation by assigning a memory region of a single color to a task and ensuring that pages of different colors do not map to the same TLB set.

Bouquillon *et al.* [108] proposed to partition the cache using page coloring combined with Integer-Linear-Programming (ILP) techniques. Each task is assigned a certain number of colors, keeping the total of assigned colors less than or equal to the total available in the cache. Two heuristic algorithms are used, one that assigns the same number of pages to each color and the other where a task that has j colors assigns a different color to the first j-1 pages ordered by decreasing punctuation. The cache space distribution is represented as a Multiple Choice Knapsack Problem (MCKP) [119] and encoded as an ILP problem.

A cache management technique that combines set and way partitioning was proposed in SWAP [109]. SWAP partitions the shared cache in a two-dimensional structure providing hundreds of fine-grained cache partitions. The SWAP implementation introduces small changes to the Linux page allocator and employs ThunderX's native architectural support for way partitioning.

Cache partitioning techniques are used to reduce interference in the shared cache. However, according to Valsan *et al.* [111], in non-blocking caches, partitioning techniques do not guarantee predictable behavior due to contention in the miss status holding registers (MSHR) [120], where the requests cache-miss on a non-blocking cache are registered. In [111] a mechanism based on hardware and software is proposed to limit these contentions. It is considered a multicore system with identical cores where each core has a private level 1 cache without blocking. MSHR contention on the private cache is removed by limiting the maximum number of MSHR entries to be used by all private caches of all cores to be equal to or less than the number of MSHRs in the shared cache.

### 2) CACHE-LOCKING TECHNIQUES

Papers in this sub-category reduce shared cache interference by employing the cache-locking technique. The use of cache locking techniques allows access time to be accurately known through accurate hit/miss prediction for cache accesses, enabling the use of the cache in hard real-time systems.

Cache-locking techniques prevent cache lines from being replaced by marking them as locked until an unlock operation is executed. It is an effective technique to reduce the WCET of a task by locking the appropriate contents in the caches. However, effective use of cache locking techniques requires a good selection of the contents to lock to avoid the increase of cache misses that cause loss of performance and energy, as well as addressing the solution of several critical issues in replacement policies, cache coherence schemes, and the reduction of the available space for unlocked blocks [115], [121], [122].

In the work presented in 2015 by Gracioli *et al.* [17], the main cache locking proposals from 1990 to 2014 are discussed [115], [123]–[125]. These proposals present static and dynamic cache locking techniques, showing better predictability when using static locking and better utilization with dynamic locking. The content to lock in the cache must be carefully selected, requiring computation-intensive algorithms such as ILP. Various cache locking techniques have been implemented to reduce cache interference problems [125], [126]. However, they require hardware support, which is not available in many commercial processors for embedded systems [122].

An interesting proposal for cache locking techniques in multicore systems was made in 2010 by Asaduzzaman *et al.* [123]. In this work, from the processing of the results of the WCET analysis, a miss table is implemented at the L2 cache level that contains the information of the blocks that cause the most misses when the cache locking technique is not used. The locking technique uses the information provided in this table to select the blocks to be locked efficiently. This implementation improves predictability and reduces overall energy consumption.

Sarkar *et al.* [124] proposed a predictable task migration scheme using cache locking; several cache migration models were developed to determine task migration delay. Later, in 2015, Sarkar *et al.* [127] proposed a static task partitioning that assumes that private L2 caches are large enough to hold the data space and instructions of hard real-time tasks and considers fixed latencies for L1 locked memory regions to derive safe WCET limits. Tighten WCET bounds are subsequently used for static assignment of tasks.

Dugo *et al.* [112] have proposed algorithms for ARINC-653 Compliant RTOS to implement static cache locking content selection to reduce the non-determinism and the contention on lower-level memories while improving timing performance. Zhang *et al.* [113] investigated the WCET-aware Instruction cache (I-cache) locking problem and proposed an ILP-based dynamic I-cache locking approach for reducing the WCET of a task. Their approach not only selects locking contents that have the most significant benefit in reducing a task's WCET but also finds a good locking point for each locked instruction so that extra execution time spent on locking instructions is also minimized.

#### 3) DESIGNING PREDICTABLE CACHE COHERENCE PROTOCOLS

Papers in this sub-category study the problem of data interference in a system where data is shared through cache coherence protocols. Cache coherence protocols specify the activity of one core on shared cached data as a function of the activity of other cores on the same data.

Kaushik *et al.* [128] extends the previous proposal made by Hassan *et al.* [129] focused on a predictable cache coherence protocol based on the use of certain invariants applied to the classic MSI protocol [130]. The proposal made by Kaushik *et al.* [128] focuses on design invariants for the MESI protocol [130] from analysis of the unpredictable behavior of the conventional MESI coherence protocol.

Cache coherence is the most frequently used technique in COTS architectures [131], [132] and compared to previous solutions it shows better performance [129], [133]. However, it presents a notably high worst-case memory latency. The work presented in 2010 by Fuchsen [134] shows a measure of the interference caused by the coherence protocol. Hassan *et al.* [135] propose DISCO: a discriminative coherence protocol considering the importance of data exchange in real-time systems. This protocol provides high performance by allowing simultaneous access to shared resources and provides adjusted latency limits, eliminating scenarios that cause high coherence interference.

#### 4) OTHER APPROACHES

Most approaches to implementations that ensure QoS require hardware changes, such as cache partitioning. Sivakumaran and Siromoney [136] presented a different approach requiring no hardware modifications to guarantee QoS from the cache to performance-critical programs. First, pairs of programs are established, and the effects of shared cache interference are evaluated. Subsequently, a critical program must be prioritized in each pair, and another penalized program whose performance can be degraded is considered. Static and dynamic methods are proposed to reduce the cache miss rate and guarantee the adequate functioning of the critical program. The dynamic approach is more optimal than the static method because it prevents the penalized program from disabling cache usage when the critical program requires little memory.

#### 5) PROPOSAL FOCUSED ON ACHIEVING A BETTER PERFORMANCE

Papers in this sub-category study interference reduction techniques to achieve better performance on the multicore platform without considering predictability issues. The most relevant proposals include cache partitioning techniques and the design of new cache replacement algorithms.

*Cache-Partitioning techniques*: In the work presented in 2015 by Gracioli *et al.* [17], the main proposals for way-based cache partitioning from 1990 to 2014 are discussed. The proposals discussed for multicore systems in this period [137]–[141], according to [17], were not originally proposed to improve the predictability of real-time systems and all use a specific hardware implementation.

Lee *et al.* [94] propose a way-based cache partitioning scheme for a shared LLC with the limited associativity: Selective Cache Partitioning (SCP). This technique allows the allocation of private or dynamic partitions at runtime, focused on isolating applications with poor locality on private partitions and allocating applications with the low or medium locality to a shared partition.

Selfa *et al.* [95] introduced a hardware-based cache partitioning approach that reduces shared cache interference by assigning a private cache partition to each application. The size of each cache partition is dynamically adjusted at runtime according to the requirements of each application during its execution. Partitions can only vary by one unit of cache way between two consecutive intervals to simplify hardware implementation. The implemented technique assigns a larger cache partition size to applications that suffer more slowdown.

El-Sayed *et al.* [96] presented Kpart. For offline or online memory profiling, Kpart employs a novel technique that estimates cache demand per application by periodically sampling different partition sizes across co-running applications and using basic performance counters. Kpart groups the applications into clusters from this information and later divides the cache space between these clusters.

Sun *et al.* [97] proposed a mechanism to reduce inter-core interferences caused by hardware prefetchers on shared cache multicore processors. The execution of the process is divided into two stages: an initial phase where the necessary statistics are compiled to identify the aggressive prefetch applications, and the second stage of resource allocation. Subsequently, three approaches to resource allocation are considered: Prefetch Throttling, Cache Partitioning, and Coordinated-throttling. In the way-based cache partitioning approach, they proposed to place the Aggressive Prefetch set in a small partition or divide the Aggressive Prefetch set into two subsets, prefetch friendly and unfriendly, and assign each subset a separate section. Cache partitions combined with prefetch throttling improve system performance.

The main proposals for index-based cache partitioning from 1990 to 2014 are discussed in the work presented in 2015 by Gracioli *et al.* [17]. Proposals for multicore systems in this period include cache partitioning mechanisms index-based with hardware-specific implementations [142]–[145] and software-implemented index-based cache partitioning mechanisms [116], [146]–[150].

In 2013, Suzuki *et al.* [99] introduced a coordinated cache and bank coloring mechanism to avoid cache and bank interference simultaneously in multicore systems. The authors develop a virtual page management mechanism that classifies pages in a two-level structure: their cache colors and bank colors. Pages with the same cache and bank colors are linked to a memory cell. Once the cache and bank colors are assigned to a task, it only has access to the

physical pages of the corresponding memory cell. In 2014, Ye *et al.* [101] introduced a memory management framework called COLORIS. COLORIS can create static and dynamic cache partitions using the page coloring technique.

The traditional page coloring technique is inefficient on multicore architectures that modify the physical addressing scheme to a more complex scheme that involves a hash function. In 2016, Scolari *et al.* [102] expanded the initial page coloring technique by implementing a mechanism capable of handling this addressing scheme. Similarly, page coloring techniques do not work correctly when virtualization techniques are used in real-time systems. The advantages of virtualization in embedded systems, such as reduced cost, space, and power requirements, have recently extended page coloring techniques to the virtualization environment. Multiple real-time systems are consolidated onto a single hardware platform with virtualization, running on spatially isolated virtual machines.

In 2020, Park *et al.* [105] proposed a dynamic cache partitioning mechanism that employs the page coloring technique to partition the cache space into two areas. The cache pages in each area will be based on page reuse data, divided into highly reused and slightly reused pages. In this way, it is possible to maximize the use of the cache by assigning a more significant portion to the group of highly reused pages. The proposal does not require additional hardware support, application modifications, or prior knowledge of workloads.

In 2019, Danielsson *et al.* [110] proposed a run-time cache partition controller that optimizes the cache partitioning approach proposed by Palloc [65] by obtaining more efficient partitions using correlation-based decisions. To create efficient partitions, it focuses on LLC-bound workloads. The controller regulates the size of the allocated cache partition based on the correlation between a performance metric and the increase in cache size for a specific application. A saturation point is established in the correlation decrease, after which the driver stops assigning cache partitions to the application.

*Designing a new cache replacement algorithms*: In the cache, when all blocks in a set are complete, and a new block needs to be placed from the main memory, according to the implemented replacement policy, the cache controller selects a cache line and replaces it with the new block. Among the most common cache replacement algorithms are first-in-first-out (FIFO), Least Recently Used (LRU), the Least Frequently Used (LFU), Not Recently Used (NRU), and the Dynamic Aware Insertion Policy (DIP). In practice, Pseudo-LRU (PLRU) is commonly used for high associativity caches since it requires fewer hardware gates to implement compared to LRU. Based on those basic policies, several approaches have been proposed aimed at optimizing replacement algorithms [151], [152]. However, most of the former works do not consider the interactions between cores and access patterns and are unsuitable for partially shared cache structures. Thus, new approaches have been proposed more recently to cope with this problem.

Pai *et al.* [153] proposed an algorithm that selects the victim based on the block's recency value, the behavior of the application, and the cost of the block. The cost of each block based on Memory Level Parallelism (MLP) is calculated using the number of LLC accesses to the main memory. The current information is obtained using recency counters at each block. The proposed replacement policy considers the MLP behavior of the application and the data reuse behavior, which results in decreased interference between applications using the shared cache.

Warrier [154] proposed an Application-aware Cache Replacement (ACR) policy. During cache replacement, the victim is selected, taking into account the hit-gap of the applications. The article presented the flow chart for victim selection and concluded that the victim is the cache line with the least time to live and the least recently used. Furthermore, compared to replacement policies: LRU policy, thread-aware dynamic RRIP (TA-DRRIP) [151] policy and protecting distance-based replacement policy (PDP) [155], ACR showed a performance improvement and a decrease in cache misses.

Yang *et al.* [156] proposed a core-aware re-reference interval prediction (CA-RRIP) replacement algorithm on an NoC-based multicore structure with a partially shared cache. A new cache structure is proposed that integrates the advantages of the on-chip bus with an L2 cache partially shared between all cores. The algorithm establishes a counter (RRIV) of N bits for each cache block that indicates the prediction of reuse of the cache block, the higher the value of the counter, the lower the probability of reuse of the block. It also records the identification of the accessing core and selects blocks to be replaced according to the core identification, resulting in dynamic virtual partitions of the shared cache.

#### 6) SUMMARY

Proposed approaches to reducing interference due to shared cache memory in real-time multicore systems employ cache partitioning techniques, cache locking techniques, and predictable cache coherence protocols.

Cache partitioning can be index-based partitioning or way-based partitioning. The way-based cache partitioning proposals for real-time systems use CAT for cache partitioning and allocating cache partitions to applications. The use of CAT allows mitigating shared cache interference. On the other hand, its use requires a processor that supports this technology, and the restrictions regarding cache partitions must be taken into account (available in Intel's CAT specification).

Proposals for index-based cache partitioning employ hardware-based or software-based mechanisms. For example, the hardware-based approach by Lee *et al.* [98] uses a cache partitioning technique that considers the memory access characteristics and time-sensitivity of tasks. The proposed scheme requires task profiling.

**TABLE 2. Comparative Table of Cache Partitioning Techniques.**

| Approaches | Technique | Index/Way | Soft/Hard/none | Static/Dynamic | HRT/SRT/AVG |
|---|---|---|---|---|---|
| [90]–[92] | Cache Allocation Technology (CAT) | Way | Hardware | Static/Dynamic | SRT/HRT |
| [93] | Critical-Aware Partitioning Approach | Way | Hardware | Dynamic | SRT/HRT |
| [114] | CAM and TCAM cells | Index | Hardware | Static | HRT |
| [115] | HW or SW partitioning | Index | Hard/Soft | Static | HRT |
| [115] | HW or SW partitioning | Index | Hard/Soft | Static | HRT |
| [149] | Page-coloring with scheduling | Index | Software | Static | HRT |
| [116], [150] | Page coloring | Index | Software | Static | RT/HRT |
| [98] | Dead-block based cache partitioning | Block based | Hardware | Dynamic | SRT |
| [100] | Page coloring,way based partitioning | Index/Way | Soft/Hard | Static | HRT/SRT |
| [103], [104] | Hypervisor level techniques | Index | Software | Static | SRT |
| [106] | Hypervisor level techniques | Index | Software | Dynamic | SRT |
| [107] | TLB Coloring | Index | Software | Dynamic | HRT/SRT |
| [108] | WCET-aware cache coloring | Index | Software | Static | SRT |
| [137] | Dynamically partition | Way | Hardware | Dynamic | AVG |
| [138] | Intra-Application Cache Partitioning | Way | Hard/Soft | Dynamic | AVG |
| [139] | Region Aware Cache Partitioning | Way | Hardware | Static | AVG |
| [140] | Molecular Caches | Way | Hard/Soft | Static/Dynamic | AVG |
| [141] | Utility-based cache partitioning | Way | Hardware | Dynamic | AVG |
| [94] | Selective Cache Partitioning | Way | Hard/Soft | Dynamic | AVG |
| [95] | Fair-Progress Cache Partitioning | Way | Hardware | Dynamic | AVG |
| [96] | Hybrid Cache Partitioning-Sharing | Way | Hard/Soft | Dynamic | AVG |
| [97] | Prefetch Control and Partitioning | Way | Hard/Soft | Dynamic | AVG |
| [142] | Cache controller table | Index | Hardware | Dynamic | AVG |
| [143] | Set pinning,Processor Owned Private | Index | Hardware | Dynamic | AVG |
| [144] | Selective Cache Allocation | Index | Hardware | Static/Dynamic | AVG |
| [145] | Cache quota management mechanism | Index | Hardware | Static | AVG |
| [146]–[148] | Page coloring | Index | Software | Static/Dynamic | AVG |
| [99] | Bank and cache coloring | Index | Software | Static | AVG |
| [101] | Page coloring | Index | Software | Static/Dynamic | AVG |
| [102] | Page coloring | Index | Software | Static | AVG |
| [105] | Page Reusability | Index | Software | Dynamic | AVG |
| [109] | Set and WAy Partitioning | Index/Way | Soft/Hard | Static/Dynamic | AVG |
| [110] | Cache partition controller | Index | Soft/Hard | Dynamic | AVG |

Software-based cache partitioning approaches are fundamentally based on the page coloring technique. Ward *et al.* [116] consider page coloring techniques to eliminate or control cache conflicts. For implementing the proposal in real-time systems, they assume that the memory used by a task is pre-allocated and the tasks have been divided into processors before the task system begins execution. Other proposals use the page coloring technique in hypervisors to enable the cache allocation to tasks executed in a virtual machine. In this case, to ensure timing predictability, Kim *et al.* [103] assume that each virtual machine has been allocated a sufficient number of physical host pages and that page swapping does not occur at run time. The evaluation of the cache-aware real-time virtualization for clustered multicore platforms technique by Lim *et al.* [104] shows that cache partitioning and allocation prevent cache interference in real-time virtualization. On the other hand, the evaluation results of virtualization-based coloring overhead shown by Kloda *et al.* [106] indicate, in the presence of external interference, a considerable overhead in the latency of a memory operation when the memory footprint is considerably larger than L2 and memory accesses are randomly distributed. Tests carried out by the authors indicate that a miss in the translation look-aside buffer (TLB) is the cause of this overhead. The heap allocator proposed by Panchamukhi *et al.* [107] increases TLB predictability. On the other hand, the use of the page coloring technique combined with ILP and the cache memory partition according to the needs of a task show an increase of high utilization tasks set schedulable compared to a random partition [108]. Other cache management techniques combine set and way partitioning to achieve effective cache partitioning [100], [109].

Table 2 shows a summary of the approaches mentioned above, specifying in each case the implemented technique if it is implemented in hardware or software, the structure of a set-associative cache if a dynamic or static partitioning technique is implemented, and its use in hard real-time systems, soft real-time systems or average-case if it is a proposal focused on achieving better performance on the multicore platform and not proposed for real-time systems.

The second approach proposed to reduce shared cache interference is locking techniques. These techniques allow the use of cache in hard real-time systems, ensuring predictability in access time. For example, previous investigations lock the memory line with the highest execution frequency, lock a part of the cache statically for each task, lock the most used data or employ partial cache locking techniques, among others.

On multicore systems, cache locking techniques require prior knowledge of the entire memory footprint of the task, which is complex and even impossible. Furthermore, cache

**TABLE 3.** Comparative Table of Cache Locking Techniques.

| Approaches | Technique | Static/Dynamic | Granularity of locking | HRT/SRT/AVG |
|---|---|---|---|---|
| [115] | Cache partitioning and locking | Static/Dynamic | Cache block | HRT |
| [125] | Cache partitioning and locking | Static/Dynamic | Memory pages | HRT |
| [123] | Addition of a miss table | Static | Memory blocks | SRT/HRT |
| [126] | Locked Cache Migrations | Static | Cache lines | HRT |
| [127] | Cache-Aware Task Partitioning | Static | Cache lines | HRT |
| [112] | Algorithms to select content to lock | Static | Cache lines | HRT |
| [124] | Locked caches | Dynamic | Cache lines | HRT |
| [113] | ILP-based dynamic I-cache locking | Dynamic | Cache lines | SRT |

**TABLE 4.** Comparison Table of Cache Replacement Algorithm.

| Approaches | Cache replacement policy | Victim selection technique | HRT/SRT/AVG |
|---|---|---|---|
| [153] | Application behavior aware policy | Recency counters at each block | AVG |
| [154] | Application-aware policy | Taking into account the hit-gap of the applications | AVG |
| [156] | (CA-RRIP) replacement algorithm | Prediction of reuse of the block | AVG |

**TABLE 5.** Comparison Table of Predictable Cache Coherence Protocols.

| Approaches | Predictable cache coherence protocol | Description | HRT/SRT/AVG |
|---|---|---|---|
| [129] | Predictable MSI | Design invariants to ensure predictability | SRT |
| [128] | Predictable MESI | Design invariants to ensure predictability | SRT |
| [135] | Discriminative coherence protocol | Reduces the coherence delays | SRT |

locking techniques with pessimistic assumptions reduce the cache space available for other tasks, leading to a degradation of performance.

The most recent proposals (2015-2020) employ locking techniques to tighten WCET bounds further. Sarkar *et al.* [127] use these narrowed WCET bounds to allocate tasks and increase the predictability of memory accesses statically. Dugo *et al.* [112] focus on cache content selection algorithms for cache locking. This approach improves the hit rate and predictability of the caches. However, the results show an increase in cache misses due to an overlock situation when only 60 percent of the cache is locked. Zhang *et al.* [113] also focus on selecting a set of memory blocks from the task code as locked cache content. The WCET reduction obtained with this approach varies depending on the cache size. Fewer improvements are obtained when the cache size is relatively small, such as 256B. Increasing the cache size allows for better results. However, once most blocks of memory that are valuable for locking are selected, an increase in the cache size does not allow significant improvements.

Table 3 shows the most recent approaches to cache locking techniques in multicore systems classified according to the granularity of locking and its static or dynamic characteristics.

Finally, approaches have been proposed to deal with the problem of data interference in a system where data is shared by implementing Predictable Cache Coherence Protocols. The predictable MSI (PMSI) protocol proposed by Hassan *et al.* [129] and the cache coherence protocols (PMSI and PMESI) presented by Kaushik *et al.* [128] provide considerable performance improvements, do not impose any scheduling restrictions, and do not require any source-code modifications. On the other hand, implementing these invariants to deal with unpredictable scenarios requires architecture changes and the conventional coherence protocol. In 2020, the discriminative coherence solution proposed by Hassan [135] achieved lower latency bounds compared to the state-of-the-art predictable coherence protocol. If the system on which it is implemented does not support write-through caches or cache bypassing, its implementation requires modifying the cache controller to adopt the coherence protocol.

Table 5 groups the predictable cache coherence protocols papers reviewed considering the cache coherence protocol proposed.

The most relevant proposals for interference reduction techniques focused on achieving a better performance include cache partitioning techniques and the design of new cache replacement algorithms. The most recent research proposes replacement policies that take into account the memory access behavior of the applications during the victim selection process: using recency counters at each block, taking into account the hit-gap of the applications or algorithms that use indicators the prediction of reuse of the cache block.

Table 4 shows a summary of cache replacement algorithms considering the cache replacement policy and the victim selection technique used.

### C. MEMORY BUS INTERFERENCE

Proposals to reduce shared memory bus interference in real-time multicore systems are discussed below, divided into four fundamental categories: Memory Bandwidth Regulator, Phased Execution Model, Offline Scheduling, and Hardware isolation. In addition, the main proposals that use interference

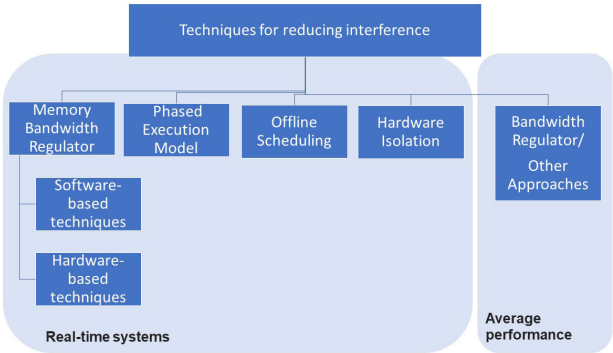


**FIGURE 8.** Distribution of proposals focused on shared memory bus.

reduction techniques without considering the predictability issue are discussed. Finally, a taxonomy of the proposals focused on the shared memory bus is illustrated in Figure 8.

### 1) MEMORY BANDWIDTH REGULATOR

Proposals that follow this approach are based on using a memory bandwidth regulator that allocates a portion of the total bandwidth to each core. Each task runs with a specific bandwidth independently of the execution of other tasks in other cores [30]. Memory bandwidth regulation approaches have been implemented at the software and hardware levels.

Software-based techniques monitor each core's memory access requests and regulate the bandwidth allocation of each core when it exceeds the allocated value. Several previous works have focused on memory bus bandwidth control at the operating system level [32], [88], [157]–[160].

MemGuard [88], [157] partitions memory bandwidth across all cores and enforces memory bandwidth allocation. After the bandwidth allocated to a core is exhausted, MemGuard suspends computation on that core. At the beginning of the following regulation period, a new assignment is made, and the stopped tasks are resumed. MemGuard also allows sharing the unused bandwidth of the total bandwidth allocated to each core among the rest of the cores with the highest memory consumption. In 2018, Agrawal *et al.* [160] presented a framework to determine the worst-case response time of a task in a real-time multicore system with the temporal dimension of memory scheduling and dynamic memory bandwidth regulation. A bandwidth partitioning scheme such as MemGuard is used. The worst-case response time calculation is determined from a maximization problem based on allocating time slots with different bandwidths to CPU and memory requirements.

BWLOCK [158] proposed a memory access control mechanism designed to prioritize the performance of soft real-time applications over other non-real-time applications. BWLOCK's main goal is to reduce memory bandwidth contention selectively. Through an Application Programming Interface (API), the application specifies the parts of the program that are memory performance-critical sections (MPCSs), and BWLOCK eliminates the contention of bandwidth by regulating the allocations of memory bandwidth of the rest of the cores that execute memory-intensive, non-real-time (NRT) applications.

Agrawal *et al.* [159] proposed a software mechanism for dynamic memory bandwidth allocation as an enhancement to Airbus' research on the use of COTS multicore for avionics applications. Starting from the worst-case of memory access patterns, it creates offline schedule tables that allocate partitions and dynamic memory bandwidth to each core. Two servers running on each core jointly control the number of memory accesses and contention between cores at runtime.

Xu *et al.* [32] presented CaM, a strategy for allocating cache and memory bandwidth resources. CaM focuses on partitioned scheduling, assigned tasks to cores, and shared cache resources and memory bandwidth are divided and assigned to each core. To optimally assign the tasks to each core, the interdependence between a task's WCET and its cache partition and memory bandwidth allocation is investigated. Finally, a heuristic resource allocation algorithm is proposed from the analysis of the experiments carried out.

On the other hand, hardware-based techniques have been implemented to regulate the assigned bandwidth. However, these approaches require hardware modification and cannot be applied to COTS platforms [33]–[35], [161].

Hassan *et al.* [33] presented an approach for scheduling memory access requests in mixed-time critical systems through the re-design of memory controllers (MCs) and an optimization framework to guarantee temporal and bandwidth requirements. It proposes a new implementation of the Time Division Multiplexing (TDM) schedule and implements a page policy scheme that dynamically switches between open and closed pages. Later it was extended in [161] to allow for rank interleaving and support dynamically interleaving across the different number of banks.

A new memory controller architecture capable of efficiently supporting variable transaction sizes was introduced in 2016 by Li *et al.* [34]. The front-end uses a TDM arbiter with a new work-conserving policy, and the back-end uses a dynamic command scheduling algorithm. Compared to existing memory controller architectures such as the one based on First-Ready First-Come-First-Serve (FR-FCFS) policy and the ROC [162], the proposal adds additional components such as the lookup table and the parameter queue, included in the back-end to support variable transaction sizes.

Another approach was adopted in Memory Inter-Arrival Time Traffic Shaping (MITTS) [35]. MITTS regulates the memory bandwidth of each application based on the frequency of each memory request inter-arrival time. Compared to the static allocation of memory bandwidth between applications, the proposed mechanism allows better isolation of memory bandwidth, which is vital for real-time systems.

Farshchi *et al.* [36] proposed a hardware unit: Bandwidth Regulation Unit (BRU), whose objective is to regulate per-core accesses to shared memory resources, limiting interference between cores without incurring a significant software overhead, unlike previous software-based proposals [10],

[88]. This approach improves memory bandwidth utilization and allows regulation in more precise time intervals.

#### 2) PHASED EXECUTION MODEL

Papers in this sub-category are based on controlling access to the memory bus to avoid or eliminate contentions using a phased execution model of each task and a suitable scheduling algorithm.

A global memory-centric scheduling approach was proposed in 2015 by Yao *et al.* [163]. Based on previous Predictable Execution Models (PREM) [164]), the set of tasks is modelled in two phases, a memory phase, and an execution phase. The central idea of this work is to provide isolation in access to main memory by dividing the execution of tasks into phases. The number of cores that can access memory simultaneously is limited to avoid memory bus bandwidth saturation. A memory-centric global scheduler sets the maximum number of cores that can access simultaneously and assigns higher priority to memory phases over execution phases to prevent interference.

Following a model similar to PREM, Rivas *et al.* [37] proposed implementation of Memory Centric Scheduling (MCS) in a Real-time Operating System (HIPPEROS). In the scheduling framework, tasks follow a similar model to PREM and use a memory phases scheduler that dynamically decides which memory phase to execute. It is considered partitioned scheduling with fixed task priorities (FTP) and a modified PREM task model that defines two types of memory phases in addition to the execution phases. To avoid interference, assigning a shared cache partition to each task is also applied.

In 2018, Pagetti *et al.* [165] presented a framework to guarantee time-predictability in the software development process. To eliminate bus interference, they use the Acquisition Execution Restitution (AER) execution model and to simplify the calculation of the upper limits of WCET, they use a non-preemptive partitioned schedule.

In 2021, Gifford *et al.* [166] presented other strategies called Dynamic Allocation (DNA) and deadline-aware DNA (DADNA) for allocating memory bandwidth resources and cache memory. The proposed algorithms allocate resources dynamically depending on the execution phase of a task. The DNA algorithm focuses on allocating resources to the tasks whose execution rate will increase the most, and the DADNA algorithm focuses on allocating resources to the tasks that need them to meet their deadlines. Knowing the set of tasks to execute before starting the system is necessary. From the execution profiles obtained, the identification of phases and resource requirements is carried out.

#### 3) OFFLINE SCHEDULING

Papers in this sub-category focus on regulating access to the shared memory bus through a table containing the allocations for use at run-time. This information is obtained from offline static scheduling techniques.

An algorithm for automatic allocation and scheduling in multiprocessor systems was proposed in 2015 by Carle *et al.* [167]. An offline scheduling algorithm is implemented that calculates a scheduling table. Each task is associated with a processor in the scheduling table, and a set of time intervals are assigned for its execution. A static time-division multiplexing (TDM) mechanism is used, and the execution of the task is limited to reserved time intervals to eliminate the interference between the partitions in a processor.

Perret *et al.* [168] proposed an execution model that consists of a set of rules to guarantee temporary isolation between applications. Taking into account the interference caused by the shared resources between partitions, four fundamental rules are defined: each local SRAM bank and each core can only be reserved for one partition, strictly periodic time intervals control NoC accesses established offline, memory buffers used by DMAs are defined offline, and a bank of the external DDRx-SDRAM can be only shared if the partitions are not accessed simultaneously. The approach allows temporal isolation of hard real-time applications on many-core processors.

#### 4) HARDWARE ISOLATION

Papers in this sub-category focus on the use of arbitration policies such as TDMA [169] or Round Robin to guarantee hardware isolation in shared bus access.

Dasari *et al.* [170] presented a framework for memory access contention analysis. The approach considers different bus arbiters and calculates the maximum interference caused by the shared memory bus. Then, through an arbitration policy, which depends on the platform considered, the dispute of the cores that try to access the shared bus is resolved.

A bus-aware ILP formulation for estimating WCET, including the dynamic prediction of bus offsets and effects on execution times, was presented in 2017 by Oehlert *et al.* [171]. This approach focuses on access lock times to a shared memory when using the TDMA arbitration policy on the shared memory bus.

In 2019, Park *et al.* [38] proposed an approach to reduce inter-core interference by applying the Time Division Multiple Access (TDMA) [169] and Acquisition-Execution-Restitution (AER) [172] execution models without modifying the Operating System. For the application of the execution models, they use pseudo-partitions. A pseudo-partition ensures that accesses to the shared resources of each partition do not interfere with each other. Subsequently, Park *et al.* [39] proposed a multi-TDMA model. The proposed model allows the simultaneous execution of two cores, taking into account that the memory bus of the target board can process two requests simultaneously. The use of multi-TDMA shows a higher utilization than the TDM model and a shorter execution time than the AER model, which is obtained by not requiring phase control.

Hebbache *et al.* [173] presented a dynamic arbitration scheme based on the Time Division Multiplexing (TDM) arbitration policy. This work proposed a dynamic arbitration scheme where the arbiter can change the order in which

requests are handled based on the slack time of each pending request in the system. The arbiter prioritizes critical tasks over non-critical guaranteeing deadlines of critical requests. In 2018, a new approach [174], instead of TDM slot-level arbitration schemes, operates at the granularity of clock cycles.

#### 5) PROPOSAL FOCUSED ON ACHIEVING A BETTER PERFORMANCE

Papers in this sub-category employ interference reduction techniques to achieve better performance on the multicore platform without considering the predictability issues.

*Memory Bandwidth Regulator*: Combined with virtualization techniques for spatial partitioning on Last Level Cache (LLC), in 2018, Modica *et al.* [31] proposed implementation for temporary isolation in the DRAM controller using a memory bandwidth reservation mechanism combined with the scheduling logic of the hypervisor.

*Other approaches*: In 2020, proposals were implemented focused on evaluating the contentions in the memory bus. The method proposed by Rashid *et al.* [40] evaluates the impact of cache persistence on memory bus contention, considering that the higher the reuse of cache blocks, the lower the number of accesses to main memory and, consequently, less will be the contention on the memory bus. On the other hand, Restuccia *et al.* [41] focus on bounding the worst-case bus contention experienced by the hardware accelerators deployed in the FPGA fabric, considering that the main source of the unpredictability of hardware accelerators is access to memory through the bus.

#### 6) SUMMARY

The techniques proposed to reduce interference in the shared memory bus focus on controlling access to the bus through memory bandwidth regulators, mechanisms that use models of execution in phases, offline scheduling, or arbitration policies.

Memory bandwidth regulation approaches have been implemented at the software [32], [157], [158] and hardware level. Memguard [157] ensures temporal isolation by reserving bandwidth and further improves performance by exploiting the best effort bandwidth after satisfying each core's reserved bandwidth. However, MemGuard does incur some implementation overhead. CaM [32] integrates the CAT cache partitioning mechanism and MemGuard bandwidth regulation mechanisms. BWLOCK [158] reduces memory bandwidth contention and improves the performance of soft real-time applications with a controllable performance impact on non-real-time tasks. Its limitations lie in overhead in the millisecond range due to interrupt handling and not offering protection if all tasks executed simultaneously are in real-time. The method proposed by Agrawal *et al.* [159] allows dynamic memory bandwidth isolation without application source-code modifications. Its implementation requires creating offline schedule tables.

Hardware-based techniques for memory bandwidth regulation employ redesigns of the memory controller to allocate a portion of memory bandwidth to each core [33], [34], [161]. Proposals such as [35], [36] use other hardware mechanisms that allow fine-grain bandwidth allocation with their respective hardware cost. In addition to the independent requirements of each proposal, the use of bandwidth regulators in many cases leads to not fully utilizing resources.

Proposals focused on a phased execution model provide temporary isolation by using a scheduling algorithm that allows a limited number of memory phases simultaneously. The division of tasks into phases and the global scheduler implemented by Yao *et al.* [163] avoid contention on the memory bus and main memory. The Memory Centric Scheduling implemented by Rivas *et al.* [37] is based on the PREM model and employs an asymmetric master-slave architecture that allows the slave cores can execute tasks with a baremetal level of overheads. Pagetti *et al.* [165] eliminate bus interference thanks to the AER execution model. The authors present a framework for producing time-predictable code for an ARM-based multicore platform. The proposal requires computing offline mapping and scheduling. Finally, the allocation algorithms proposed by Gifford *et al.* [166] can improve the performance of a system without adding more resources, but they require knowing the set of tasks to execute before starting the system. Consideration should be given to the code overheads added by the memory phases and the delays that a memory phase experiences until the scheduler grants it access to the bus in implementing approaches focused on phased execution models.

Proposals focused on offline scheduling employ scheduling tables [167] or allocation rules [168] obtained through offline static scheduling techniques. The use of approaches focused on offline scheduling implies the need to recalculate the completely assignments for each modification in the set of tasks.

Other proposals employ arbitration policies to guarantee hardware isolation in access to the shared memory bus. Hardware isolation techniques require the use of specific hardware designs. The framework presented by Dasari *et al.* [170] allows calculating the maximum interference caused by the shared memory bus considering different bus arbiters. Park *et al.* [38] use TDMA and AER models. Your implementation of the TDMA model requires additional partitions on each core. The execution model can reduce interference without modifying the code. On the other hand, memory space is wasted to configure such additional partitions. Compared to the TDMA model, running the AER model uses fewer memory areas. Later, Park *et al.* [39] proposed a multi-TDMA model. The evaluation shows that the multi-TDMA model has higher utilization than the TDMA model and a lower execution time than the AER model.

Other proposals [173], [174] use the time-division multiplexing (TDM) arbitration policy. TDM ensures predictable behaviour by guaranteeing exclusive access to a shared memory; however, it causes memory underutilization. Instead

**TABLE 6. Comparison Table of Memory bus Interference Reduction Techniques.**

| Approaches | Technique | Description | Hard/Soft | HRT/SRT/AVG |
|---|---|---|---|---|
| [88], [157] | Bandwidth Regulator | Memory bandwidth allocation | Software | HRT/SRT |
| [158] | Bandwidth Regulator | Prioritize soft real-time applications | Software | SRT |
| [32] | Bandwidth Regulator | Focuses on partitioned scheduling | Software | SRT |
| [160] | Bandwidth Regulator | Budget-based memory bandwidth regulation | Software | SRT |
| [159] | Bandwidth Regulator | Dynamic memory bandwidth allocation | None | HRT |
| [33], [161] | Bandwidth Regulator | Re-design of memory controllers | Hardware | HRT/SRT |
| [34] | Bandwidth Regulator | New memory controller architecture | Hardware | SRT |
| [35] | Bandwidth Regulator | Memory Inter-Arrival Time Traffic Shaping | Hardware | SRT |
| [36] | Bandwidth Regulator | Bandwidth Regulation Unit (BRU) | Hardware | SRT |
| [168] | Offline Scheduling | Execution model composed of rules | Soft/Hard | HRT |
| [167] | Offline Scheduling | Scheduling table | Software | HRT |
| [37], [163] | Phased Execution Model | Predictable Execution Model | Software | SRT |
| [165] | Phased Execution Model | Acquisition Execution Restitution | Software | HRT |
| [166] | Phased Execution Model | Execution profile and resource allocation | Hard/Soft | SRT |
| [170] | Hardware isolation | Arbitration policy | Hardware | SRT |
| [38], [171] | Hardware isolation | TDMA arbitration policy | Hardware | HRT |
| [173], [174] | Hardware isolation | Time Division Multiplexing (TDM) | Hardware | SRT |
| [39] | Hardware isolation | Multi-TDMA model | Hardware | HRT |
| [31] | Bandwidth Regulator | Memory bandwidth reservation mechanism | Software | AVG |
| [40] | Other Approaches | Persistence-aware bus contention analysis | None | AVG |
| [41] | Other Approaches | Bounding the worst-case bus contention | Hardware | AVG |

of arbitrating at the TDM slot level, operating at the granularity of clock cycles improves memory utilization.

Table 6 shows a summary of the main approaches to reduce interference on the shared memory bus.

## IV. SCHEDULABILITY ANALYSIS TECHNIQUES

Schedulability analysis or tests have been developed to determine if a set of tasks is schedulable or not. Schedulability analyzes make it possible to guarantee that the time requirements of the tasks are met according to the scheduling algorithm implemented. Verifying the time limits of the system tasks allows improving the efficiency of designing and implementing real-time systems.

Obtaining the WCET of each task in real-time is the basis for the schedulability analysis. Different approaches have been studied for scheduling tasks in real-time multicore systems: partitioned [54], [175], semi-partitioned [176]–[178] and global [179]–[181] scheduling. In the partitioned scheduling approach, tasks are assigned between available processors, and each task can only be executed on the processor where it was assigned. The global scheduling approach allows a task to migrate from one processor to another, and the semi-partitioned scheduling approach decides whether a task is divided into subtasks and then assigns each subtask to a processor [182].

Early research on partitioning and global multiprocessor scheduling employs the Earliest Deadline First (EDF) or fixed-priority scheduling using Rate Monotonic (RM) priority assignment techniques. Pereira and Mejia [183] carry out a study of the real-time scheduling techniques: RM and EDF and present the schedulability conditions for RM and EDF used for multiprocessor scheduling.

According to Davis and Burns [182], in preemptive uniprocessor scheduling using fixed-task priorities, approaches like the priority assignment techniques, that includes rate monotonic (RM) and deadline monotonic (DM), and the EDF technique used for sporadic tasks sets had a strong influence on research into multiprocessor scheduling. Let us consider that all task sets are periodic, preemptable, and independent of each other. Then, using online scheduling with fixed priorities where each task has a priority assigned using RM or DM can benefit from a schedulability test based on CPU utilization rate. The base condition compares the utilization of the task set with a utilization bound value to determine if the task set is schedulable. For arbitrary task sets using fixed priorities, including RM, DM, and others, the response time analysis is a valid option for testing the schedulability of the task set. These tests are based on the critical instant concept. The critical instant occurs when the task is requested simultaneously with all the other higher priority tasks. The schedulability condition establishes that a feasible schedule is obtained when the task response times (including the task computational time and the interference times from higher priority tasks) at their critical instants are less than their respective deadlines [184]–[186]. The investigations of preemptive scheduling have been sufficiently matured to guarantee time restrictions in real-time systems. However, studies of non-preemptive scheduling have not matured for real-time tasks subject to timing constraints. Considering that non-preemptive scheduling is necessary for tasks that inherently disallow any preemption or have large preemption/migration overheads, a significant number of proposals have appeared in recent years to resolve this issue [187]–[191].

A selection of proposals for scheduling tasks in a real-time multicore system carried out in the 2015-2020 period is discussed below.

Fixed-priority partitioned multiprocessor scheduling for sporadic real-time systems was studied in 2016 by Chen [175]. They show that when considering task systems with arbitrary deadlines, a greedy mapping strategy has a speedup factor that is valid for polynomial-time

schedulability tests and exponential-time (exact) schedulability tests.

Among the semi-partitioned scheduling approaches, is the proposal made by Brandenburg and Gül [176]. Brandenburg and Gül [176] establish that an adequate schedulability can be achieved with a simpler approach that combines the techniques of Reservation-based EDF Scheduling, semi-partitioning, period transformation, and appropriate task-placement strategies. However, the effects of interference due to shared resources are not considered.

Although semi-partitioned scheduling allows near-optimal scheduling performance and is simpler to implement compared to global scheduling; generally, semi-partitioned schedulers leverage an offline phase for allocating the tasks, which requires a-priori knowledge of the workload. Casini *et al.* [177] proposed a semi-partitioned approach that implements the C = D splitting algorithm [178] with linear-time approximate methods that allow it to be used online without incurring high overhead.

In global scheduling, a task can migrate from one core to another. However, in multicore systems, accurate analysis of the global schedulability of a task would require considering all possible patterns, which is impractical.

Sun and Di Natale [192], based on the previous work [193], presented an analysis of pessimism in multicore global schedulability. Yalcinkaya *et al.* [179] proposed an exact schedulability test for non-preemptive self-suspending real-time tasks under a global fixed-priority scheduling policy. Zhao *et al.* [180] and Lee *et al.* [181] proposed simplified schedulability models considering only tasks with uncertain worst-case execution time, respectively. Lee and Choi [194] generalized the problem by presenting a constraint-based schedulability analysis in real-time multiprocessor systems, considering a global approach. Power was included in the schedulability analysis in Lee *et al.* [195] by considering transiently powered processors in the schedule.

Bael *et al.* [196] proposed an improvement for the schedulability analysis of the contention-free(CF) policy for real-time systems. The proposal is based on the principle of the CF policy: to improve schedulability of the existing scheduling algorithms by demoting a job's priority when its schedulability is guaranteed during its execution.

In Chen *et al.* [187] the non-preemptive and strictly periodic scheduling upon a multicore platform is studied. From the schedulability analysis, the conditions are presented to determine if a task can be scheduled without modifying the offsets of the rest of the tasks that are executed, and a formula is made to select all the valid start time offsets of the task. Subsequently, a task assignment algorithm is proposed to obtain the number of cores required by a periodic task set.

Hardware heterogeneity was considered in Wanget *et al.* [197]. They presented a schedulability analysis based on Timed Automata and the symbolic verification method for heterogeneous multicore real-time systems. Moulik *et al.* [198] proposed a low-overhead algorithm for the scheduling of real-time periodic tasks. The algorithm is based on DPFair [199], an optimal real-time homogeneous multiprocessor scheduler. Moulik *et al.* [198] modified the task allocation and scheduling to satisfy the constraints related to the heterogeneous system. The results obtained in comparison with the implementation of the Sort and Assign (SA) algorithm [200] improve the rate of accepted task sets concerning the total number of submitted task sets. Another scheduling technique for heterogeneous multicore systems was proposed in 2018 by Moulik [201]. The proposal extends the optimal Hetero-Fair algorithm [202] to be applied on generic heterogeneous platforms consisting of more than two processor types. Bertout *et al.* [203] proposed and proved a novel and efficient algorithm to build the second step of the feasibility analysis of preemptive real-time systems upon heterogeneous multiprocessor platforms presented by Baruah [204]. Nair *et al.* [205] presented a standby-sparing based fault-tolerant energy-aware scheduling strategy for heterogeneous systems. The scheduling strategy can handle a specified number of transient faults per frame and minimize overall energy consumption. In 2019, Zhang *et al.* [206] implemented a linear programming algorithm and a dynamic programming algorithm for real-time task scheduling under a heterogeneous platform with task duplication. Subsequently, Devaraj [207] demonstrated that the linear programming algorithm fails to produce valid execution schedules because the proposed problem formulation does not correctly capture the execution requirements of real-time tasks. Devaraj [207] presented the necessary modifications for optimal execution of the algorithm proposed by Zhang *et al.* [206].

Other proposals consider the delays in the execution of a task due to the arbitration policies used and the contention due to the simultaneous use of shared resources and integrate these delays into schedulability analysis.

### A. INTEGRATING INTERFERENCE EFFECTS INTO SCHEDULABILITY ANALYSIS

In recent years (2015-2020), schedulability analysis techniques in real-time multicore systems have integrated interference effects into schedulability analysis. The investigations carried out are grouped into different categories considering the shared resource they consider: memory bus, main memory, cache memory, or the integration of the effect of multiple shared resources.

#### 1) MAIN MEMORY INTERFERENCE

The proposals in this subsection mainly focus on analyzing contention in shared main memory and their integration in the schedulability analysis.

Kim *et al.* [22], [71] proposed an analysis that focuses on limiting the worst response time of a task in the presence of memory interference. The study considers delays due to row conflicts, DRAM access scheduling, the FR-FCFS policy on the DRAM controller, interbank and intrabank interference. The model implements dedicated and shared bank partitions, splits DRAM banks, and implements a task

allocation algorithm that allocates intensive memory tasks on the same core with dedicated DRAM banks.

An approach to schedule resource access sporadic tasks in an architecture that has memory banks of limited capacity was proposed in 2017 by Cheng *et al.* [208]. They adapt the symmetric analysis presented by Huang *et al.* [209] to systems with multiple memory banks. It suggests limiting data partitioning between banks for a task, taking into account that the lower the number of data partitions between different banks, the less memory interference will be on other tasks.

A detailed analysis of memory contention experienced by parallel tasks executed on a multicore system was presented by Casini *et al.* [72]. The proposal uses a holistic approach to limit memory contention by formulating an optimization problem that maximizes the overall interference generated by all types of inter and intra-bank interfering requests. Another approach to memory interference analysis is presented by Dinechin *et al.* [73], where an algorithm was proposed to optimize the complexity of the algorithm previously proposed by Rihani *et al.* [210], based on calculating the timing of a task graph to estimate the impact of interference on memory accesses.

Durand *et al.* [211] proposed a measurement-based approach and statistical analysis of task execution times focused on characterizing the effects of shared memory interference. In 2020, Hassan *et al.* [212] proposed an approach for main memory contention analysis in COTS MPSoCs to provide a safe limit of the delays that these memory contentions cause to critical tasks in a Mixed-Criticality Systems (MCS).

Reder *et al.* [213] proposed a generic multicore platform model and an ILP-based memory allocation scheme to cope with the problem of interference-aware memory allocation. Both are designed to optimize allocation by mapping the program data, taking into account interference costs.

#### 2) BUS INTERFERENCE

The proposals in this subsection mainly focus on analyzing the contention in the shared memory bus and integrating these in the schedulability analysis.

Schedulability analysis for memory bandwidth regulated multicore real-time systems was presented in Yao *et al.* [214]. The fundamental advantage of the proposed research is that it does not require any knowledge about the tasks that are executed in the other cores.

Maia *et al.* [43] performed a schedulability test focused on the global fixed-priority scheduling of the 3-phase task model by modelling interference on the shared bus. The 3-phase task model divides tasks into three successive phases: Acquisition, Execution, and Restitution. Tasks never have access to the bus during their execution phase. The schedulability analysis is performed from the perspective of the bus, considering that only two memory phases can be executed simultaneously.

In 2016, Usui *et al.* [215] proposed a Deadline-Aware memory Scheduler for Heterogeneous systems (DASH) that requires hardware support to monitor hardware accelerators (HWAs). The algorithm manages memory bandwidth between cores and HWAs through a scheduling policy that prioritizes HWA memory requests over memory-intensive CPU applications and establishes a combined scheduling policy for HWAs with a long deadline period and a short deadline period.

Rouxel *et al.* [44] presented two bus contention awareness scheduling strategies to effectively estimate contention delays while scheduling parallel applications on multicore architectures. The estimation of contention delays is determined from the knowledge of the structure of the modelled application as directed acyclic task graphs (DAGs). The proposed approach applies to multicore platforms where cores are connected to a round-robin bus.

#### 3) CACHE INTERFERENCE

The proposals in this subsection mainly focus on analyzing shared cache contention and integrating these into the schedulability analysis.

Sun and Lipari [45] addressed the schedulability analysis for a set of tasks scheduled by the Global Earliest Deadline First (GEDF) policy. It proposes a schedulability test that integrates the limited carry-in technique and Response Time Analysis (RTA) procedure in the analysis.

A global Preemptive Fixed-Priority scheduling with dynamic cache allocation, where shared cache interference is eliminated through a combination of cache partitioning and cache-aware scheduling, was proposed in Xu *et al.* [216]. As an alternative, Xiao *et al.* [217] presented a schedulability analysis of non-preemptive real-time scheduling, which, unlike previous work, addresses the shared cache problem without implementing any cache isolation techniques. Using an iterative algorithm obtains the upper limit of cache interference in executing a task and later integrates it in the schedulability analysis. In this aspect, Srinivasan *et al.* [46] presented an interesting paper trying to define empirical bounds of multicore cache interference for real-time schedulability analysis. Finally, interference is included in the schedulability analysis. Xiao *et al.* [47] extends the proposal taking into account in the schedulability analysis in addition to Fixed Priority (FP) algorithm, Earliest Deadline First (EDF) algorithm. The evaluations show that EDF is slightly better than FP in terms of sets of tasks that are considered schedulable.

In 2020, Xiao *et al.* [49] proposed a task partitioning algorithm called CITTA that takes into account the cache interference exhibited by a task obtained from an integer programming formulation. Also, in 2020, Sheikh and Pasha [50] proposed a dynamic cache-partition schedulability analysis for partitioned scheduling. The study uses the problem window approach to obtain the maximum response time for a task. First, the iterative analysis of Davis *et al.* [218] is proposed to determine the upper limit of the interference experienced by tasks executing on fixed-priority preemptive scheduling algorithms for single-core processors where a task can only suffer contentions due to the execution of

higher priority tasks running on the same core. Subsequently, Sheikh and Pasha [50] proposed to define the upper limit of interference, taking into account that cache partitions are a global resource. Therefore a task with a higher global priority can precede a task with a relatively lower priority that is executed on another core if there are not enough free cache partitions.

Nguyen *et al.* [219] took into account the effect of private caches on tasks' WCETs in task scheduling. The main objective is to obtain more reduced scheduling from cache reuse. Two methods are proposed, an Integer Linear Programming (ILP) formulation and a heuristic method based on list scheduling, both taking into account the variation of the WCET of the tasks as a function of the reuse of the cache according to the order of execution of the task. The proposal focuses on non-preemptive and partitioned scheduling. Task scheduling is generated offline, and its execution is triggered at predefined instants of time.

Other approaches, such as the one proposed by Sheik *et al.* [220], focus on modelling cache contention and using scheduling algorithms to minimize energy consumption.

#### 4) MULTIPLE RESOURCE INTERFERENCE

The proposals in this subsection mainly focus on analyzing the contention of several resources and the integration in the schedulability analysis.

The dependency of tasks on estimating shared resource contention was studied in Choi *et al.* [221]. Choi *et al.* [221] employ a non-preemptive and preemptive scheduling policy with a fixed priority. The proposed methodology uses a shared resource contention module and a WCRT analysis module. The shared resource contention module verifies if two tasks are entirely separated based on the tasks' earliest start and finish times and their restrictions. When one task is wholly separated from another, you do not need to consider the resource demands of the other task. Subsequently, clusters of tasks that do not interfere with each other are performed executed in a particular order on specific processing elements. The WCRT analysis module obtains the upper limit of demand for shared resources of each cluster in each processing element. The results obtained show an improvement in the schedulability due to a reduction in the demand for resources in a certain execution time window.

Huang *et al.* [209] proposed asymmetric analysis of the response time of task processing and access to shared resources, taking into account that a task runs in the core until it needs to access a shared resource and suspends its execution in the core. The scheduling capacity analysis improves the results obtained by Altmeyer *et al.* [222]. A partitioned multicore system is considered using fixed-priority preemptive scheduling. The analysis of the response time and schedulability considers the so-called execution intervals and suspension intervals of a task. The time slots in which the task suspends access to shared resources either occupied by a higher priority task or in which the task executes local calculations are called suspend intervals. When the task is running on or accessing a shared resource, it is called the execution interval. The proposed analysis strategy considers that the viability of a task is given if its deadline does not exceed the sum of the suspension intervals and the execution intervals.

A framework for schedulability and memory interference analysis of multicore preemptive real-time systems was proposed in Boudjadar and Nadjm-Tehrani [51]. The platform model consists of a multicore system where the cores share main memory and L2 cache. The cache coloring policy [150] and the FR-FCFS policy are used to arbitrate DRAM accesses and to handle simultaneous access requests to the L2 cache. In addition, the symbolic model checking technique [223] is used to perform the schedulability analysis of the system. In the case of non-schedulability, Boudjadar and Nadjm-Tehrani [51] propose the reassignment of tasks according to the utilization of the cores. In 2020, Guo *et al.* [52] proposed an intertask interference-aware scheduling framework on multiprocessors. The mechanism presents Intertask Interference Matrix (ITIM) to quantify intertask interference and guide task partitioning to cores. From the ITIM determined for a system, the cache-aware partitioned scheduling problem is configured as a mixed-integer linear programming (MILP) problem. The central idea is to find the best option to partition the tasks in terms of interference so that tasks with high cache interference executed in the same core are mapped to a different core, leading to an increase in resource use efficiency.

Chwa *et al.* [53] presented a Global EDF Schedulability Analysis for Parallel Tasks on Multicore Platforms. The analysis is based on the Directed Acyclic Graph (DAG) model, considering concepts such as critical interference and p-depth critical interference to capture the parallelism at the thread level more accurately.

Yang *et al.* [54] proposed a Resource-Oriented Partitioned (ROP) scheduling with a distributed resource sharing policy based on the concept of Distributed Priority Ceiling Protocol (DPCP). Tasks and resources are allocated statically among available processors. Each task can execute its non-critical sections only on the processor to which it is assigned, and all task requests for a given resource can only run on the processor to which that resource has been assigned. When a task requests a shared resource, the priority of the corresponding critical sections is increased to ensure that it is higher than any non-critical section of any task on the same processor. It is proposed to partition shared resources first and then assign tasks in decreasing order of priority to ensure task scheduling.

A schedulability test considering fixed-priority preemptive partitioned scheduling of constrained-deadline sporadic tasks was presented in Andersson *et al.* [224]. The proposed model describes how the tasks have co-runner-dependent execution times, the analysis is based on the execution times obtained through static analysis or measurements. Also, considering partitioned fixed-priority scheduling, Al-Bayati *et al.* [55]

focused on resource-aware partitioning approaches. It implements an ILP formulation for the efficient assignment of tasks to the core, the assignment of priorities to the tasks, and the selection of resource protection mechanisms. The tradeoffs among these mechanisms are analyzed to select the most efficient resource protection mechanisms (lock-based or wait-free).

Based on the global scheduling algorithm, Lei *et al.* [56] proposed a scheduling strategy for real-time parallel tasks. The proposed scheduling strategy establishes that the total number of cores that access memory simultaneously cannot be greater than the total number of memory banks. If the total number of tasks ready to access the memory exceeds this value, the lower priority tasks are blocked. Tasks that access memory have higher priority than those that do not, and tasks that use data in cache have higher priority over all other tasks.

Sinha *et al.* [57] presented a scheduling strategy integrated into mixed-criticality systems and based on the execution progress of a task. The proposal focuses on improving QoS for low-criticality tasks in these systems where all low-criticality tasks are stopped when a high-criticality task runs for more than its budget. The application evaluates the execution time of a critical task from an identified offline checkpoint. Based on the progress of the checkpoint, a higher budget is assigned to the critical task without degrading the performance of low-criticality tasks. In case the task does not finish within the expected time, the low criticality tasks are finally dropped.

In 2020, Schwäricke *et al.* [58] presented and performed the analysis of a memory-centric scheduler for deterministic memory management on COTS multiprocessor platforms without hardware support. The memory-centric scheduler is implemented in a hypervisor. The proposal uses the Predictable Execution Model to limit main memory interference and software cache partitioning based on cache coloring.

In 2021, Aceituno *et al.* [225] presented a hardware resource contention-aware scheduling of hard real-time multiprocessor systems. The proposal focuses on partitioned scheduling. The authors define a task model that considers interference delays due to the contention of shared hardware resources and proposes three task allocation algorithms (UDmin, UDmax, and Wmin allocator) to reduce this interference.

### 5) APPLICATION EXAMPLE

In this subsection, a scheduling algorithm for real-time multicore systems with its corresponding scheduling condition is applied. The example includes applying techniques to reduce interference due to the simultaneous use of shared resources. We consider a set of 4 real-time tasks t = t1,t2,t3,t4 on a set of 2 processors, P = P1,P2. All tasks are periodic, preemptable, and independent of each other. The parameters that define a task are the execution time Ci, the period Ti, and the deadline Di. The deadline of each task is considered equal to its period. Tasks are distributed among processors statically using the

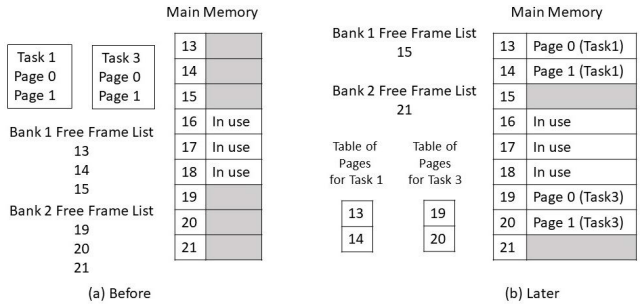


**FIGURE 9.** Example application of the PALLOC technique.

Rate Monotonic Next Fit (RMNF) heuristic algorithm [183]. In this algorithm, the tasks are initially ordered in increasing order according to the value of their periods. Then a task is allocated to the processor only if there is a feasible schedule for the task set after using the schedulability condition Increasing Period. Otherwise, the task is assigned to the next processor.

Considering T = 7; 12; 20; 25 ms and an estimate of C = 3; 3; 5; 7 ms for tasks t1, t2, t3, and t4, respectively, the schedulability condition is evaluated for the static assignment of the tasks to the processors. As a result of applying the condition, tasks t1 and t2 are assigned to processor P1, and tasks t3 and t4 are assigned to processor P2.

The execution time used to evaluate the schedulability condition is assumed to be the worst-case execution time for all tasks. Techniques to eliminate the interference between the tasks due to simultaneous access to shared resources are considered to guarantee the predictability of the response time

To eliminate interference due to shared main memory, PALLOC [65] can be used. PALLOC creates partitions by assigning free page frames to virtual pages. The allocated free page frames belong to a specific list of determined DRAM banks. Since banks are not shared under this scheme, cores cannot interfere with each other by closing rows opened by another core. It is considered that both task 1 (running on processor 1) and task 3 (running on processor 2) consist of 2 pages stored on disk. Using PALLOC, when it is time to load each task, the pages of each task are loaded into free frames from different memory banks. This method minimizes unpredictability by eliminating bank sharing among parallel executing applications. Figure 9 illustrates an example of the application of the technique for task 1 and task 3.

Another technique can be used to reduce interference due to shared cache memory. Page coloring is a software mechanism that allows partitioning the shared cache by guiding the allocation of physical pages. Each memory page has a fixed mapping to a physically contiguous group of cache lines. Figure 10 illustrates an example of the application of page coloring for task 1 and task 3.

A coordinated cache and bank coloring mechanism to avoid cache and bank interference simultaneously were implemented by Suzuki *et al.* [99].

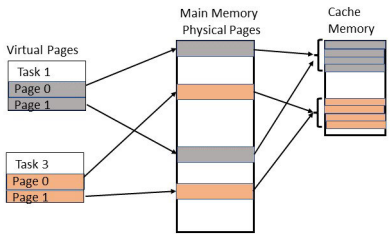


**FIGURE 10.** Example application of the page coloring technique.

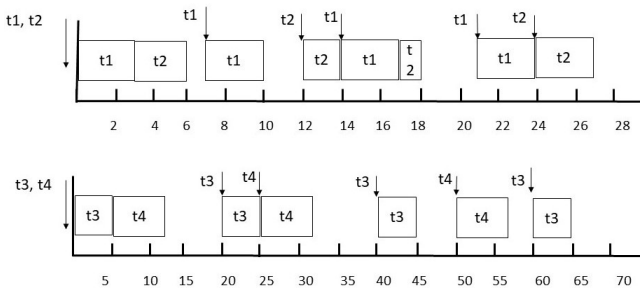


**FIGURE 11.** Task execution diagram.

Finally, meeting the schedulability condition ensures that the set of tasks on each processor will have a feasible schedule under the Rate Monotonic algorithm. The Rate Monotonic scheduler assigns the highest priority to the shortest period task. When a task is triggered, if another lower priority task is using the CPU, the lower priority task will go to the "ready" state, and the CPU will execute the higher priority task. The behaviour of the system is represented in Figure 11.

### 6) SUMMARY

Task scheduling methods in real-time systems include scheduling algorithms to determine the order of task access to system resources and analysis methods to calculate the temporal behaviour of the system. Schedulability analysis techniques are used to check whether the temporal requirements are guaranteed in all cases. Proposed approaches to scheduling tasks in real-time multicore systems are divided into partitioned, semi-partitioned, and global scheduling. The problem of static assignment of tasks when using partitioned scheduling can be formulated as a bin packing problem; there are also heuristic algorithms to provide an efficient solution to task partitioning between cores. Techniques focused on semi-partitioned scheduling generally employ an offline phase for task assignment, and approaches that focus on global schedulability rely on a set of conditions since it is impractical to consider all possible patterns.

In recent years (2015-2020), schedulability analysis techniques in real-time multicore systems have integrated interference effects into schedulability analysis. The proposed works consider a single shared resource such as the memory bus, main memory, cache memory, or the integration of the effect of multiple resources. Table 7 shows a comparison of the proposals that integrate the interference effects into schedulability analysis.

Generally, the approaches used for system analysis assume some kind of hardware or software isolation or model interference in a time window. Then, the upper limit of the resource interference is obtained and integrated into the schedulability analysis.

Existing approaches for interferences take into account:

- Only shared main memory employs hardware or software modifications to provide predictable system behaviour. To model interference, Kim *et al.* [22] assume that each core has a fully timing compositional architecture. The iterative response time test is extended by incorporating memory interference delay for response-time-based schedulability analysis. In addition to providing a tighter upper bound on the worst-case response time of a task in the presence of memory interference, the isolation techniques implemented reduce the amount of memory interference among tasks. Cheng *et al.* [208] take into account in the response time analysis a fixed-priority scheduling policy to preemptively schedule tasks assigned to a core and task partitioning across cores and memory banks. The two-phase memory partitioning algorithm implemented significantly outperformed the state-of-the-art algorithm proposed by Kim *et al.* [22] in terms of schedulability test.
  Reder *et al.* [213] rely on synchronization analysis techniques for interference modelling. Their results show that analyzing the synchronization structure of parallel programs allows adjusting the limits of interference costs. They also show a reduction of the interference when considering these costs in memory allocation. Other proposals focus on interference analysis and limiting memory contention [72], [73], [210]–[212]. Quantification of interference effects can be applied later in developing guidelines for more predictable real-time multicore systems.
- Only the interference on the shared memory bus feature a schedulability analysis for memory bandwidth regulated multicore real-time systems and a schedulability test by modelling the interference on the shared bus. The analysis proposed by Yao *et al.* [214] assumes regulated memory bandwidth and a private or partitioned LLC for each core. The study does not require information about the resource demands of tasks on other cores. The schedulability analysis proposed by Maia *et al.* [43] is performed from the perspective of the bus. The interference is modelled considering the total demand of the resource in a time window. Other proposals [44], [215] use precise estimates of contention delays to define scheduling and mapping strategies.
- Only the interference in the shared cache memory integrates the effects of contention into the schedulability analysis. Some proposals use isolation techniques. For example, Xu *et al.* [216] consider cache overhead and derive an overhead-aware schedulability analysis combined with cache partitioning techniques.

**TABLE 7. Comparative Table of the Proposals That Integrate the Interference Effects Into Schedulability Analysis.**

| Approaches | Shared resource | Technique | Description | HRT/SRT |
|---|---|---|---|---|
| [208] | Main Memory | Fixed priority | Isolation technique | SRT |
| [22], [71] | Main Memory | Partitioned fixed-priority | Isolation technique | SRT |
| [72] | Main Memory | Partitioned non-preemptive scheduling | Memory Contention Analysis | SRT |
| [73], [210]–[212] | Main Memory | Algorithm optimization/Statistical analysis | Memory Contention Analysis | HRT/SRT |
| [213] | Main Memory | ILP formulation | Interference-Aware Allocation | HRT/SRT |
| [214] | Memory Bus | Partition-based | Isolation technique | HRT/SRT |
| [43] | Memory Bus | Global fixed-priority | Problem window | SRT |
| [215] | Memory Bus | Deadline-Aware memory Scheduler | Memory scheduling | SRT |
| [44] | Memory Bus | Partitioned, time-triggered and non-preemptive | Estimate contentions | SRT |
| [45] | Cache Memory | Global Earliest Deadline First | Problem window | SRT |
| [216] | Cache Memory | Global Preemptive Fixed-Priority | Isolation technique | SRT |
| [217] | Cache Memory | Non-preemptive global scheduling | Problem window | HRT |
| [47] | Cache Memory | Non-preemptive global scheduling | Problem window | HRT |
| [49] | Cache Memory | Partitioned scheduling | Isolation technique | SRT |
| [50] | Cache Memory | Partitioned scheduling | Problem window/Isolation | SRT |
| [46] | Cache Memory | Schedulability Analysis | Empirical Bounds | SRT |
| [219] | Cache Memory | Non-preemptive partitioned | Reuse of the cache | SRT |
| [220] | Cache Memory | Energy-efficient Scheduling | Energy minimization | SRT |
| [221] | Multiple resource | Fixed priority | Problem window | SRT |
| [209] | Multiple resource | Fixed priority preemptive | Symmetric analysis | SRT |
| [51] | Multiple resource | Symbolic model checking | Isolation technique | SRT |
| [54] | Multiple resource | Resource-Oriented Partitioned | Task partitioning | SRT |
| [224] | Multiple resource | Fixed-priority preemptive partitioned | Static analysis | HRT |
| [55] | Multiple resource | Partitioned fixed-priority | Resource-aware partitioning | SRT |
| [52] | Multiple resource | Partitioned scheduling | Task partitioning | SRT |
| [56] | Multiple resource | Global scheduling | Scheduling parallel tasks | SRT |
| [222] | Multiple resource | Fixed-priority pre-emptive | Timing verification | SRT |
| [53] | Multiple resource | Global Analysis | DAG model | HRT |
| [57] | Multiple resource | Progress-Aware Scheduling | Predicting the execution time | SRT |
| [58] | Multiple resource | fixed-priority scheduling | Memory-centric scheduler | SRT |
| [225] | Multiple resource | Partitioned scheduling | Task allocation algorithms | HRT |

Proposals [45], [47], [49], [50], [217] calculate an upper bound on the cache interference exhibited by a task within a given execution window. A fixed-point iteration derives the upper bound. Finally, interference is included in the schedulability analysis. The analysis is better by integrating task partitioning techniques or requesting pattern information.

Other proposals consider the context-sensitive WCETs per task in the scheduling or minimise energy consumption.

- The contentions of multiple resources, as well as previous works, focused on the interference of a specific resource employ isolation techniques, model the interference exhibited by a task within a given execution window as well as schedulability analysis taking into account the intervals of execution and suspension of a task. The proposals include task partitioning techniques taking into account intertask interference, scheduling strategies for parallel real-time tasks, task allocation algorithms, among others.

Modelling the interference in time windows considering the total demand for a resource leads to a less pessimistic value of the WCRT than the methods that employ the sum of the worst-case resource delays over a short time.

## V. CONCLUSION

Multicore processors show an increasing usage trend due to improvements in performance and efficiency compared to integrated systems with a single-core CPU and the possibility of running composite mixed-criticality application workloads. These multicore hardware platforms share various hardware resources. When two or more tasks are running in parallel, the interaction between the tasks on the shared hardware resources can lead to unforeseen and unpredictable delays. Real-time systems need to meet both functional requirements and time constraints. In the future (2020 onwards), real-time systems will involve multiple concurrent and complex applications on multicore hardware platforms. Such systems need to handle interference due to contentions on shared resources to ensure that tasks meet their time constraints. This survey provides an overview of the scientific literature on techniques for reducing interference on shared resources in real-time multicore systems focusing on the approaches proposed in the 2015-2020 period.

The bibliographic review indicates the main sources of interference as main memory, cache memory, and the memory bus. Real-time system requirements could be ensured by employing contention reduction strategies on shared resources. In the case of reducing interference due to contentions in shared main memory, these strategies include DRAM bank partitioning techniques, private DRAM banks, memory request scheduling, memory request throttling, and designing predictable DRAM controllers. In the case of reducing interference due to contentions in shared cache memory, these strategies include cache-partitioning techniques, cache-locking techniques, and designing predictable

cache coherence protocols. In the case of reducing interference due to contentions in the shared memory bus, these strategies include memory bandwidth regulators, mechanisms that use execution models in phases, offline scheduling, or arbitration policies.

The bibliographic review also shows a greater incidence in the proposals that integrate the interference into schedulability analysis. The integration of the interference effects into the schedulability analysis has the advantage of a more precise analysis when considering, for example, the set of co-running tasks.

In our opinion, the extensive literature review and discussion of the advantages and disadvantages of each approach presented in this survey are helpful for users designing and implementing systems that require some quality of service and guarantees of execution, which includes real-time systems in sectors such our aerospace, automotive, industry 4.0 and mobile applications that are time-bound due to restrictions of the system or the user response time, etc. The survey also presents proposals that use interference reduction techniques without considering the predictability issue. We assume that the works focused on achieving a better performance are also relevant for researchers and engineers dealing with interference in real-time systems.

To conclude, while we recognize that significant progress has been made to exploit the potentials of multicore systems, we consider that reducing interference in multicore architectures is still a challenge in real-time systems. This is especially true for a hard real-time system, where accurately computing the Worst-Case Execution Time of the tasks is critical to ensure predictability and increase the platforms' usage rate.

## REFERENCES


[1] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström, "The worst-case execution-time problem—Overview of methods and survey of tools," *ACM Trans. Embedded Comput. Syst.*, vol. 7, no. 3, pp. 1–53, Apr. 2008.

[2] J. Abella, D. Hardy, I. Puaut, E. Quinones, and F. J. Cazorla, "On the comparison of deterministic and probabilistic WCET estimation techniques," in *Proc. 26th Euromicro Conf. Real-Time Syst.*, Jul. 2014, pp. 266–275.

[3] K. Compton and S. Hauck, "Reconfigurable computing: A survey of systems and software," *ACM Comput. Surv.*, vol. 34, no. 2, pp. 171–210, 2002.

[4] M. Daněk, L. Kafka, L. Kohout, J. Sỳkora, and R. Bartosiński, "The LEON3 processor," in *UTLEON3: Exploring Fine-Grain Multi-Threading FPGAs*. Berlin, Germany: Springer, 2013, pp. 9–14.

[5] M. A. Sánchez-Puebla and J. Carretero, "A new approach for distributed computing in avionics systems," *ISICT*, vol. 3, pp. 579–584, Sep. 2003.

[6] T. Ungerer, F. Cazorla, P. Sainrat, G. Bernat, Z. Petrov, C. Rochange, E. Quinones, M. Gerdes, M. Paolieri, J. Wolf, H. Casse, S. Uhrig, I. Guliashvili, M. Houston, F. Kluge, S. Metzlaff, and J. Mische, "Merasa: Multicore execution of hard real-time applications supporting analyzability," *IEEE Micro*, vol. 30, no. 5, pp. 66–75, Sep. 2010.

[7] D. Dasari, B. Akesson, V. Nelis, M. A. Awan, and S. M. Petters, "Identifying the sources of unpredictability in COTS-based multicore systems," in *Proc. 8th IEEE Int. Symp. Ind. Embedded Syst. (SIES)*, Jun. 2013, pp. 39–48.

[8] A. Löfwenmark and S. Nadjm-Tehrani, "Understanding shared memory bank access interference in multi-core avionics," in *Proc. 16th Int. Workshop Worst-Case Execution Time Anal. (WCET)*. 2016, pp. 1–11.

[9] K. Nagalakshmi and N. Gomathi, "The impact of interference due to resource contention in multicore platform for safety-critical avionics systems," *Int. J. Res. Eng. Appl. Manage.*, vol. 2, no. 8, pp. 39–48, 2016.

[10] J. Nowotsch, M. Paulitsch, D. Buhler, H. Theiling, S. Wegener, and M. Schmidt, "Multi-core interference-sensitive WCET analysis leveraging runtime resource capacity enforcement," in *Proc. 26th Euromicro Conf. Real-Time Syst.*, Jul. 2014, pp. 109–118.

[11] C.-G. Lee, H. 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," *IEEE Trans. Comput.*, vol. 47, no. 6, pp. 700–713, Jun. 1998.

[12] T. Blaß, S. Hahn, and J. Reineke, "Write-back caches in WCET analysis," in *Proc. 29th Euromicro Conf. Real-Time Syst. (ECRTS)*, 2017, pp. 1–33.

[13] P. Benedicte, C. Hernandez, J. Abella, and F. J. Cazorla, "HWP: Hardware support to reconcile cache energy, complexity, performance and WCET estimates in multicore real-time systems," in *Proc. 30th Euromicro Conf. Real-Time Syst. (ECRTS)*, 2018, pp. 1–22.

[14] R. Banakar, S. Steinke, B.-S. Lee, M. Balakrishnan, and P. Marwedel, "Scratchpad memory: Design alternative for cache on-chip memory in embedded systems," in *Proc. 10th Int. Symp. Hardw./Softw. Codesign (CODES)*, 2002, pp. 73–78.

[15] L. Alvarez, L. Vilanova, M. Moreto, M. Casas, M. Gonzàlez, X. Martorell, N. Navarro, E. Ayguadé, and M. Valero, "Coherence protocol for transparent management of scratchpad memories in shared memory manycore architectures," in *Proc. 42nd Annu. Int. Symp. Comput. Archit.*, Jun. 2015, pp. 720–732.

[16] J. Matějka, B. Forsberg, M. Sojka, Z. Hanzálek, L. Benini, and A. Marongiu, "Combining PREM compilation and ILP scheduling for high-performance and predictable MPSoC execution," in *Proc. 9th Int. Workshop Program. Models Appl. Multicores Manycores*, Feb. 2018, pp. 11–20.

[17] G. Gracioli, A. Alhammad, R. Mancuso, A. A. Fröhlich, and R. Pellizzoni, "A survey on cache management mechanisms for real-time embedded systems," *ACM Comput. Surveys*, vol. 48, no. 2, pp. 1–36, Nov. 2015.

[18] R. Tabish, R. Mancuso, S. Wasly, A. Alhammad, S. S. Phatak, R. Pellizzoni, and M. Caccamo, "A real-time scratchpad-centric OS for multi-core embedded systems," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2016, pp. 1–11.

[19] R. Tabish, R. Mancuso, S. Wasly, R. Pellizzoni, and M. Caccamo, "A real-time scratchpad-centric OS with predictable inter/intra-core communication for multi-core embedded systems," *Real-Time Syst.*, vol. 55, no. 4, pp. 850–888, Oct. 2019.

[20] I. Puaut and C. Pais, "Scratchpad memories vs locked caches in hard real-time systems: A quantitative comparison," in *Proc. Design, Autom. Test Eur. Conf. Exhib.*, Apr. 2007, pp. 1–6.

[21] M. Dupont De Dinechin, M. Schuh, M. Moy, and C. Maiza, "Scaling up the memory interference analysis for hard real-time many-core systems," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2020, pp. 330–333.

[22] H. Kim, D. de Niz, B. Andersson, M. Klein, O. Mutlu, and R. Rajkumar, "Bounding and reducing memory interference in COTS-based multi-core systems," *Real-Time Syst.*, vol. 52, no. 3, pp. 356–395, May 2016.

[23] O. Kotaba, J. Nowotsch, M. Paulitsch, S. M. Petters, and H. Theiling, "Multicore in real-time systems–temporal isolation challenges due to shared resources," in *Proc. 16th Design, Autom. Test Eur. Conf. Exhib.*, 2013, pp. 1–8.

[24] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens, "Memory access scheduling," *ACM SIGARCH Comput. Archit. News. ACM*, vol. 28, no. 2, pp. 128–138, 2000.

[25] S. H. VanderLeest, J. Millwood, and C. Guikema, "A framework for analyzing shared resource interference in a multicore system," in *Proc. IEEE/AIAA 37th Digit. Avionics Syst. Conf. (DASC)*, Sep. 2018, pp. 1–10.

[26] M. Hassan and R. Pellizzoni, "Bounding DRAM interference in COTS heterogeneous MPSoCs for mixed criticality systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2323–2336, Nov. 2018.

[27] S. Wegener, "Towards multicore WCET analysis," in *Proc. 17th Int. Workshop Worst-Case Execution Time Anal. (WCET)*, 2017, pp. 85–132.

[28] J. Bin, S. Girbal, D. G. Pérez, A. Grasset, and A. Merigot, "Studying co-running avionic real-time applications on multi-core COTS architectures," in *Proc. Embedded Real Time Softw. Syst. (ERTS)*, 2014, pp. 1–10.

[29] D. Iorga, T. Sorensen, J. Wickerson, and A. F. Donaldson, "Slow and steady: Measuring and tuning multicore interference," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2020, pp. 200–212.

[30] C. Maiza, H. Rihani, J. M. Rivas, J. Goossens, S. Altmeyer, and R. I. Davis, "A survey of timing verification techniques for multi-core real-time systems," *ACM Comput. Surveys*, vol. 52, no. 3, pp. 1–38, May 2020.

[31] P. Modica, A. Biondi, G. Buttazzo, and A. Patel, "Supporting temporal and spatial isolation in a hypervisor for ARM multicore platforms," in *Proc. IEEE Int. Conf. Ind. Technol. (ICIT)*, Feb. 2018, pp. 1651–1657.

[32] M. Xu, R. Gifford, and L. T. X. Phan, "Holistic multi-resource allocation for multicore real-time virtualization," in *Proc. 56th Annu. Design Autom. Conf.*, Jun. 2019, pp. 1–6.

[33] M. Hassan, H. Patel, and R. Pellizzoni, "A framework for scheduling DRAM memory accesses for multi-core mixed-time critical systems," in *Proc. 21st IEEE Real-Time Embedded Technol. Appl. Symp.*, Apr. 2015, pp. 307–316.

[34] Y. Li, B. Akesson, and K. Goossens, "Architecture and analysis of a dynamically-scheduled real-time memory controller," *Real-Time Syst.*, vol. 52, no. 5, pp. 675–729, Sep. 2016.

[35] Y. Zhou and D. Wentzlaff, "Mitts: Memory inter-arrival time traffic shaping," *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 532–544, 2016.

[36] F. Farshchi, Q. Huang, and H. Yun, "BRU: Bandwidth regulation unit for real-time multicore processors," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2020, pp. 364–375.

[37] J. M. Rivas, J. Goossens, X. Poczekajlo, and A. Paolillo, "Implementation of memory centric scheduling for COTS multi-core real-time systems," in *Proc. 31st Euromicro Conf. Real-Time Syst. (ECRTS)*, 2019, pp. 153–630.

[38] S. Park, D. Song, H. Jang, M.-Y. Kwon, S.-H. Lee, H.-K. Kim, and H. Kim, "Interference analysis of multicore shared resources with a commercial avionics RTOS," in *Proc. IEEE/AIAA 38th Digit. Avionics Syst. Conf. (DASC)*, Sep. 2019, pp. 1–10.

[39] S. Park, M.-Y. Kwon, H.-K. Kim, and H. Kim, "Execution model to reduce the interference of shared memory in ARINC 653 compliant multicore RTOS," *Appl. Sci.*, vol. 10, no. 7, p. 2464, Apr. 2020.

[40] S. A. Rashid, G. Nelissen, and E. Tovar, "Cache persistence-aware memory bus contention analysis for multicore systems," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2020, pp. 442–447.

[41] F. Restuccia, A. Pagani, M. and Biondi, M. Marinoni, and G. Buttazzo, "Modeling and analysis of bus contention for hardware accelerators in FPGA SOcS," in *Proc. 32nd Euromicro Conf. Real-Time Syst. (ECRTS)*, 2020, pp. 1–23.

[42] J. Fang, M. Wang, and Z. Wei, "A memory scheduling strategy for eliminating memory access interference in heterogeneous system," *J. Supercomput.*, vol. 76, no. 4, pp. 3129–3154, 2020.

[43] C. Maia, G. Nelissen, L. Nogueira, L. M. Pinho, and D. G. Perez, "Schedulability analysis for global fixed-priority scheduling of the 3-phase task model," in *Proc. IEEE 23rd Int. Conf. Embedded Real-Time Comput. Syst. Appl. (RTCSA)*, Aug. 2017, pp. 1–10.

[44] B. Rouxel, S. Derrien, and I. Puaut, "Tightening contention delays while scheduling parallel applications on multi-core architectures," *ACM Trans. Embedded Comput. Syst.*, vol. 16, no. 5s, pp. 1–20, Oct. 2017.

[45] Y. Sun and G. Lipari, "Response time analysis with limited carry-in for global earliest deadline first scheduling," in *Proc. IEEE Real-Time Syst. Symp.*, Dec. 2015, pp. 130–140.

[46] S. Srinivasan, R. Kegley, M. Gerhardt, R. Hilliard, J. Preston, C. Granger, S. Drager, M. Anderson, R. Rosa, A. Charsagua, and R. Ha, "Empirical bounds of multicore cache interference for real-time schedulability analysis," in *Proc. IEEE/AIAA 38th Digit. Avionics Syst. Conf. (DASC)*, Sep. 2019, pp. 1–10.

[47] J. Xiao, S. Altmeyer, and A. D. Pimentel, "Schedulability analysis of global scheduling for multicore systems with shared caches," *IEEE Trans. Comput.*, vol. 69, no. 10, pp. 1487–1499, Oct. 2020.

[48] F. Markovic, J. Carlson, and R. Dobrin, "Cache-aware response time analysis for real-time tasks with fixed preemption points," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2020, pp. 30–42.

[49] J. Xiao and A. D. Pimentel, "CITTA: Cache interference-aware task partitioning for real-time multi-core systems," in *Proc. 21st ACM SIGPLAN/SIGBED Conf. Lang., Compil., Tools Embedded Syst.*, Jun. 2020, pp. 97–107.

[50] S. Z. Sheikh and M. A. Pasha, "A dynamic cache-partition schedulability analysis for partitioned scheduling on multicore real-time systems," *IEEE Lett. Comput. Soc.*, vol. 3, no. 2, pp. 46–49, Jul. 2020.

[51] J. Boudjadar and S. Nadjm-Tehrani, "Schedulability and memory interference analysis of multicore preemptive real-time systems," in *Proc. 8th ACM/SPEC Int. Conf. Perform. Eng.*, 2017, pp. 263–274.

[52] Z. Guo, K. Yang, F. Yao, and A. Awad, "Inter-task cache interference aware partitioned real-time scheduling," in *Proc. 35th Annu. ACM Symp. Appl. Comput.*, Mar. 2020, pp. 218–226.

[53] H. S. Chwa, J. Lee, J. Lee, K.-M. Phan, A. Easwaran, and I. Shin, "Global EDF schedulability analysis for parallel tasks on multi-core platforms," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 5, pp. 1331–1345, May 2017.

[54] M. Yang, W.-H. Huang, and J.-J. Chen, "Resource-oriented partitioning for multiprocessor systems with shared resources," *IEEE Trans. Comput.*, vol. 68, no. 6, pp. 882–898, Jun. 2019.

[55] Z. Al-Bayati, Y. Sun, H. Zeng, M. D. Natale, Q. Zhu, and B. H. Meyer, "Partitioning and selection of data consistency mechanisms for multicore real-time systems," *ACM Trans. Embedded Comput. Syst.*, vol. 18, no. 4, pp. 1–28, Jul. 2019.

[56] Z. Lei, X. Lei, and J. Long, "Real-time scheduling parallel tasks on multicore platforms," *J. Phys., Conf. Ser.*, vol. 1673, no. 1, Nov. 2020, Art. no. 012002.

[57] S. Sinha, R. West, and A. Golchin, "PAStime: Progress-aware scheduling for time-critical computing," 2019, *arXiv:1908.06211*.

[58] G. Schwäricke, T. Kloda, G. Gracioli, M. Bertogna, and M. Caccamo, "Fixed-priority memory-centric scheduler for COTS-based multiprocessors," in *Proc. 32nd Euromicro Conf. Real-Time Syst. (ECRTS)*, 2020, pp. 1–24.

[59] S. Zhuravlev, S. Blagodurov, and A. Fedorova, "Addressing shared resource contention in multicore processors via scheduling," *ACM SIGPLAN Notices*, vol. 45, no. 3, pp. 129–142, Mar. 2010.

[60] S. P. Muralidhara, L. Subramanian, O. Mutlu, M. Kandemir, and T. Moscibroda, "Reducing memory interference in multicore systems via application-aware memory channel partitioning," in *Proc. 44th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Dec. 2011, pp. 374–385.

[61] Y. Kim, M. Papamichael, O. Mutlu, and M. Harchol-Balter, "Thread cluster memory scheduling: Exploiting differences in memory access behavior," in *Proc. 43rd Annu. IEEE/ACM Int. Symp. Microarchitecture*, Dec. 2010, pp. 65–76.

[62] E. Ebrahimi, C. J. Lee, O. Mutlu, and Y. N. Patt, "Fairness via source throttling: A configurable and high-performance fairness substrate for multi-core memory systems," *ACM Sigplan Notices*, vol. 45, no. 3, pp. 335–346, 2010.

[63] R. Mancuso, R. Pellizzoni, M. Caccamo, L. Sha, and H. Yun, "WCET(m) estimation in multi-core systems using single core equivalence," in *Proc. 27th Euromicro Conf. Real-Time Syst.*, Jul. 2015, pp. 174–183.

[64] R. Mancuso, R. Pellizzoni, N. Tokcan, and M. Caccamo, "WCET derivation under single core equivalence with explicit memory budget assignment," in *Proc. 29th Euromicro Conf. Real-Time Syst. (ECRTS)*, 2017, pp. 1–23.

[65] H. Yun, R. Mancuso, Z.-P. Wu, and R. Pellizzoni, "PALLOC: DRAM bank-aware memory allocator for performance isolation on multicore platforms," in *Proc. IEEE 19th Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2014, pp. 155–166.

[66] X. Pan and F. Mueller, "Controller-aware memory coloring for multicore real-time systems," in *Proc. 33rd Annu. ACM Symp. Appl. Comput.*, Apr. 2018, pp. 584–592.

[67] X. Pan and F. Mueller, "NUMA-aware memory coloring for multicore real-time systems," *J. Syst. Archit.*, vol. 118, Sep. 2021, Art. no. 102188.

[68] X. Pan, Y. J. Gownivaripalli, and F. Mueller, "TintMalloc: Reducing memory access divergence via controller-aware coloring," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, May 2016, pp. 363–372.

[69] J. Kim, P. Shin, S. Noh, D. Ham, and S. Hong, "Reducing memory interference latency of safety-critical applications via memory request throttling and Linux cgroup," in *Proc. 31st IEEE Int. Syst. Chip Conf. (SOCC)*, Sep. 2018, pp. 215–220.

[70] A. Alhammad, S. Wasly, and R. Pellizzoni, "Memory efficient global scheduling of real-time tasks," in *Proc. 21st IEEE Real-Time Embedded Technol. Appl. Symp.*, Apr. 2015, pp. 285–296.

[71] H. Kim, D. de Niz, B. Andersson, M. Klein, O. Mutlu, and R. Rajkumar, "Bounding memory interference delay in COTS-based multi-core systems," in *Proc. IEEE 19th Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2014, pp. 145–154.
[72] D. Casini, A. Biondi, G. Nelissen, and G. Buttazzo, "A holistic memory contention analysis for parallel real-time tasks under partitioned scheduling," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2020, pp. 239–252.
[73] M. D. D. Dinechin, M. Schuh, M. Moy, and C. Maiza, "Scaling up the memory interference analysis for hard real-time many-core systems," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2020, pp. 330–333.
[74] M. Hassan, "Reduced latency DRAM for multi-core safety-critical real-time systems," *Real-Time Syst.*, vol. 56, pp. 171–206, Sep. 2019.
[75] L. Ecco and R. Ernst, "Improved DRAM timing bounds for real-time DRAM controllers with Read/Write bundling," in *Proc. IEEE Real-Time Syst. Symp.*, Dec. 2015, pp. 53–64.
[76] D. Guo and R. Pellizzoni, "A requests bundling DRAM controller for mixed-criticality systems," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2017, pp. 247–258.
[77] D. Guo, M. Hassan, R. Pellizzoni, and H. Patel, "A comparative study of predictable DRAM controllers," *ACM Trans. Embedded Comput. Syst.*, vol. 17, no. 2, pp. 1–23, Mar. 2018.
[78] W. Mi, X. Feng, J. Xue, and Y. Jia, "Software-hardware cooperative DRAM bank partitioning for chip multiprocessors," in *Proc. IFIP Int. Conf. Netw. Parallel Comput.* Berlin, Germany: Springer, 2010, pp. 329–343.
[79] R. E. Kessler and M. D. Hill, "Page placement algorithms for large real-indexed caches," *ACM Trans. Comput. Syst.*, vol. 10, no. 4, pp. 338–359, 1992.
[80] M. Xie, D. Tong, Y. Feng, K. Huang, and X. Cheng, "Page policy control with memory partitioning for DRAM performance and power efficiency," in *Proc. Int. Symp. Low Power Electron. Design (ISLPED)*, Sep. 2013, pp. 298–303.
[81] L. Liu, Z. Cui, M. Xing, Y. Bao, M. Chen, and C. Wu, "A software memory partition approach for eliminating bank-level interference in multicore systems," in *Proc. 21st Int. Conf. Parallel Archit. Compilation Techn. (PACT)*, 2012, pp. 367–375.
[82] G. Jia, L. Shi, X. Li, and D. Dai, "PUMA: From simultaneous to parallel for shared memory system in multi-core," *J. Signal Process. Syst.*, vol. 84, no. 1, pp. 139–150, Jul. 2016.
[83] X. Shen, F. Song, H. Meng, S. An, and Z. Zhang, "RBPP: A row based DRAM page policy for the many-core era," in *Proc. 20th IEEE Int. Conf. Parallel Distrib. Syst. (ICPADS)*, Dec. 2014, pp. 999–1004.
[84] J. Fang, J. Lu, and M. Cai, "Bank partitioning based adaptive page policy in multi-core memory systems," in *Proc. 14th Int. Symp. Distrib. Comput. Appl. Bus. Eng. Sci. (DCABES)*, Aug. 2015, pp. 240–243.
[85] D. Lee, L. Subramanian, R. Ausavarungnirun, J. Choi, and O. Mutlu, "Decoupled direct memory access: Isolating CPU and IO traffic by leveraging a dual-data-port DRAM," in *Proc. Int. Conf. Parallel Archit. Compilation (PACT)*, Oct. 2015, pp. 174–187.
[86] S. A. Rashid, "Server based task allocation to reduce inter-task memory interference in multicore systems," in *Proc. Int. Conf. Frontiers Inf. Technol. (FIT)*, Dec. 2019, pp. 322–3225.
[87] K. Bletsas and B. Andersson, "Preemption-light multiprocessor scheduling of sporadic tasks with high utilisation bound," *Real-Time Syst.*, vol. 47, no. 4, pp. 319–355, Jul. 2011.
[88] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha, "MemGuard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms," in *Proc. IEEE 19th Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2013, pp. 55–64.
[89] S. Mittal, "A survey of techniques for cache partitioning in multicore processors," *ACM Comput. Surv.*, vol. 50, no. 2, pp. 1–39, Mar. 2018.
[90] *Improving Real-Time Performance by Utilizing Cache Allocation Technology*, Intel Corporation, Santa Clara, CA, USA, Apr. 2015.
[91] M. Xu, L. Thi, X. Phan, H.-Y. Choi, and I. Lee, "VCAT: Dynamic cache management using CAT virtualization," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2017, pp. 211–222.
[92] Z. Satka and H. Hodzic, "Minimizing the unpredictability that real-time tasks suffer due to inter-core cache interference," Mälardalens Univ., Västerås, Sweden, Tech. Rep. 48512, 2020, p. 49.
[93] L. Pons, V. Selfa, J. Sahuquillo, S. Petit, and J. Pons, "Improving system turnaround time with Intel cat by identifying LLC critical applications," in *Proc. Eur. Conf. Parallel Process.* Berlin, Germany: Springer, 2018, pp. 603–615.
[94] B. Lee and E.-Y. Chung, "Efficient way-based cache partitioning for low-associativity cache," *IEICE Proc. Ser.*, vol. 61, pp. 1–4, Jul. 2016.
[95] V. Selfa, J. Sahuquillo, S. Petit, and M. E. Gomez, "A hardware approach to fairly balance the inter-thread interference in shared caches," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 11, pp. 3021–3032, Nov. 2017.
[96] N. El-Sayed, A. Mukkara, P.-A. Tsai, H. Kasture, X. Ma, and D. Sanchez, "KPart: A hybrid cache partitioning-sharing technique for commodity multicores," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2018, pp. 104–117.
[97] G. Sun, J. Shen, and A. V. Veidenbaum, "Combining prefetch control and cache partitioning to improve multicore performance," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, May 2019, pp. 953–962.
[98] M. Lee and S. Kim, "Time-sensitivity-aware shared cache architecture for multi-core embedded systems," *J. Supercomput.*, vol. 75, no. 10, pp. 6746–6776, Oct. 2019.
[99] N. Suzuki, H. Kim, D. D. Niz, B. Andersson, L. Wrage, M. Klein, and R. Rajkumar, "Coordinated bank and cache coloring for temporal protection of memory accesses," in *Proc. IEEE 16th Int. Conf. Comput. Sci. Eng.*, Dec. 2013, pp. 685–692.
[100] N. Kim, B. C. Ward, M. Chisholm, J. H. Anderson, and F. D. Smith, "Attacking the one-out-of-m multicore problem by combining hardware management with mixed-criticality provisioning," *Real-Time Syst.*, vol. 53, no. 5, pp. 709–759, Sep. 2017.
[101] Y. Ye, R. West, Z. Cheng, and Y. Li, "COLORIS: A dynamic cache partitioning system using page coloring," in *Proc. 23rd Int. Conf. Parallel Archit. Compilation*, Aug. 2014, pp. 381–392.
[102] A. Scolari, D. B. Bartolini, and M. D. Santambrogio, "A software cache partitioning system for hash-based caches," *ACM Trans. Archit. Code Optim.*, vol. 13, no. 4, pp. 1–24, Dec. 2016.
[103] H. Kim and R. Rajkumar, "Real-time cache management for multi-core virtualization," in *Proc. 13th Int. Conf. Embedded Softw.*, Oct. 2016, pp. 1–10.
[104] Y. Lim and H. Kim, "Cache-aware real-time virtualization for clustered multi-core platforms," *IEEE Access*, vol. 7, pp. 128628–128640, 2019.
[105] J. Park, H. Yeom, and Y. Son, "Page reusability-based cache partitioning for multi-core systems," *IEEE Trans. Comput.*, vol. 69, no. 6, pp. 812–818, Jun. 2020.
[106] T. Kloda, M. Solieri, R. Mancuso, N. Capodieci, P. Valente, and M. Bertogna, "Deterministic memory hierarchy and virtualization for modern multi-core embedded systems," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2019, pp. 1–14.
[107] S. A. Panchamukhi and F. Mueller, "Providing task isolation via TLB coloring," in *Proc. 21st IEEE Real-Time Embedded Technol. Appl. Symp.*, Apr. 2015, pp. 3–13.
[108] F. Bouquillon, C. Ballabriga, G. Lipari, and S. Niar, "A WCET-aware cache coloring technique for reducing interference in real-time systems," 2019, *arXiv:1903.09310*.
[109] X. Wang, S. Chen, J. Setter, and J. F. Martinez, "SWAP: Effective fine-grain management of shared last-level caches with minimum hardware support," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2017, pp. 121–132.
[110] J. Danielsson, M. Jägemar, M. Behnam, T. Seceleanu, and M. Sjödin, "Run-time cache-partition controller for multi-core systems," in *Proc. IECON 45th Annu. Conf. IEEE Ind. Electron. Soc.*, vol. 1, Oct. 2019, pp. 4509–4515.
[111] P. K. Valsan, H. Yun, and F. Farshchi, "Addressing isolation challenges of non-blocking caches for multicore real-time systems," *Real-Time Syst.*, vol. 53, no. 5, pp. 673–708, Sep. 2017.
[112] A. T. A. Dugo, J.-B. Lefoul, F. G. D. Magalhaes, D. Assal, and G. Nicolescu, "Cache locking content selection algorithms for ARINC-653 compliant RTOS," *ACM Trans. Embedded Comput. Syst.*, vol. 18, no. 5s, pp. 1–20, Oct. 2019.
[113] T. Zhang, W. Zheng, Y. Xiao, and G. Xu, "A dynamic instruction cache locking approach for minimizing worst case execution time of a single task," *IEEE Access*, vol. 8, pp. 208003–208015, 2020.
[114] A. Chousein and R. N. Mahapatra, "Fully associative cache partitioning with don't care bits for real-time applications," *ACM SIGBED Rev.*, vol. 2, no. 2, pp. 35–38, Apr. 2005.
[115] V. Suhendra and T. Mitra, "Exploring locking & partitioning for predictable shared caches on multi-cores," in *Proc. 45th Annu. Conf. Design Autom. (DAC)*, 2008, pp. 300–303.
[116] B. C. Ward, J. L. Herman, C. J. Kenna, and J. H. Anderson, "Outstanding paper award: Making shared caches more predictable on multicore platforms," in *Proc. 25th Euromicro Conf. Real-Time Syst.*, Jul. 2013, pp. 157–167.

[117] J. L. Herman, C. J. Kenna, M. S. Mollison, J. H. Anderson, and D. M. Johnson, "RTOS support for multicore mixed-criticality systems," in *Proc. IEEE 18th Real Time Embedded Technol. Appl. Symp.*, Apr. 2012, pp. 197–208.
[118] M. S. Mollison, J. P. Erickson, J. H. Anderson, S. K. Baruah, and J. A. Scoredos, "Mixed-criticality real-time scheduling for multicore systems," in *Proc. 10th IEEE Int. Conf. Comput. Inf. Technol.*, Jun. 2010, pp. 1864–1871.
[119] C. Gao, G. Lu, X. Yao, and J. Li, "An iterative pseudo-gap enumeration approach for the multidimensional multiple-choice knapsack problem," *Eur. J. Oper. Res.*, vol. 260, no. 1, pp. 1–11, Jul. 2017.
[120] D. Kroft, "Lockup-free instruction fetch/prefetch cache organization," in *Proc. 25 years Int. symposia Comput. Archit. (Selected Papers) (ISCA)*, 1998, pp. 195–201.
[121] I. Puaut, "Cache analysis vs static cache locking for schedulability analysis in multitasking real-time systems," in *Proc. WCET*, Vienna, Austria, 2002, pp. 1–6.
[122] S. Mittal, "A survey of techniques for cache locking," *ACM Trans. Design Autom. Electron. Syst.*, vol. 21, no. 3, pp. 1–24, Jul. 2016.
[123] A. Asaduzzaman, F. N. Sibai, and M. Rani, "Improving cache locking performance of modern embedded systems via the addition of a miss table at the l2 cache level," *J. Syst. Archit.*, vol. 56, nos. 4–6, pp. 151–162, Apr. 2010.
[124] A. Sarkar, F. Mueller, and H. Ramaprasad, "Predictable task migration for locked caches in multi-core systems," in *Proc. SIGPLAN/SIGBED Conf. Lang., Compil. Tools Embedded Syst. (LCTES)*, 2011, pp. 131–140.
[125] R. Mancuso, R. Dudko, E. Betti, M. Cesati, M. Caccamo, and R. Pellizzoni, "Real-time cache management framework for multi-core architectures," in *Proc. IEEE 19th Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2013, pp. 45–54.
[126] M. Shekhar, A. Sarkar, H. Ramaprasad, and F. Mueller, "Semi-partitioned hard-real-time scheduling under locked cache migration in multicore systems," in *Proc. 24th Euromicro Conf. Real-Time Syst.*, Jul. 2012, pp. 331–340.
[127] A. Sarkar, F. Mueller, and H. Ramaprasad, "Static task partitioning for locked caches in multicore real-time systems," *ACM Trans. Embedded Comput. Syst.*, vol. 14, no. 1, pp. 1–30, Jan. 2015.
[128] A. M. Kaushik, M. Hassan, and H. Patel, "Designing predictable cache coherence protocols for multi-core real-time systems," *IEEE Trans. Comput.*, vol. 70, no. 12, pp. 2098–2111, Dec. 2021.
[129] M. Hassan, A. M. Kaushik, and H. Patel, "Predictable cache coherence for multi-core real-time systems," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2017, pp. 235–246.
[130] A. Saparon and F. N. B. Razlan, "Cache coherence protocols in multiprocessor," in *Proc. Int. Conf. Comput. Sci. Inf. Syst. (ICSIS)*, 2014, pp. 17–18.
[131] M. M. K. Martin, M. D. Hill, and D. J. Sorin, "Why on-chip cache coherence is here to stay," *Commun. ACM*, vol. 55, no. 7, pp. 78–89, Jul. 2012.
[132] P. Stenstrom, "A survey of cache coherence schemes for multiprocessors," *Computer*, vol. 23, no. 6, pp. 12–24, Jun. 1990.
[133] N. Sritharan, A. Kaushik, M. Hassan, and H. Patel, "Enabling predictable, simultaneous and coherent data sharing in mixed criticality systems," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, Dec. 2019, pp. 433–445.
[134] R. Fuchsen, "How to address certification for multi-core based IMA platforms: Current status and potential solutions," in *Proc. 29th Digit. Avionics Syst. Conf.*, Oct. 2010, p. 5.
[135] M. Hassan, "Discriminative coherence: Balancing performance and latency bounds in data-sharing multi-core real-time systems," in *Proc. 32nd Euromicro Conf. Real-Time Syst. (ECRTS)*, 2020, pp. 1–24.
[136] K. Sivakumaran and A. Siromoney, "Priority based yield of shared cache to provide cache QoS in multicore systems," *Int. J. Parallel Program.*, vol. 45, no. 3, pp. 634–656, Jun. 2017.
[137] Y. Chen, W. Li, C. Kim, and Z. Tang, "Efficient shared cache management through sharing-aware replacement and streaming-aware insertion policy," in *Proc. IEEE Int. Symp. Parallel Distrib. Process.*, May 2009, pp. 1–11.
[138] S. P. Muralidhara, M. Kandemir, and P. Raghavan, "Intra-application cache partitioning," in *Proc. IEEE Int. Symp. Parallel Distrib. Process. (IPDPS)*, Apr. 2010, pp. 1–12.
[139] K. T. Sundararajan, T. M. Jones, and N. P. Topham, "RECAP: Region-aware cache partitioning," in *Proc. IEEE 31st Int. Conf. Comput. Design (ICCD)*, Oct. 2013, pp. 294–301.
[140] K. Varadarajan, S. K. Nandy, V. Sharda, A. Bharadwaj, R. Iyer, S. Makineni, and D. Newell, "Molecular caches: A caching structure for dynamic creation of application-specific heterogeneous cache regions," in *Proc. 39th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Dec. 2006, pp. 433–442.
[141] M. Qureshi and Y. Patt, "Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches," in *Proc. 39th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Dec. 2006, pp. 423–432.
[142] C. Liu, A. Sivasubramaniam, and M. Kandemir, "Organizing the last line of defense before hitting the memory wall for CMPs," in *Proc. 10th Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2004, pp. 176–185.
[143] S. Srikantaiah, M. Kandemir, and M. J. Irwin, "Adaptive set pinning: Managing shared caches in chip multiprocessors," *ACM SIGPLAN Notices*, vol. 43, no. 3, pp. 135–144, Mar. 2008.
[144] R. Iyer, "CQoS: A framework for enabling QoS in shared caches of CMP platforms," in *Proc. 18th Annu. Int. Conf. Supercomput. (ICS)*, 2004, pp. 257–266.
[145] N. Rafique, W.-T. Lim, and M. Thottethodi, "Architectural support for operating system-driven CMP cache management," in *Proc. 15th Int. Conf. Parallel Archit. Compilation Techn. (PACT)*, 2006, pp. 2–12.
[146] D. Tam, R. Azimi, L. Soares, and M. Stumm, "Managing shared L2 caches on multicore systems in software," in *Proc. Workshop Interact. Between Operating Syst. Comput. Archit.*, 2007, pp. 26–33.
[147] J. Lin, Q. Lu, X. Ding, Z. Zhang, X. Zhang, and P. Sadayappan, "Gaining insights into multicore cache partitioning: Bridging the gap between simulation and real systems," in *Proc. IEEE 14th Int. Symp. High Perform. Comput. Archit.*, Feb. 2008, pp. 367–378.
[148] X. Zhang, S. Dwarkadas, and K. Shen, "Towards practical page coloring-based multicore cache management," in *Proc. 4th ACM Eur. Conf. Comput. Syst. (EuroSys)*, 2009, pp. 89–102.
[149] N. Guan, M. Stigge, W. Yi, and G. Yu, "Cache-aware scheduling and analysis for multicores," in *Proc. 7th ACM Int. Conf. Embedded Softw. (EMSOFT)*, 2009, pp. 245–254.
[150] H. Kim, A. Kandhalu, and R. Rajkumar, "A coordinated approach for practical OS-level cache management in multi-core real-time systems," in *Proc. 25th Euromicro Conf. Real-Time Syst.*, Jul. 2013, pp. 80–89.
[151] A. Jaleel, K. B. Theobald, S. C. Steely, Jr., and J. Emer, "High performance cache replacement using re-reference interval prediction (RRIP)," *ACM SIGARCH Comput. Archit. News*, vol. 38, no. 3, pp. 60–71, 2010.
[152] D. Zhan, H. Jiang, and S. C. Seth, "CLU: Co-optimizing locality and utility in thread-aware capacity management for shared last level caches," *IEEE Trans. Comput.*, vol. 63, no. 7, pp. 1656–1667, Jul. 2014.
[153] S. Pai, N. Singh, and V. Singh, "AB-aware: Application behavior aware management of shared last level caches," in *Proc. Great Lakes Symp. (VLSI)*, May 2018, pp. 237–242.
[154] T. S. Warrier, "ACR: Application aware cache replacement for shared caches in multi-core systems," *Int. J. Comput. Eng. Technol.*, vol. 10, no. 2, pp. 1–3, Apr. 2019.
[155] N. Duong, D. Zhao, T. Kim, R. Cammarota, M. Valero, and A. V. Veidenbaum, "Improving cache management policies using dynamic reuse distances," in *Proc. 45th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Dec. 2012, pp. 389–400.
[156] P. Yang, Q. Wang, H. Ye, and Z. Zhang, "Partially shared cache and adaptive replacement algorithm for NoC-based many-core systems," *J. Syst. Archit.*, vol. 98, pp. 424–433, Sep. 2019.
[157] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha, "Memory bandwidth management for efficient performance isolation in multi-core platforms," *IEEE Trans. Comput.*, vol. 65, no. 2, pp. 562–576, Feb. 2016.
[158] H. Yun, W. Ali, S. Gondi, and S. Biswas, "BWLOCK: A dynamic memory access control framework for soft real-time applications on multicore platforms," *IEEE Trans. Comput.*, vol. 66, no. 7, pp. 1247–1252, Jul. 2017.
[159] A. Agrawal, G. Fohler, J. Freitag, J. Nowotsch, S. Uhrig, and M. Paulitsch, "Contention-aware dynamic memory bandwidth isolation with predictability in COTS multicores: An avionics case study," in *Proc. 29th Euromicro Conf. Real-Time Syst. (ECRTS)*, 2017, p. 39.
[160] A. Agrawal, R. Mancuso, R. Pellizzoni, and G. Fohler, "Analysis of dynamic memory bandwidth regulation in multi-core real-time systems," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, Dec. 2018, pp. 230–241.
[161] M. Hassan, H. Patel, and R. Pellizzoni, "PMC: A requirement-aware DRAM controller for multicore mixed criticality systems," *ACM Trans. Embedded Comput. Syst.*, vol. 16, no. 4, pp. 1–28, Nov. 2017.

[162] Y. Krishnapillai, Z. P. Wu, and R. Pellizzoni, "A rank-switching, open-row DRAM controller for time-predictable systems," in *Proc. 26th Euromicro Conf. Real-Time Syst.*, Jul. 2014, pp. 27–38.

[163] G. Yao, R. Pellizzoni, S. Bak, H. Yun, and M. Caccamo, "Global real-time memory-centric scheduling for multicore systems," *IEEE Trans. Comput.*, vol. 65, no. 9, pp. 2739–2751, Sep. 2016.

[164] R. Pellizzoni, E. Betti, S. Bak, G. Yao, J. Criswell, M. Caccamo, and R. Kegley, "A predictable execution model for COTS-based embedded systems," in *Proc. 17th IEEE Real-Time Embedded Technol. Appl. Symp.*, Apr. 2011, pp. 269–279.

[165] C. Pagetti, J. Forget, H. Falk, D. Oehlert, and A. Luppold, "Automated generation of time-predictable executables on multicore," in *Proc. 26th Int. Conf. Real-Time Netw. Syst.*, Oct. 2018, pp. 104–113.

[166] R. Gifford, N. Gandhi, L. T. X. Phan, and A. Haeberlen, "DNA: Dynamic resource allocation for soft real-time multicore systems," in *Proc. IEEE 27th Real-Time Embedded Technol. Appl. Symp. (RTAS)*, May 2021, pp. 196–209.

[167] T. Carle, D. Potop-Butucaru, Y. Sorel, and D. Lesens, "From dataflow specification to multiprocessor partitioned time-triggered real-time implementation," *Leibniz Trans. Embedded Syst.*, vol. 2, no. 2, p. 01, 2015.

[168] Q. Perret, P. Maurere, E. Noulard, C. Pagetti, P. Sainrat, and B. Triquet, "Temporal isolation of hard real-time applications on many-core processors," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2016, pp. 1–11.

[169] A. Schranzhofer, J.-J. Chen, and L. Thiele, "Timing analysis for TDMA arbitration in resource sharing systems," in *Proc. 16th IEEE Real-Time Embedded Technol. Appl. Symp.*, Apr. 2010, pp. 215–224.

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

[171] D. Oehlert, A. Luppold, and H. Falk, "Bus-aware static instruction SPM allocation for multicore hard real-time systems," in *Proc. 29th Euromicro Conf. Real-Time Syst. (ECRTS)*, 2017, pp. 1–22.

[172] G. Durrieu, M. Faugère, S. Girbal, D. Gracia Pérez, C. Pagetti, and W. Puffitsch, "Predictable flight management system implementation on a multicore processor," in *Proc. Embedded Real Time Softw. (ERTS)*, TOULOUSE, France, Feb. 2014, pp. 1–9. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01121700

[173] F. Hebbache, M. Jan, F. Brandner, and L. Pautet, "Dynamic arbitration of memory requests with TDM-like guarantees," in *Proc. Int. Workshop Compositional Theory Technol. Real-Time Embedded Syst. (CRTS)*, 2017, pp. 1–4.

[174] F. Hebbache, M. Jan, F. Brandner, and L. Pautet, "Shedding the shackles of time-division multiplexing," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, Dec. 2018, pp. 456–468.

[175] J.-J. Chen, "Partitioned multiprocessor fixed-priority scheduling of sporadic real-time tasks," in *Proc. 28th Euromicro Conf. Real-Time Syst. (ECRTS)*, Jul. 2016, pp. 251–261.

[176] B. B. Brandenburg and M. Gul, "Global scheduling not required: Simple, near-optimal multiprocessor real-time scheduling with semi-partitioned reservations," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, Nov. 2016, pp. 99–110.

[177] D. Casini, A. Biondi, and G. Buttazzo, "Semi-partitioned scheduling of dynamic real-time workload: A practical approach based on analysis-driven load balancing," in *Proc. 29th Euromicro Conf. Real-Time Syst. (ECRTS)*, 2017, pp. 1–23.

[178] A. Burns, R. I. Davis, P. Wang, and F. Zhang, "Partitioned EDF scheduling for multiprocessors using a C=D task splitting scheme," *Real-Time Syst.*, vol. 48, no. 1, pp. 3–33, Jan. 2012.

[179] B. Yalcinkaya, M. Nasri, and B. B. Brandenburg, "An exact schedulability test for non-preemptive self-suspending real-time tasks," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2019, pp. 1228–1233.

[180] Y. Zhao, R. Zhou, and H. Zeng, "An optimization framework for real-time systems with sustainable schedulability analysis," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, Dec. 2020, pp. 333–344.

[181] J. Lee, S. Y. Shin, S. Nejati, L. C. Briand, and Y. I. Parache, "Schedulability analysis of real-time systems with uncertain worst-case execution times," 2020, *arXiv:2007.10490*.

[182] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Comput. Surveys*, vol. 43, no. 4, pp. 1–44, Oct. 2011.

[183] O. U. P. Zapata and P. M. Alvarez, "Edf and RM multiprocessor scheduling algorithms: Survey and performance evaluation," *Seccion de Computacion Av. IPN*, vol. 2508, pp. 1–22, Feb. 2002.

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

[185] R. I. Davis and A. Burns, "Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems," *Real-Time Syst.*, vol. 47, no. 1, pp. 1–40, 2011.

[186] M. Bertogna and M. Cirinei, "Response-time analysis for globally scheduled symmetric multiprocessor platforms," in *Proc. 28th IEEE Int. Real-Time Syst. Symp. (RTSS)*, Dec. 2007, pp. 149–160.

[187] J. Chen, C. Du, F. Xie, and Z. Yang, "Schedulability analysis of non-preemptive strictly periodic tasks in multi-core real-time systems," *Real-Time Syst.*, vol. 52, no. 3, pp. 239–271, May 2016.

[188] J. Lee, "Improved schedulability analysis using carry-in limitation for non-preemptive fixed-priority multiprocessor scheduling," *IEEE Trans. Comput.*, vol. 66, no. 10, pp. 1816–1823, Oct. 2017.

[189] H. Baek and J. Lee, "Improved schedulability test for non-preemptive fixed-priority scheduling on multiprocessors," *IEEE Embedded Syst. Lett.*, vol. 12, no. 4, pp. 129–132, Dec. 2020.

[190] H. Baek, N. Jung, H. S. Chwa, I. Shin, and J. Lee, "Non-preemptive scheduling for mixed-criticality real-time multiprocessor systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 8, pp. 1766–1779, Aug. 2018.

[191] Y.-W. Zhang, "Energy-aware non-preemptive scheduling of mixed-criticality real-time task systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, early access, Oct. 14, 2021, doi: 10.1109/TCAD.2021.3120326.

[192] Y. Sun and M. Di Natale, "Pessimism in multicore global schedulability analysis," *J. Syst. Archit.*, vol. 97, pp. 142–152, Aug. 2019.

[193] Y. Sun and M. Di Natale, "Assessing the pessimism of current multicore global fixed-priority schedulability analysis," in *Proc. 33rd Annu. ACM Symp. Appl. Comput.*, Apr. 2018, pp. 575–583.

[194] H. Lee and J.-Y. Choi, "Constraint-based schedulability analysis in multiprocessor real-time systems," *IEEE Access*, vol. 8, pp. 165168–165177, 2020.

[195] D. Lee, H. Jung, and H. Yang, "Real-time schedulability analysis and enhancement of transiently powered processors with NVMs," *IEEE Trans. Comput.*, vol. 70, no. 3, pp. 372–383, Mar. 2021.

[196] H. Baek and J. Lee, "Improved schedulability analysis of the contention-free policy for real-time systems," *J. Syst. Softw.*, vol. 154, pp. 112–124, Aug. 2019.

[197] W. Wang, "Schedulability analysis and symbolic verification method for heterogeneous multicore real-time systems," *Int. J. Performability Eng.*, vol. 13, no. 6, p. 785, 2017.

[198] S. Moulik, R. Devaraj, and A. Sarkar, "HETERO-SCHED: A low-overhead heterogeneous multi-core scheduler for real-time periodic tasks," in *Proc. IEEE 20th Int. Conf. High Perform. Comput. Commun.; IEEE 16th Int. Conf. Smart City; IEEE 4th Int. Conf. Data Sci. Syst. (HPCC/SmartCity/DSS)*, Jun. 2018, pp. 659–666.

[199] S. Funk, G. Levin, C. Sadowski, I. Pye, and S. Brandt, "DP-fair: A unifying theory for optimal hard real-time multiprocessor scheduling," *Real-Time Syst.*, vol. 47, pp. 389–429, Sep. 2011.

[200] G. Raravi, B. Andersson, V. Nélis, and K. Bletsas, "Task assignment algorithms for two-type heterogeneous multiprocessors," *Real-Time Syst.*, vol. 50, no. 1, pp. 87–141, Jan. 2014.

[201] S. Moulik, R. Devaraj, and A. Sarkar, "COST: A cluster-oriented scheduling technique for heterogeneous multi-cores," in *Proc. IEEE Int. Conf. Syst., Man, Cybern. (SMC)*, Oct. 2018, pp. 1951–1957.

[202] H. S. Chwa, J. Seo, J. Lee, and I. Shin, "Optimal real-time scheduling on two-type heterogeneous multicore platforms," in *Proc. IEEE Real-Time Syst. Symp.*, Dec. 2015, pp. 119–129.

[203] A. Bertout, J. Goossens, E. Grolleau, and X. Poczekajlo, "Template schedule construction for global real-time scheduling on unrelated multiprocessor platforms," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2020, pp. 216–221.

[204] S. Baruah, "Feasibility analysis of preemptive real-time systems upon heterogeneous multiprocessor platforms," in *Proc. 25th IEEE Int. Real-Time Syst. Symp.*, Dec. 2004, pp. 37–46.

[205] P. P. Nair, R. Devaraj, and A. Sarkar, "FEST: Fault-tolerant energy-aware scheduling on two-core heterogeneous platform," in *Proc. 8th Int. Symp. Embedded Comput. Syst. Design (ISED)*, Dec. 2018, pp. 63–68.

[206] W. Zhang, Y. Hu, H. He, Y. Liu, and A. Chen, "Linear and dynamic programming algorithms for real-time task scheduling with task duplication," *J. Supercomput.*, vol. 75, no. 2, pp. 494–509, Feb. 2019.
[207] R. Devaraj, "A solution to drawbacks in capturing execution requirements on heterogeneous platforms," *J. Supercomput.*, vol. 76, pp. 6901–6916, Jan. 2020.
[208] S.-W. Cheng, J.-J. Chen, J. Reineke, and T.-W. Kuo, "Memory bank partitioning for fixed-priority tasks in a multi-core system," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, Dec. 2017, pp. 209–219.
[209] W.-H. Huang, J.-J. Chen, and J. Reineke, "MIRROR: Symmetric timing analysis for real-time tasks on multicore platforms with shared resources," in *Proc. 53rd Annu. Design Autom. Conf.*, Jun. 2016, pp. 1–6.
[210] H. Rihani, M. Moy, C. Maiza, R. I. Davis, and S. Altmeyer, "Response time analysis of synchronous data flow programs on a many-core processor," in *Proc. 24th Int. Conf. Real-Time Netw. Syst. (RTNS)*, 2016, pp. 67–76.
[211] J. Durand, Y. Bouchebaba, and L. Santinelli, "Statistical analysis for shared resources effects with multi-core real-time systems," in *Proc. IEEE 13th Int. Symp. Embedded Multicore/Many-Core Syst. Chip (MCSoC)*, Oct. 2019, pp. 362–371.
[212] M. Hassan and R. Pellizzoni, "Analysis of memory-contention in heterogeneous COTS MPSoCs," in *Proc. 32nd Euromicro Conf. Real-Time Syst. (ECRTS)*, 2020, pp. 1–24.
[213] S. Reder and J. Becker, "Interference-aware memory allocation for real-time multi-core systems," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2020, pp. 148–159.
[214] G. Yao, H. Yun, Z. P. Wu, R. Pellizzoni, M. Caccamo, and L. Sha, "Schedulability analysis for memory bandwidth regulated multicore real-time systems," *IEEE Trans. Comput.*, vol. 65, no. 2, pp. 601–614, Feb. 2016.
[215] H. Usui, L. Subramanian, K. K.-W. Chang, and O. Mutlu, "DASH: Deadline-aware high-performance memory scheduler for heterogeneous systems with hardware accelerators," *ACM Trans. Archit. Code Optim.*, vol. 12, no. 4, pp. 1–28, 2016.
[216] M. Xu, L. T. X. Phan, H.-Y. Choi, and I. Lee, "Analysis and implementation of global preemptive fixed-priority scheduling with dynamic cache allocation," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2016, pp. 1–12.
[217] J. Xiao, S. Altmeyer, and A. Pimentel, "Schedulability analysis of non-preemptive real-time scheduling for multicore processors with shared caches," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, Dec. 2017, pp. 199–208.
[218] 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 Syst.*, vol. 54, no. 3, pp. 607–661, Jul. 2018.
[219] V. A. Nguyen, D. Hardy, and I. Puaut, "Cache-conscious off-line real-time scheduling for multi-core platforms: Algorithms and implementation," *Real-Time Syst.*, vol. 55, no. 4, pp. 810–849, Oct. 2019.
[220] S. Z. Sheikh and M. A. Pasha, "Energy-efficient real-time scheduling on multicores: A novel approach to model cache contention," *ACM Trans. Embedded Comput. Syst.*, vol. 19, no. 4, pp. 1–25, Jul. 2020.
[221] J. Choi, D. Kang, and S. Ha, "Conservative modeling of shared resource contention for dependent tasks in partitioned multi-core systems," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2016, pp. 181–186.
[222] S. Altmeyer, R. I. Davis, L. Indrusiak, C. Maiza, V. Nelis, and J. Reineke, "A generic and compositional framework for multicore response time analysis," in *Proc. 23rd Int. Conf. Real Time Netw. Syst.*, Nov. 2015, pp. 129–138.
[223] A. Boudjadar, A. David, J. H. Kim, K. G. Larsen, M. Mikučionis, U. Nyman, and A. Skou, "A reconfigurable framework for compositional schedulability and power analysis of hierarchical scheduling systems with frequency scaling," *Sci. Comput. Program.*, vol. 113, pp. 236–260, Dec. 2015.
[224] B. Andersson, H. Kim, D. D. Niz, M. Klein, R. Rajkumar, and J. Lehoczky, "Schedulability analysis of tasks with corunner-dependent execution times," *ACM Trans. Embedded Comput. Syst.*, vol. 17, no. 3, pp. 1–29, May 2018.
[225] J. M. Aceituno, A. Guasque, P. Balbastre, J. Simó, and A. Crespo, "Hardware resources contention-aware scheduling of hard real-time multiprocessor systems," *J. Syst. Archit.*, vol. 118, Sep. 2021, Art. no. 102223.

**TAMARA LUGO** (Member, IEEE) received the master's degree in electronic systems and applications engineering. He is currently pursuing the Ph.D. degree in computer science and technology with the Universidad Carlos III de Madrid, Spain. He is also a Telecommunications and Electronics Engineer. His research interest includes embedded real-time systems applied to space systems.

**SANTIAGO LOZANO** is currently pursuing the Ph.D. degree in computer science and technology with the Carlos III University of Madrid. He is also an Aerospace and Defense Software Engineer at SENER Aeroespacial, where he has participated in numerous European projects, such as the development of the Vega-C Rocket Navigation Unit or the Future Combat Air System (FCAS) Program.

**JAVIER FERNÁNDEZ** (Member, IEEE) was a Visiting Researcher with the EPCC Supercomputing Center, Edinburgh, U.K., in 2009 and 2019, and the HLRS Supercomputing Center, Stuttgart, Germany, in 2011. He is currently an Associate Professor with the Computer Science and Engineering Department, University Carlos III of Madrid, Spain. He has published more than 60 journals and conference papers, mainly related with HPC, parallel programming frameworks, and real time/embedded systems. He has participated in 26 publicly funded research projects and 14 technology transfer contracts, several with major companies in the aerospace field, such as EADS and GMV or in the railways field, such as RENFE or ADIF.

**JESUS CARRETERO** (Senior Member, IEEE) has been a Full Professor in computer architecture and technology with the Universidad Carlos III de Madrid, Spain, since 2002. His research interests include high-performance computing systems, large-scale distributed systems, and real-time systems. He is currently involved in three other EU projects, coordinating the ADMIRE FET-HPC. He was the Action Chair of the IC1305 COST Action "Network for Sustainable Ultrascale Computing Systems (NESUS)." He has been the General Chair of HPCC 2011, MUE 2012, ISPA 2016, and CCGRID 2017. He is a Senior Member of the IEEE Computer Society.