SP-IMPact: A Framework for Static Partitioning Interference Mitigation and Performance Analysis

Diogo Costa ✉ 🔾
Centro ALGORITMI / LASI, Universidade do Minho, Portugal

Gonçalo Moreira ✉ 🔾
Centro ALGORITMI / LASI, Universidade do Minho, Portugal

Afonso Oliveira ✉ 🔾
Centro ALGORITMI / LASI, Universidade do Minho, Portugal

José Martins ✉ 🔾
Centro ALGORITMI / LASI, Universidade do Minho, Portugal

Sandro Pinto ✉ 🔾
Centro ALGORITMI / LASI, Universidade do Minho, Portugal

── Abstract ──

Modern embedded systems are evolving toward complex, heterogeneous architectures to accommodate increasingly demanding applications. Driven by industry SWAP-C (Size, Weight, Power, and Cost) constraints, this shift has led to the consolidation of multiple systems onto single hardware platforms. Static Partitioning Hypervisors (SPHs) offer a promising solution to partition hardware resources and provide spatial isolation between critical workloads. However, shared hardware resources like the Last-Level Cache (LLC) and system bus can introduce significant temporal interference between virtual machines (VMs), negatively impacting performance and predictability. Over the past decade, academia and industry have focused on developing interference mitigation techniques, such as cache partitioning and memory bandwidth reservation. Configuring these techniques, however, is complex and time-consuming. Cache partitioning requires careful balancing of cache sections across VMs, while memory bandwidth reservation requires tuning bandwidth budgets and periods. With numerous possible configurations, testing all combinations is impractical and often leads to suboptimal configurations. Moreover, there is a gap in understanding how these techniques interact, as their combined use can result in compounded or conflicting effects on system performance. Static analysis solutions that estimate worst-case execution times (WCET) and upper bounds on execution times provide some guidance for configuring interference mitigation techniques. While useful in identifying potential interference effects, these tools often fail to capture the full complexity of modern multi-core systems, as they typically focus on a limited set of shared resources and neglect other sources of contention, such as IOMMUs and interrupt controllers. To address these challenges, we introduce SP-IMPact, an open-source framework designed to analyze and guide the configuration of interference mitigation techniques, through the deployment of diverse VM configurations and setups, and assessment of hardware-level contention (leveraging SPHs). It supports two mitigation techniques: (i) cache coloring and (ii) memory bandwidth reservation, while also evaluating the interactions between these techniques and their cumulative impact on system performance. By providing insights on real hardware platforms, SP-IMPact helps to optimize the configuration of these techniques in mixed-criticality systems, ensuring both performance and predictability.

1   Introduction

In recent decades, a significant trend toward digitization has revolutionized various industries including automotive, robotics, medical, and aerospace [9, 40, 41]. This shift brought an exponential increase in system features, prompting high-end embedded platforms to evolve from basic designs. Past simple MCUs with single cores have given way to today's intricate and highly complex platforms [11]. The transition from single-core to multi-core architectures, accommodating multiple CPUs, and integrating diverse hardware accelerators like Graphics Processing Units (GPUs), Tensor Processing Units (TPUs), Neural Processing Units (NPUs), and Field-Programmable Gate Arrays (FPGAs) [18, 33, 37], has fundamentally altered the landscape, resulting in highly heterogeneous designs.

Simultaneously, market demands for compact and efficient systems have driven the consolidation of multiple functionalities onto single hardware platforms to meet Size, Weight, Power, and Cost (SWaP-C) constraints. This consolidation has led to the rise of Mixed Criticality Systems (MCSs) [21], where components with varying criticality levels coexist on the same platform. Virtualization technologies have been instrumental in enabling such consolidation, with hypervisor-based solutions – particularly static-partitioning hypervisors [35, 34, 41, 22, 27] – striking a balance between safety, security, and resource efficiency. These hypervisors allow for the deployment of diverse workloads within MCSs while adhering to stringent industry safety standards, such as ISO 26262 [39].

Achieving robust system consolidation requires addressing critical challenges to ensure safety and security, particularly spatial and temporal isolation. Spatial isolation guarantees that architectural resources (e.g., CPUs and main memory) allocated to one system remain inaccessible to others. Temporal isolation ensures that the execution of one system's workloads does not interfere with another's timing requirements. While static partitioning effectively addresses spatial isolation, temporal isolation remains a significant challenge due to contention on shared microarchitectural resources like the Last-Level Cache (LLC), main memory, and system bus. Such contention leads to increased execution times and reduced determinism [1, 7, 8, 31, 1], making timing predictability particularly difficult for hard real-time systems.

Techniques such as cache partitioning [17] and memory bandwidth reservation [51] have emerged as promising solutions to mitigate temporal interference in MCSs. Cache partitioning segments the LLC into regions assigned to specific Virtual Machines (VMs), while memory bandwidth reservation regulates the number of memory accesses within a given time frame. However, configuring these techniques effectively requires careful balancing of resources across VMs and fine-tuning parameters (e.g., define cache regions and/or memory budgets and periods). This process is complex, time-consuming, and impractical for real-world MCSs, often leading to suboptimal configurations. Static analysis tools [1, 3] have been explored to address these challenges, offering a means to understand interference impacts in MCSs and guide the configuration of mitigation techniques. By estimating Worst-Case Execution Time (WCET) and quantifying interference effects, these tools provide a foundation for informed decision-making. However, existing static analysis solutions often focus on specific shared resources, such as the LLC, overlooking other shared hardware resources (e.g., IOMMUs and interrupt controllers).

To address these limitations, we introduce SP-IMPact, an open-source framework[1] designed to analyze and support the configuration of interference mitigation techniques. SP-IMPact enables a comprehensive understanding of the impact of shared hardware resources

---

[1] https://gitlab.com/ESRGv3/sp-impact
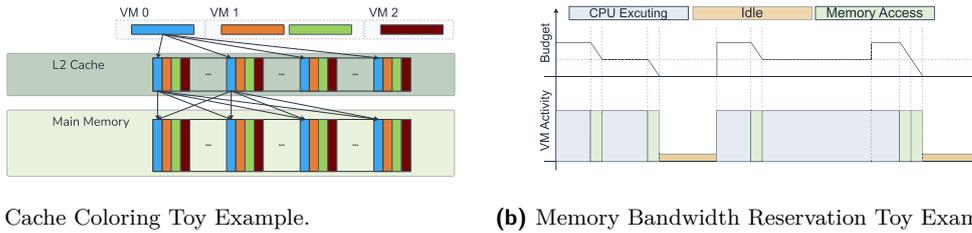
(a) Cache Coloring Toy Example.

(b) Memory Bandwidth Reservation Toy Example.

Figure 1 Illustrative examples of cache coloring and memory bandwidth reservation mechanisms.

on real platforms by considering all potential sources of contention - filling critical gaps left by currently available solutions. Furthermore, it allows the deployment of diverse configurations of cache coloring and memory bandwidth reservation to evaluate their effects on the workloads of different VMs. The insights gained through SP-IMPact can guide the optimization of these techniques, easing their usage by industry due to the framework's workload-agnostic design, which supports various operating systems and workloads. For academia, SP-IMPact provides a versatile launchpad for deploying and testing new interference mitigation techniques. While this paper leverages the Bao hypervisor [34] as a use case, SP-IMPact is agnostic to the underlying hypervisor and can be extended to support other static partitioning hypervisors.

2 Background

The consolidation of MCSs introduces a well-known challenge: interference between co-existing VMs, which can degrade performance and disrupt real-time guarantees [18, 34, 28, 50, 48, 51, 4, 46, 13, 30, 29]. This interference typically arises from contention over shared micro-architectural resources, such as the LLC, main memory, and the system bus, leading to increased execution time and lack of determinism. To mitigate these issues, techniques such as cache partitioning [28, 36, 26] and memory bandwidth reservation [51, 50, 36, 15], among others solutions targeting I/O and interrupts regulation [52, 16, 12] , have been proposed.

Cache Partitioning. Cache partitioning techniques enable the selective allocation of cache regions to specific workloads, thereby reducing cache contention and improving predictability. One of the most used techniques for cache partitioning is cache coloring, which divides the LLC into distinct regions by assigning specific "colors" to individual workloads, where each "color" corresponds to different cache sets, which helps to control cache access patterns and minimize interference. Cache coloring is commonly used to assign dedicated portions of the cache to individual VMs, effectively limiting LLC contention, as depicted in Figure 1 (a).

Memory Bandwidth Reservation. Memory bandwidth reservation techniques regulate the memory access rate to reduce interference and ensure temporal isolation. MemGuard [51], for example, limits the memory access rate per CPU by allocating specific portions of memory bandwidth, or "budgets", over defined periods, as depicted in Figure 1 (b). This prevents any workload from monopolizing memory, ensuring fair resource distribution and reducing delays, thus improving predictable performance and minimizing interference between workloads.
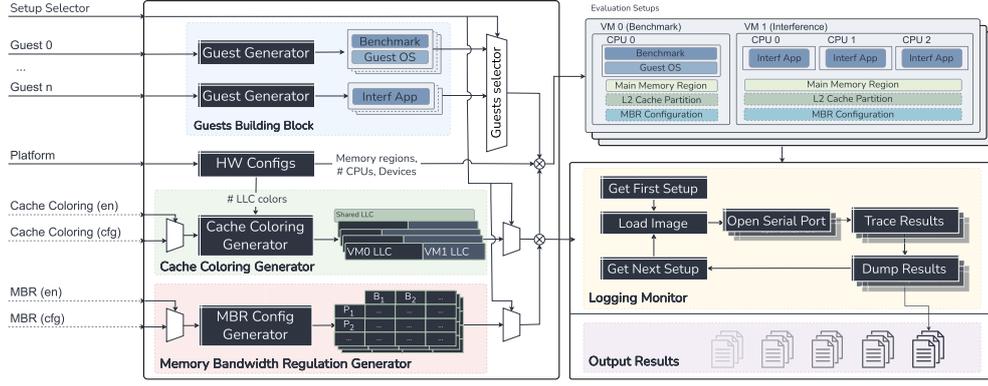
Figure 2 SP-IMPact System Overview.

3 System Overview

In this paper, we introduce SP-IMPact, a framework developed to evaluate and benchmark the performance of MCSs, with a focus on measuring interference and assessing the effectiveness and interaction of interference mitigation techniques. Using a configuration file, users can specify the platform, guest definitions, and test setups. As shown in Figure 2, the framework leverages three key components: (i) the **Guest Generator**, which build guests; (ii) the **Cache Coloring Generator**, which defines cache partitioning configurations; and the (iii) **Memory Bandwidth Regulation Generator**, which generates different configurations of the mechanism. Together, these components ensure precise and consistent performance evaluations. SP-IMPact also includes a **Logging Monitor** for collecting run-time data and an **Output Results** module to handle the gathered information for further analysis.

Guests Generator.   In the context of evaluating system performance, the framework provides support for constructing different types of guests, each tailored for specific benchmarking or interference generation tasks. For the sake of simplicity, this discussion focuses on two primary guest types: a Linux guest and a baremetal guest. However, it should be noted that the framework can be extended to support more and varied guest types as needed.

1. **Linux Benchmark**: Designed to simplify the deployment of various Linux-based workloads, enabling the evaluation of system behavior across diverse scenarios.
2. **Contention Engine**: A baremetal guest tailored to create memory and hardware resource pressure, targeting the LLC and main memory. Key parameters include CPU count, workload sizes, and operation types (reads, writes, or both).

To formalize these configuration options for the baremetal guest, let M represent the total number of CPUs available, L denote the set of possible cache line sizes, W signify the set of workload sizes, and O define the set of operation types, where $O = \{\text{read}, \text{write}, \text{read/write}\}$. The configuration space for the baremetal guest can thus be expressed as:

$$G_{\text{baremetal}} = \{(m, l, w, o) \mid m \in M, l \in L, w \in W, o \in O\} \tag{1}$$

where $G_{\text{baremetal}}$ represents the set of all possible configurations for the baremetal guest. To represent the complete configuration space for all guests, including both baremetal and Linux guests, let G denote the overall set of guest configurations, which is equal to $G_{\text{baremetal}} \cup G_{\text{linux}}$, where G_{linux} represents the set of configurations for the Linux guest.

Cache Coloring Generator. In MCSs, cache partitioning is crucial for reducing LLC interference and ensuring predictable performance. This process, tipically based on cache coloring, aims to divide cache sets into non-overlapping regions for each VM. To assess the impact of different cache partitions, the BaoRTI framework supports the generation of distinct cache color configurations based on the following parameters:

1. The total number of cache sets S, which corresponds to the bit length of the bitmap used to define the color assignments (i.e., each cache set index is represented as a bit in the range $[0, S-1]$);
2. The number of VMs, N, to which distinct cache partitions will be assigned.

The objective of the function is to generate unique configurations of bit masks, dividing the bit range $[0, S-1]$ into N distinct non-overlapping sections. Each section represents a cache partition that can be assigned as a color to the VMs. Given S cache sets and N VMs, the function produces a unique configuration of non-overlapping bit masks for each VM. To facilitate this process, we denote C as the set of all possible $(N-1)$-combinations of bit positions within the range $[0, S-1]$. Each combination in C represents potential VM coloring configurations and can be formally defined as:

$$C = \{(b_1, b_2, \ldots, b_{N-1}) \mid 0 \le b_1 < b_2 < \cdots < b_{N-1} < S\} \tag{2}$$

where each b_i denotes a bit position that separates the partitions for each VM.

To generate the bit masks for each VM, we begin by initializing the starting bit s_0 to 0. For each VM i, where i ranges from 0 to $N-1$, we define the end bit e_i based on the boundary positions: if $i < N-1$, e_i is set to b_i, whereas for the last VM ($i = N-1$), e_i is assigned the total number of cache sets S. With these boundaries established, we compute the bit mask M_i for each VM using the formula:

$$M_i = ((1 \ll (e_i - s_i)) - 1) \ll s_i \tag{3}$$

where "$\ll$" represents a left bit shift operation. This formula creates a mask with $(e_i - s_i)$ bits set to 1, aligned to begin at the position defined by s_i. After calculating the bit mask, we update the starting bit s_i for the next VM by setting it to the current end bit e_i. This iterative process continues until all masks for the VMs are generated, ensuring that each VM receives a unique configuration of non-overlapping cache partitions. After generating a list of bit masks for each VM in the current combination, this configuration is added to a result set colors_assignments if it does not already exist in the set. This ensures that all configurations in that list are unique.

Memory Bandwidth Regulation Generator. In real-time systems, generating distinct memory bandwidth configurations for VMs is essential for ensuring predictable performance and efficient resource utilization. This process, known as memory bandwidth reservation, focuses on creating unique combinations of budget and sampling period for each VM. To evaluate the effects of various bandwidth configurations, the BaoRTI framework supports the generation of distinct MBR configurations based on the following parameters:

1. A list of budgets B available for reservation, where each budget specifies the maximum amount of bandwidth allocated to a VM.
2. A list of sampling periods P, which define the time intervals at which the allocated bandwidth should be monitored.

The objective of the memory bandwidth reservation assignment generation is to produce all the possible combinations of budget and period assignments for each guest. Given the set of all guests, G, where each guest, g, is associated with a set of budgets, B_g, and a set of periods, P_g, the configuration for each guest can be expressed as:

$$MBR_g = \{(B, P) \mid B \in B_g, P \in P_g\} \tag{4}$$

where MBR_g represents the set of all combinations of memory bandwidth reservation configurations for the guest g. Let G be the total number of guests, B be the maximum number of budgets across guests, and P be the maximum number of sampling periods across guests. The time complexity for generating the budget-period combinations for each guest is $O(B \times P)$. Since there are G guests, the overall complexity for processing all guests and generating their combinations can be expressed as $O(G \times B \times P)$.

SP-IMPact features a results logging system designed to capture essential performance and behavioral metrics from the target platform during test execution. The framework collects data from multiple serial ports, each mapped to a specific \ac{VM}, ensuring comprehensive monitoring across the system. The captured metrics include execution time and key micro-architectural events, such as acLLC misses, memory access counts, and cycles spent on the system bus. These metrics are vital for evaluating the impact of shared hardware resources – like the LLC and memory controllers – on workload performance and predictability. This versatile design enables SP-IMPact to support a wide array of benchmarks and metrics tailored to diverse interference scenarios. By correlating data across multiple VMs and configurations, SP-IMPact provides the granularity required to assess the effectiveness of interference mitigation techniques and optimize their configurations.

4 Evaluation

4.1 Evaluation Setup

Hardware Platform.   The experiments were conducted on a Xilinx ZCU104 evaluation board equipped with a Zynq Ultrascale+ ZU7EV SoC. This platform includes a quad-core Arm Cortex-A53 processor, operating at 1.2 GHz. While the SoC supports up to 16 distinct cache colors for cache coloring, the Bao hypervisor constrains this to 8 colors to avoid partitioning the L1 cache. Each core has a dedicated 32 KiB L1 instruction and data cache, along with a unified 1 MiB L2 cache. Additionally, the board is equipped with an Arm Performance Monitoring Unit (PMU), which was leveraged to collect microarchitectural events (such as cache misses and system bus accesses) and profile the benchmark.

Workloads.   For our evaluation, we leveraged the MiBench Automotive and Industrial Control System (AICS) [19] Suite within the critical VM. This subset includes three memory-intensive benchmarks: *qsort*, *susan-c*, and *susan-e*. To generate interference at the memory hierarchy, we deployed a baremetal application that continuously performs read or write operations on a buffer with different sizes. Specifically, buffer sizes include 32 KiB (100% of the L1 cache), 512 KiB (50% of the L2 cache), 1 MiB (100% of the L2 cache), 1.5 MiB (150% of the L2 cache), 2 MiB (200% of the L2 cache), and 4 MiB (400% of the L2 cache).

Setups.   According to Equation 1, we consider the following parameters: $L = 1$ (using only the cache line size matching the cache line size of the target hardware platform), $C = 1$ (using only one CPU configuration, which assigns 3 CPUs to the baremetal VM), $W = 6$ (the total
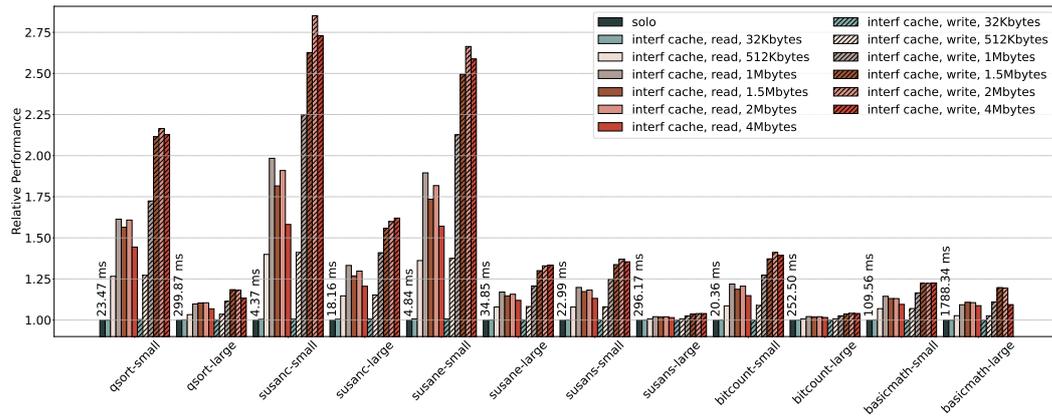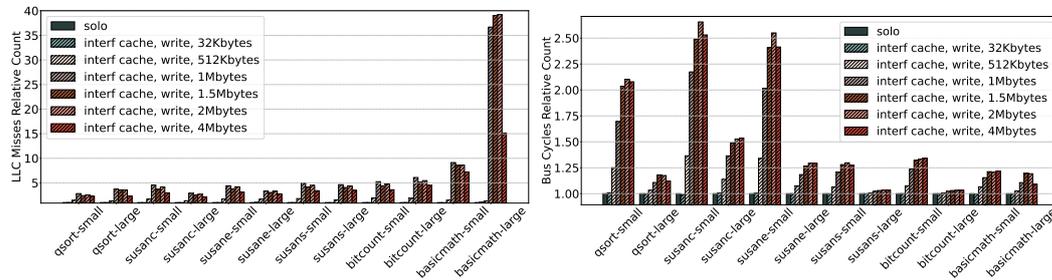
Figure 3 Performance overheads of MiBench automotive benchmark with different workloads.

(a) Interference Impact on LLC.

**(b)** Interference Impact on system bus.

Figure 4 Collected PMU events from MiBench benchmark.

number of workloads used), and $O = 2$ (representing both read and write operations). This results in a total of 12 variations of the baremetal guest. Additionally, for cache coloring, since there are 2 VMs ($N = 2$) and 8 possible cache sets ($S = 8$), there are 8 unique configurations of cache coloring. However, we excluded scenarios in which the Linux VM would be allocated only a single cache color, as such configurations would not provide meaningful performance benefits. Thus, combining these factors results in a total of 84 setups to be tested. For simplicity, we will not consider the configuration of MBR, as introducing it would significantly increase the total number of setups in this evaluation section.

Setup Naming Convention. Setups are named `solo` or `interf_<access>_<buffer_size>`. The `solo` setup serves as the baseline, where a Linux VM runs the MiBench benchmarks without interference. In `interf_<access>_<buffer size>` setups, an additional workload creates cache contention, with `access` specifying `read` or `write` interference type and `buffer_size` indicating the buffer size used. Cache coloring setups add the suffix `<cc_num-colors>`, where `num-colors` denotes the cache colors allocated to the critical VM.

4.2 Interference Impact on Multi-core Platforms

Empirical results presented in Figure 3 indicate that contenetion on shared hardware resources can severely hamper the performance of memory-intensive benchmarks such as *qsort-small*, *susan-c-small*, and *susan-e-small*. The results confirm the theoretical expectations of how the interference buffer size influences resource contention, providing valuable insight into
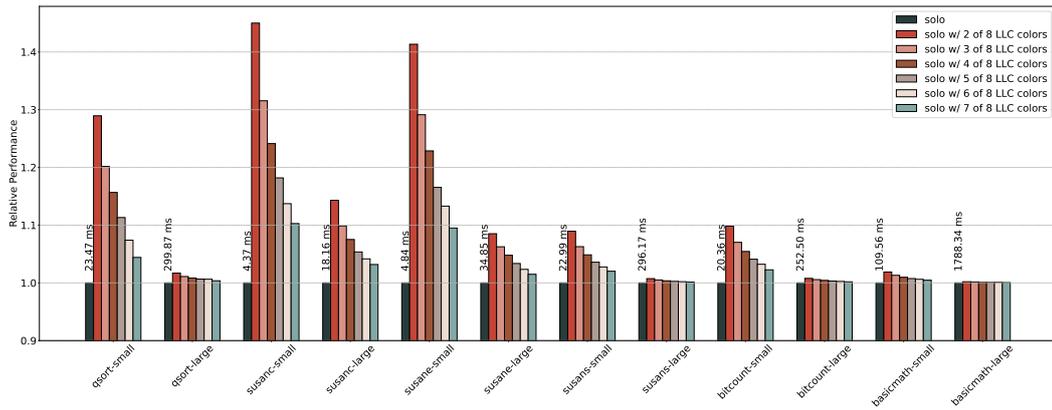
■ **Figure 5** Cache coloring configuration impact on MiBench Benchmark.

the SP-IMPact framework's role in identifying and quantifying such issues. This framework proves essential in assessing how system configurations can exacerbate or mitigate performance bottlenecks in multi-core platforms. The observed interference patterns, where larger buffer sizes lead to increased contention for shared resources, underscore the importance of understanding system-level interactions in MCSs.

While the the empirical resutls where theoretically expected, the empirical evidence reinforces the critical need for tools like SP-IMPact to understand the impact of consolidating different workloads on top of the same hardware platform. Not only does the framework help identify these issues, but it also enables developers to quantify the effects of interference under different configurations, a key insight to drive the deployment of MCSs. By running different workloads with different configurations, developers can collect key performance metrics, such as execution time, cache misses, and bus cycles, which are essential for understanding the severity of the interference. These metrics provide a comprehensive view of how shared resources (e.g. the LLC and the system bus) impact overall system performance. For example, as depicted in Figure 4(a), increasing the buffer size leads to a notable rise in cache misses, which in turn increases the execution time. Specifically, in the `interf_write_1MiB` scenario, the execution time for the *susanc-small* benchmark increases from 4.37 ms to 9.83 ms, demonstrating the growing impact of interference as the buffer size increases.

The role of the SP-IMPact framework in identifying these performance impacts is critical, as it helps pinpoint where interference is most pronounced. Once performance metrics are gathered, the framework allows for in-depth analysis to identify the root causes of performance degradation. For instance, as the interference buffer size grows, portions of the L2 cache become occupied, leading to cache contention and cache evictions. These evictions result in increased memory access time, contributing to further performance slowdowns. The underlying mechanism driving this issue is the competition for cache lines, which causes more frequent evictions and delays in data retrieval. This phenomenon is compounded by the finite size of the cache, which limits the amount of data that can be stored and retrieved quickly. Additionally, Figure 4(b) shows that increasing the buffer size also introduces contention on the system bus, further exacerbating the performance overhead. As workloads compete for access to shared bus resources, the time spent transferring data between the CPU and memory increases, leading to a marked decline in overall system efficiency. These findings underscore the importance of managing resource contention in multi-core environments, where shared hardware resources are increasingly stressed by demanding workloads.
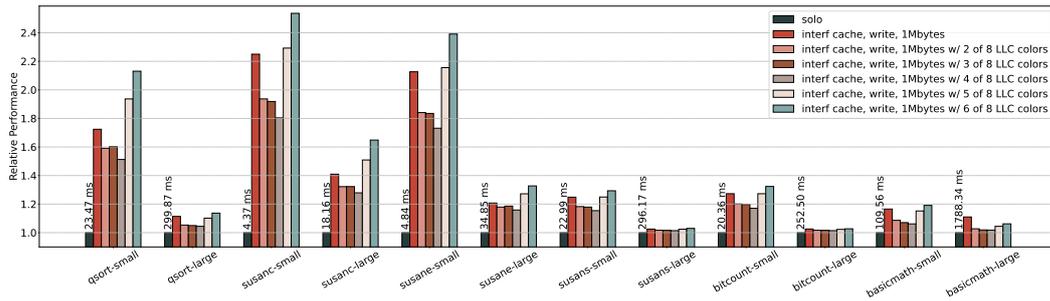
(a) Cache Coloring on 1MiB interference scenario.
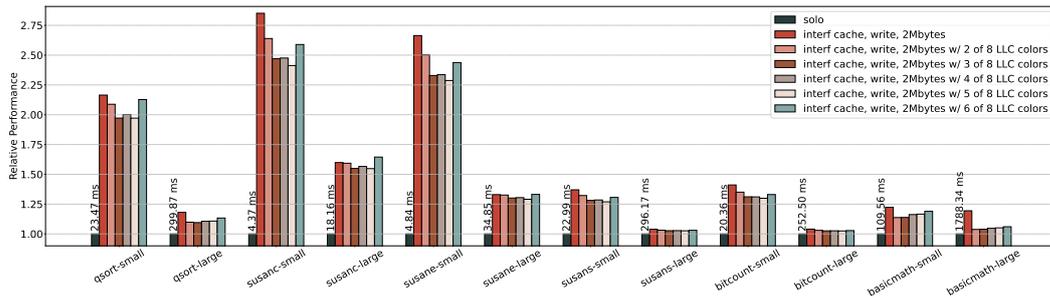
(b) Cache Coloring on 2MiB interference scenario.

Figure 6 Cache Coloring interference mitigation on MiBench Benchmark.

4.3 Interference Mitigation Techniques

Cache Coloring Overhead. Empirical results shows that cache coloring, even in a single-VM environment without interference, can introduce variations in benchmark performance depending on the number of cache colors available for the benchmark's use. This effect is most pronounced in memory-intensive benchmarks, such as *qsort-small*, *susan-c-small*, and *susan-e-small*, where reduced cache availability leads to significant slowdowns due to increased cache misses. With only two of the eight cache colors available, the execution time of benchmarks like *susanc-small* and *susane-small* increases by 1.45x and 1.41x, respectively, due to reduced cache allocation. As the number of colors increases, performance gradually approaches the baseline. With five cache colors available, benchmarks generally perform closer to their solo execution times. For example, *susanc-small* and *susane-small* improve to slowdowns of 1.18x and 1.17x, respectively. Benchmarks with lower memory intensity, such as *qsort-large*, *basicmath-small*, and *basicmath-large*, show minimal to no performance degradation across various cache coloring scenarios. With only two colors, *basicmath-large* shows no measurable slowdown across all coloring configurations. Similarly, *qsort-large* and *basicmath-small* maintain near-baseline performance, with minimal slowdowns of 1.02x.

Interference Mitigation. The SP-IMPact framework plays a key role in assisting developers with mitigating memory contention issues during the development of MCSs. After identifying bottlenecks caused by shared hardware resources, developers can leverage the framework to simulate different scenarios and adjust system configurations accordingly. For example, cache coloring can be leveraged to minimize interference. Figures 6(a) and 6(b) show the impact of different cache coloring configurations on memory-intensive benchmarks, such as *qsort-small* and *susanc-small*, when consolidated with the interference baremetal VM

(e.g, running the `interf_write_1MiB` and the `interf_write_2MiB` scenarios). Applying 2 cache colors reduces execution time overhead from 1.72x and 2.25x (no coloring) to 1.59x and 1.94x, respectively. Further improvements are observed with 4 cache colors, reducing interference to 1.51x and 1.80x for *qsort-small* and *susanc-small*. While cache coloring is effective for memory-intensive workloads, developers should consider diminishing returns beyond 4 colors, where performance gains decrease, and system-level contention (especially on the bus) may increase. The SP-IMPact framework helps identify these diminishing returns, allowing developers to select the most optimal configuration. For less memory-intensive benchmarks like *qsort-large* and *basicmath-large*, cache coloring has minimal impact, enabling developers to focus on other optimization techniques for such workloads.

5   Discussion and Future Directions

In this section, we discuss some of the open issues and potential research directions to understand and improve the impact of interference in multi-core platforms.

Workload Interference Analysis in Mixed-Criticality Systems.   MCSs face significant challenges when consolidating workloads with varying criticality levels and timing requirements on the same hardware platform. As workloads compete for shared resources such as caches, memory buses, and system interconnects, predicting interactions and maintaining reliable performance for critical tasks remains a complex problem. While the framework presented in this work enables profiling and quantifying interference effects under diverse scenarios, further research is needed to explore how workload characteristics - such as memory access patterns and computation intensity (e.g., memory access rate - can be modeled more accurately. Moreover, one important limitation of the current evaluation is that it primarily focuses on interference effects in terms of LLC misses and bus cycles; while the SP-IMPact framework allows the analysis of the contention in these components, the lack of state-of-the-art benchmarks to evaluate them limits their inclusion in this study.

Interference Mitigation Techniques.   Configuring interference mitigation mechanisms, such as cache coloring, presents its own set of challenges. Each possible configuration (e.g. the number of cache colors or memory bandwidth regulation configuration) can produce different impacts on performance and contention levels. Selecting the optimal configuration requires an understanding of both the workload's memory demands and the system's architectural characteristics. The framework aids in this process by providing in-depth evaluations of various interference mitigation techniques configurations and their impact on interference on multi-core platforms. Additionally, exploring the interaction between the proposed framework and high-performance hardware features, such as quality-of-service (QoS) mechanisms that control on-chip and DRAM traffic, would be valuable, as presented in [45].

Future Work.   Building on the insights from this study, several extensions to the current framework are planned: an immediate enhancement of SP-IMPact involves the development of a hypervisor-level performance monitor to enable VM profiling without requiring guest instrumentation. Currently, the framework simplifies the generation of Linux-based benchmarks and baremetal VMs; next-steps focus on extending this capability to other OSes (e.g., FreeRTOS and Zephyr), enabling a comprehensive interference analysis and mitigation evaluations across a wider range of workloads and system configurations. In the long term, we aim

to integrate AI-driven techniques for adaptive interference management, which may enable the optimization of interference patterns, allowing the simulation of worst-case scenarios and providing more accurate performance assessments under challenging conditions.

6   Related Work

Interference analysis in multi-core systems has been approached using two primary frameworks: (i) generic task models and (ii) phased execution models. Each approach has its strengths but also limitations when it comes to capturing the complexities of modern high-complexity MCSs, especially those that include hypervisors. In the following, we provide a brief overview of these two categories and highlight the key research efforts within each.

Generic Task Models.   Generic task models provide abstractions for task behaviors on multi-core systems by focusing on resource usage patterns such as memory access, computation, and synchronization. These models typically focus on quantifying contention between tasks based on broad assumptions and often omit platform-specific characteristics. Generic task models can be divided into two main categories: (i) memory bus contention and [10, 2, 42, 24, 23, 14], (ii) main memory contention [49, 25, 20].

Phased Execution Models.   While generic task models provide broad abstractions, they are limited in their ability to model complex, dynamic interference patterns that arise in multi-core systems. This limitation led to the development of phased execution models, which break down task execution into distinct phases. Phased execution models offer more detailed representations of how tasks interact with shared resources during different execution phases. To address the issues left by generic task models, phased execution models are divided in: (i) offline scheduling-based approaches [44, 38, 5], (ii) shared resource contention-based approaches [32, 6, 3], and (iii) memory-centric scheduling-based approaches [47, 43].

Limitations of Existing Approaches.   While generic task and phased execution models help understand some aspects of interference, they fall short in high-complexity MCSs, especially those with hypervisors. These models focus on limited contention sources, like LLC and system buses, but omit others such as IOMMUs or interrupt controllers, which are crucial in real hardware. Moreover, they overlook the combined effects of multiple mitigation techniques. SP-IMPact fills these gaps by enabling the assessment of interference in hypervisor-based systems and evaluating the effectiveness and interactions of interference mitigation techniques. Unlike analytical models, SP-IMPact simplifies configuration by allowing real-time experimentation on actual hardware, making it easier to identify bottlenecks and test configurations in a flexible way. The framework was tested with Bao hypervisor and currently supports its configuration interface to define VMs' configurations and hardware partitioning, but it can be extended to support other hypervisors in the future.

7   Conclusion

In this paper, we propose the design, implementation, and evaluation of SP-IMPact, a framework for analyzing the impact of multi-core contention, and the impact of interference mitigation techniques. This framework facilitates the automated deployment, configuration, and data collection of multiple setups to quantify platform-level contention and evaluate the impact of interference mitigation techniques, such as cache coloring. Using the Zynq Ultrascale+ platform, our evaluation demonstrated how the framework enables precise

analysis of interference effects under various workload configurations, providing critical insights for deploying consolidated multi-core systems. We believe that this framework lays a solid foundation for future extensions, including AI-driven interference management, expanded workload patterns, and support for additional platforms.

References

1   Jaume Abella, Carles Hernandez, Eduardo Quiñones, Francisco J. Cazorla, Philippa Ryan Conmy, Mikel Azkarate-askasua, Jon Perez, Enrico Mezzetti, and Tullio Vardanega. WCET analysis methods: Pitfalls and challenges on their trustworthiness. In *10th IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 1–10, 2015.

2   Alexandru Andrei, Zebo Peng, Jakob Rosen, and Petru Eles. Bus Access Optimization for Predictable Implementation of Real-Time Applications on Multiprocessor Systems-on-Chip . In *IEEE 34th Real-Time Systems Symposium*, pages 49–60, 2007. `doi:10.1109/RTSS.2007.24`.

3   Jatin Arora, Cláudio Maia, Syed Aftab Rashid, Geoffrey Nelissen, and Eduardo Tovar. Bus-contention aware wcrt analysis for the 3-phase task model considering a work-conserving bus arbitration scheme. *Journal of Systems Architecture*, 122:102345, 2022. `doi:10.1016/J.SYSARC.2021.102345`.

4   Michael Bechtel and Heechul Yun. Denial-of-Service Attacks on Shared Cache in Multicore: Analysis and Prevention . In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 357–367, 2019.

5   Matthias Becker, Dakshina Dasari, Borislav Nicolic, Benny Akesson, Vincent Nelis, and Thomas Nolte. Contention-Free Execution of Automotive Applications on a Clustered Many-Core Platform . In *28th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 14–24, 2016.

6   Daniel Casini, Alessandro Biondi, Geoffrey Nelissen, and Giorgio Buttazzo. A Holistic Memory Contention Analysis for Parallel Real-Time Tasks under Partitioned Scheduling . In *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 239–252, 2020. `doi:10.1109/RTAS48715.2020.000-3`.

7   Francisco J. Cazorla, Leonidas Kosmidis, Enrico Mezzetti, Carles Hernandez, Jaume Abella, and Tullio Vardanega. Probabilistic worst-case timing analysis: Taxonomy and comprehensive survey. *ACM Comput. Surv.*, 52, 2019. `doi:10.1145/3301283`.

8   Francisco J. Cazorla, Eduardo Quiñones, Tullio Vardanega, Liliana Cucu, Benoit Triquet, Guillem Bernat, Emery Berger, Jaume Abella, Franck Wartel, Michael Houston, Luca Santinelli, Leonidas Kosmidis, Code Lo, and Dorin Maxim. PROARTIS: Probabilistically Analyzable Real-Time Systems. *ACM Trans. Embed. Comput. Syst.*, 12(2s), 2013. `doi:10.1145/2465787.2465796`.

9   Jon Perez Cerrolaza, Roman Obermaisser, Jaume Abella, Francisco J. Cazorla, Kim Grüttner, Irune Agirre, Hamidreza Ahmadian, and Imanol Allende. Multi-core Devices for Safety-critical Systems: A Survey. *ACM Comput. Surv.*, 53(4), 2020. `doi:10.1145/3398665`.

10 Sudipta Chattopadhyay, Abhik Roychoudhury, and Tulika Mitra. Modeling shared cache and bus in multi-cores for timing analysis. In *Proceedings of the 13th international workshop on software & compilers for embedded systems*, pages 1–10, 2010.

11 Diogo Costa, Luca Cuomo, Daniel Oliveira, Ida Maria Savino, Bruno Morelli, José Martins, Fabrizio Tronci, Alessandro Biasci, and Sandro Pinto. IRQ Coloring: Mitigating Interrupt-Generated Interference on ARM Multicore Platforms. In *Fourth Workshop on Next Generation Real-Time Embedded Systems (NG-RES)*, volume 108, pages 2:1–2:13, 2023. `doi:10.4230/OASICS.NG-RES.2023.2`.

12 Diogo Costa, Luca Cuomo, Daniel Oliveira, Ida Maria Savino, Bruno Morelli, José Martins, Alessandro Biasci, and Sandro Pinto. IRQ Coloring and the Subtle Art of Mitigating Interrupt-Generated Interference. In *IEEE 29th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 47–56, 2023. `doi:10.1109/RTCSA58653.2023.00015`.

13 Dakshina Dasari, Benny Akesson, Vincent Nelis, Muhammad Ali Awan, and Stefan M Petters. Identifying the sources of unpredictability in cots-based multicore systems. In *8th IEEE international symposium on industrial embedded systems (SIES)*, pages 39–48, 2013.

14 Robert I Davis, Sebastian Altmeyer, Leandro S Indrusiak, Claire Maiza, Vincent Nelis, and Jan Reineke. An extensible framework for multicore response time analysis. *Real-Time Systems*, 54:607–661, 2018. `doi:10.1007/S11241-017-9285-4`.

15 Farzad Farshchi, Qijing Huang, and Heechul Yun. BRU: Bandwidth Regulation Unit for Real-Time Multicore Processors. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 364–375, 2020. `doi:10.1109/RTAS48715.2020.00011`.

16 Abel Gordon, Nadav Amit, Nadav Har'El, Muli Ben-Yehuda, Alex Landau, Assaf Schuster, and Dan Tsafrir. Eli: bare-metal performance for i/o virtualization. *SIGPLAN Not.*, 47(4):411–422, 2012. `doi:10.1145/2150976.2151020`.

17 Giovani Gracioli, Ahmed Alhammad, Renato Mancuso, Antônio Augusto Fröhlich, and Rodolfo Pellizzoni. A Survey on Cache Management Mechanisms for Real-Time Embedded Systems. *ACM Comput. Surv.*, 48(2), 2015. `doi:10.1145/2830555`.

18 Giovani Gracioli, Rohan Tabish, Renato Mancuso, Reza Mirosanlou, Rodolfo Pellizzoni, and Marco Caccamo. Designing Mixed Criticality Applications on Modern Heterogeneous MPSoC Platforms. In *31st Euromicro Conference on Real-Time Systems (ECRTS)*, volume 133, pages 27:1–27:25, 2019. `doi:10.4230/LIPICS.ECRTS.2019.27`.

19 M.R. Guthaus, J.S. Ringenberg, D. Ernst, T.M. Austin, T. Mudge, and R.B. Brown. Mibench: A free, commercially representative embedded benchmark suite. In *Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization.*, pages 3–14, 2001.

20 Mohamed Hassan and Rodolfo Pellizzoni. Bounding DRAM Interference in COTS Heterogeneous MPSoCs for Mixed Criticality Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2323–2336, 2018. `doi:10.1109/TCAD.2018.2857379`.

21 Thomas A. Henzinger and Joseph Sifakis. "The Embedded Systems Design Challenge". In Jayadev Misra, Tobias Nipkow, and Emil Sekerinski, editors, *FM 2006: Formal Methods*, pages 1–15, 2006. `doi:10.1007/11813040_1`.

22 Joo-Young Hwang, Sang-Bum Suh, Sung-Kwan Heo, Chan-Ju Park, Jae-Min Ryu, Seong-Yeol Park, and Chul-Ryun Kim. Xen on arm: System virtualization using xen hypervisor for arm-based secure mobile phones. In *5th IEEE Consumer Communications and Networking Conference*, pages 257–261, 2008. `doi:10.1109/CCNC08.2007.64`.

23 Michael Jacobs, Sebastian Hahn, and Sebastian Hack. A Framework for the Derivation of WCET Analyses for Multi-core Processors. In *28th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 141–151, 2016. `doi:10.1109/ECRTS.2016.19`.

24 Timon Kelter, Heiko Falk, Peter Marwedel, Sudipta Chattopadhyay, and Abhik Roychoudhury. Bus-Aware Multicore WCET Analysis through TDMA Offset Bounds. In *23rd Euromicro Conference on Real-Time Systems*, pages 3–12, 2011. `doi:10.1109/ECRTS.2011.9`.

25 Hyoseung Kim, Dionisio De Niz, Björn Andersson, Mark Klein, Onur Mutlu, and Ragunathan Rajkumar. Bounding memory interference delay in COTS-based multi-core systems. In *IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 145–154, 2014. `doi:10.1109/RTAS.2014.6925998`.

26 Hyoseung Kim and Ragunathan (Raj) Rajkumar. Predictable Shared Cache Management for Multi-Core Real-Time Virtualization. *ACM Trans. Embed. Comput. Syst.*, 17(1), 2017. `doi:10.1145/3092946`.

27 Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, et al. seL4: Formal verification of an OS kernel. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 207–220, 2009.

28 Tomasz Kloda, Marco Solieri, Renato Mancuso, Nicola Capodieci, Paolo Valente, and Marko Bertogna. Deterministic Memory Hierarchy and Virtualization for Modern Multi-Core Embedded Systems. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 1–14, 2019. `doi:10.1109/RTAS.2019.00009`.

29   Ondrej Kotaba, Jan Nowotsch, Michael Paulitsch, Stefan M Petters, and Henrik Theiling. Multicore in real-time systems–temporal isolation challenges due to shared resources. In *16th Design, Automation & Test in Europe Conference and Exhibition*, 2013.

30   Andreas Löfwenmark and Simin Nadjm-Tehrani. Understanding Shared Memory Bank Access Interference in Multi-Core Avionics. In *16th International Workshop on Worst-Case Execution Time Analysis (WCET)*, volume 55 of *Open Access Series in Informatics (OASIcs)*, pages 12:1–12:11, 2016. `doi:10.4230/OASICS.WCET.2016.12`.

31   Tamara Lugo, Santiago Lozano, Javier Fernández, and Jesus Carretero. A survey of techniques for reducing interference in real-time applications on multicore platforms. *IEEE Access*, 10:21853–21882, 2022. `doi:10.1109/ACCESS.2022.3151891`.

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

33   Renato Mancuso, Roman Dudko, Emiliano Betti, Marco Cesati, Marco Caccamo, and Rodolfo Pellizzoni. Real-time cache management framework for multi-core architectures. In *IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 45–54, 2013. `doi:10.1109/RTAS.2013.6531078`.

34   José Martins, Adriano Tavares, Marco Solieri, Marko Bertogna, and Sandro Pinto. Bao: A Lightweight Static Partitioning Hypervisor for Modern Multi-Core Embedded Systems. In *Workshop on Next Generation Real-Time Embedded Systems (NG-RES)*, volume 77, pages 3:1–3:14, 2020. `doi:10.4230/OASICS.NG-RES.2020.3`.

35   José Martins and Sandro Pinto. Shedding Light on Static Partitioning Hypervisors for Arm-based Mixed-Criticality Systems. In *IEEE 29th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 40–53, 2023. `doi:10.1109/RTAS58335.2023.00011`.

36   Paolo Modica, Alessandro Biondi, Giorgio Buttazzo, and Anup Patel. Supporting temporal and spatial isolation in a hypervisor for ARM multicore platforms. In *IEEE International Conference on Industrial Technology (ICIT)*, pages 1651–1657, 2018. `doi:10.1109/ICIT.2018.8352429`.

37   Afonso Oliveira, Gonçalo Moreira, Diogo Costa, Sandro Pinto, and Tiago Gomes. IA&AI: Interference Analysis in Multi-core Embedded AI Systems. In *Data Science and Artificial Intelligence*, pages 181–193, 2025.

38   Claire Pagetti, Julien Forget, Heiko Falk, Dominic Oehlert, and Arno Luppold. Automated generation of time-predictable executables on multicore. In *Proceedings of the 26th International Conference on Real-Time Networks and Systems*, RTNS '18, pages 104–113. Association for Computing Machinery, 2018. `doi:10.1145/3273905.3273907`.

39   Rob Palin, David Ward, Ibrahim Habli, and Roger Rivett. Iso 26262 safety cases: Compliance and assurance. In *6th IET International Conference on System Safety*, pages 1–6, 2011.

40   Sandro Pinto, Jorge Pereira, Tiago Gomes, Adriano Tavares, and Jorge Cabral. LTZVisor: TrustZone is the Key. In *29th Euromicro Conference on Real-Time Systems (ECRTS)*, volume 76 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 4:1–4:22, 2017. `doi:10.4230/LIPICS.ECRTS.2017.4`.

41   Ralf Ramsauer, Jan Kiszka, Daniel Lohmann, and Wolfgang Mauerer. Look Mum, no VM Exits! (Almost). *CoRR*, 2017.

42   Andreas Schranzhofer, Jian-Jia Chen, and Lothar Thiele. Timing analysis for TDMA arbitration in resource sharing systems. In *16th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 215–224, 2010. `doi:10.1109/RTAS.2010.24`.

43   Gero Schwäricke, Tomasz Kloda, Giovani Gracioli, Marko Bertogna, and Marco Caccamo. Fixed-Priority Memory-Centric Scheduler for COTS-Based Multiprocessors. In *32nd Euromicro Conference on Real-Time Systems (ECRTS)*, volume 165 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 1:1–1:24, 2020. `doi:10.4230/LIPICS.ECRTS.2020.1`.

44   Ikram Senoussaoui, Houssam-Eddine Zahaf, Giuseppe Lipari, and Kamel Mohamed Benhaoua. Contention-free scheduling of PREM tasks on partitioned multicore platforms. In *IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8, 2022. `doi:10.1109/ETFA52439.2022.9921531`.

45 Alejandro Serrano Cases, Juan M Reina, Jaume Abella Ferrer, Enrico Mezzetti, and Francisco Javier Cazorla Almeida. Leveraging hardware QoS to control contention in the Xilinx Zynq UltraScale+ MPSoC. In *33rd Euromicro Conference on Real-Time Systems (ECRTS)*, volume 196, pages 3–1, 2021.

46 Theo Ungerer, Francisco Cazorla, Pascal Sainrat, Guillem Bernat, Zlatko Petrov, Christine Rochange, Eduardo Quiñones, Mike Gerdes, Marco Paolieri, Julian Wolf, Hugues Cassé, Sascha Uhrig, Irakli Guliashvili, Michael Houston, Floria Kluge, Stefan Metzlaff, and Jorg Mische. Merasa: Multicore Execution of Hard Real-Time Applications Supporting Analyzability. *IEEE Micro*, 30(5):66–75, 2010. `doi:10.1109/MM.2010.78`.

47 Gang Yao, Rodolfo Pellizzoni, Stanley Bak, Heechul Yun, and Marco Caccamo. Global Real-Time Memory-Centric Scheduling for Multicore Systems. *IEEE Transactions on Computers*, 65(9):2739–2751, 2016. `doi:10.1109/TC.2015.2500572`.

48 Heechul Yun, Renato Mancuso, Zheng-Pei Wu, and Rodolfo Pellizzoni. PALLOC: DRAM bank-aware memory allocator for performance isolation on multicore platforms. In *IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 155–166, 2014. `doi:10.1109/RTAS.2014.6925999`.

49 Heechul Yun, Rodolfo Pellizzon, and Prathap Kumar Valsan. Parallelism-aware memory interference delay analysis for COTS multicore systems. In *27th Euromicro Conference on Real-Time Systems*, pages 184–195, 2015. `doi:10.1109/ECRTS.2015.24`.

50 Heechul Yun, Gang Yao, Rodolfo Pellizzoni, Marco Caccamo, and Lui Sha. Memory Access Control in Multiprocessor for Real-Time Systems with Mixed Criticality . In *24th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 299–308, 2012. `doi:10.1109/ECRTS.2012.32`.

51 Heechul Yun, Gang Yao, Rodolfo Pellizzoni, Marco Caccamo, and Lui Sha. MemGuard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms. In *IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 55–64, 2013. `doi:10.1109/RTAS.2013.6531079`.

52 Matteo Zini, Giorgiomaria Cicero, Daniel Casini, and Alessandro Biondi. Profiling and controlling I/O-related memory contention in COTS heterogeneous platforms. *Software: Practice and Experience*, 52(5):1095–1113, 2022. `doi:10.1002/SPE.3053`.