Analyzing Arm's MPAM From the Perspective of Time Predictability

Matteo Zini, Daniel Casini, *Member, IEEE*, and Alessandro Biondi, *Member, IEEE*

Abstract—With heterogeneous multi-core platforms being crucial to execute the highly demanding workloads of modern applications, memory-access predictability remains a key issue for the system's safety. Many solutions have been proposed over the years, but none has been applied on a large scale. Nowadays, we are in front of an unprecedented opportunity to have an impact on commercial platforms: the Memory System Resource Partitioning and Monitoring (MPAM) specification by Arm, which describes different memory-access regulation mechanisms, presenting a valuable industrial attempt to address this issue. However, several points of the specification are described at a high level only, leaving plenty of room for interpretation to hardware manufacturers. This paper takes a close look at the memory-access regulation mechanisms in the MPAM specification and provides some detailed instantiations of such mechanisms. A fine-grained memory contention analysis is presented for each of them to finally enable a comparison of their worst-case performance.

Index Terms—Real-time systems, memory-access predictability, memory-contention analysis, predictability

1 Introduction

Memory access predictability is a key challenge for modern heterogeneous platforms. Indeed, with the advent of always more demanding workloads such as those required in autonomous driving applications, the usage of powerful computing platforms equipped with multiple processing cores and hardware accelerators is becoming a de-facto requirement for many systems [1]. On the other hand, many of those applications are grandly critical, thus calling for a high degree of predictability.

The real-time systems research community is studying this problem since almost a decade, proposing many clever solutions to improve the memory access predictability of different types of memories and shared buses, both when accessed by CPU cores [2], [3], [4], I/O devices [5], [6], [7], and hardware accelerators [8], [9], [10]. These solutions include the usage of performance counters to keep track of the number of memory accesses [2], the development of memory-aware execution models [11], [12], and the design and implementation of custom components as predictable buses [7], [13] and memory controllers [14], [15], [16].

- *Matteo Zini is with the TeCIP Institute, Scuola Superiore Sant'Anna, 56124 Pisa, Italy. E-mail: matteo.zini@santannapisa.it.*
- *Daniel Casini and Alessandro Biondi are with the TeCIP Institute, Scuola Superiore Sant'Anna, 56124 Pisa, Italy, and also with the Department of Excellence in Robotics & AI, Scuola Superiore Sant'Anna, 56124 Pisa, Italy. E-mail: {daniel.casini, alessandro.biondi}@santannapisa.it.*

Manuscript received 1 February 2022; revised 26 July 2022; accepted 23 August 2022. Date of publication 29 August 2022; date of current version 13 December 2022.
This work was supported in part by Huawei and in part by the Italian Ministry of University and Research (MIUR) through the SPHERE Project funded within the PRIN-2017 framework under Grant 93008800505.
(Corresponding author: Matteo Zini.)
Recommended for acceptance by B. Childers.
This article has supplementary downloadable material available at https://doi.org/10.1109/TC.2022.3202720, provided by the authors.
Digital Object Identifier no. 10.1109/TC.2022.3202720

While these efforts greatly helped understand and mitigate many memory-related problems for predictability, they suffer from a substantial issue: the lack of general applicability and built-in support from hardware vendors. Indeed, providing predictable access to shared memories and buses has been largely under-considered by chip vendors, but they are the only ones that can ultimately solve the problem on a large scale by providing predictable hardware support.

Very recently, a unique opportunity has been offered from the chip vendors landscape: The Memory System Resource Partitioning and Monitoring (MPAM) [17] specification by Arm. After at most a decade in which this problem is well-known and continuously studied in the research community, Arm acted from the industry side by providing a specification with several guidelines to implement memory-access regulation mechanisms, thus giving an unprecedented opportunity to solve the problem on billions of Arm-based platforms, while providing the needed long-term support that is difficult to reach with custom solutions.

Interestingly, to date, the number of platforms implementing the MPAM specification is extremely limited, so there is hopefully still room to provide recommendations to chip manufacturers implementing MPAM to maximize execution predictability. Indeed, the MPAM specification provides quite a high-level description of the regulation mechanisms, leaving plenty of room for interpretations to whoever wants to implement it. Slightly different behaviors of such mechanisms may lead to drastically different worst-case timing performance, and therefore each detail needs to be carefully addressed at design time.

When "measuring" the predictability of a system component, a key metric is the accuracy that we can achieve in deriving analytical bounds. Indeed, while other kinds of performance indicators, such as those that can be collected by running the system in simulation or implementing a prototype, are useful and complementary to the analytical characterization of the worst-case latency that can be experienced

in accessing a memory-system component, the possibility of deriving accurate latency bounds really makes a system more or less amenable to certification.

Contribution. In the light of these facts, this paper proposes an analysis-driven evaluation of a set of MPAM mechanisms. First, we take a close look at the MPAM specification, and in particular to those mechanisms aimed at achieving predictability at the level of the Double Data Rate Synchronous Dynamic Access Memory (DDR SDRAM) memory. While doing this, we highlight each step in which the specification is missing relevant information to unambiguously specify the behavior of a mechanism. We then propose a number of different lower-level specifications for the studied mechanisms, providing all the required elements to enable real-time analysis. This allows building a memory-contention analysis based on an optimization problem. Thanks to the modularity of the approach, mechanism-specific constraints are derived and independently studied while keeping the overall structure of the analysis framework unaltered. Finally, an analysis-driven comparison of the studied mechanisms is presented, reaching conclusions on their effectiveness in enabling time predictability.

2 Essential Background on DDR DRAMs

A DRAM memory device consists of a set of memory chips organized into *ranks*. Each rank is further divided into multiple *banks*. A bank consists of a matrix of memory cells, and it is provided with a *row buffer*, which behaves similarly as a cache for the memory bank: multiple consecutive accesses to the same row result in smaller lateness. To access a specific memory cell, the content of its *row* in the matrix needs to be copied into the row buffer. Since the row buffer can store at most one row at a time, this operation may involve copying back the row currently stored in the row buffer. A memory access targeting a row contained in the row buffer is said to be a *row-hit*; otherwise, it is said to be a *row-conflict*.

When a row-hit occurs, the memory access (either a read or a write) can be performed by means of the `CAS` (Column Access Strobe) *DRAM command*.

In case of a row-conflict, three commands need to be issued in sequence. The first one is the `PRE` (PREcharge) command, which copies back the content currently stored row-buffer in the corresponding DRAM row; the second one is the `ACT` (ACTivate) command, which copies the data from the target DRAM row to the row buffer; the last one is the `CAS` command, needed to actually perform the operation.

The access to the DRAM memory is orchestrated by a *memory controller* (MC), which collects the requests incoming from the processing elements and routes DRAM commands to the DRAM chips via the buses that connect them. The memory controller and the DRAM chips are interconnected by means of two buses, one for data and one for commands. Memory requests directed to different banks can be handled in parallel if no contention occurs in the two buses. The memory controller is responsible to implement the scheduling logic for memory requests. The timing constraints to be fulfilled between consecutive transmissions of commands and data on the DRAM buses are regulated by the JEDEC (Joint Electron Device Engineering Council) standard for DRAM memories, and they can be coarsely classified into *intra-bank* constraints, i.e., related to commands and data targeting the same bank, and *inter-bank* constraints, i.e., related to different banks. More details on the JEDEC timing constraints can be found in the appendix, available online.

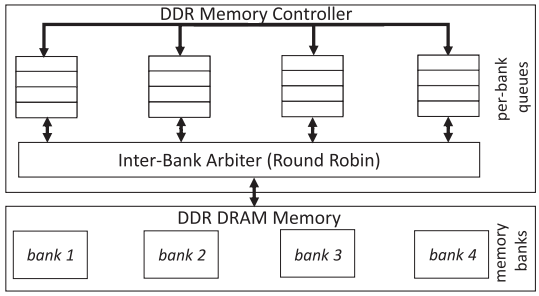

Fig. 1. Illustration of the MC structure.

3 System Model

The system comprises a set $\mathcal{P} = \{p_1, \ldots, p_P\}$ of P identical processing elements (i.e., cores or other bus masters). Processing elements implement the *out-of-order* execution paradigm and access a globally-shared memory by means of an interconnect and a memory controller.

All the processing elements share a global DRAM memory $\mathcal{G}$. A crossbar switch allows conflict-free point-to-point communication between each processing element and the DRAM memory controller. We consider a single-channel [18] and single-rank DRAM subsystem, where the DRAM memory is composed of one rank divided into a set $\mathcal{B}$ of N_B banks.

Following the MPAM specifications, different sources of memory transactions are identified by *partition identifiers* (PARTIDs). This work considers a set $\mathcal{R}$ of PARTIDs, where each individual PARTID $\rho_i \in \mathcal{R}$ is used to determine the partitioning of memory resources. As extensively discussed later in Section 4, a PARTID can be, for example, assigned to a task or a virtual machine.

Memory Controller Model. In this paper, we build on the memory controller (MC) model used in previous work [19], which is inspired to realistic designs used both by the industry [20], [21], [22] and academia [18], [23], [24]. Its high-level structure is shown in Fig. 1. For each bank $b_y \in \mathcal{B}$, the MC provides a queue to accommodate read requests. Each of these queues exposes at most one request that is ready for inter-bank arbitration. An inter-bank scheduler selects the request to be forwarded to the DRAM memory among those exposed by the intra-bank queues. Requests are in general composed of multiple commands (i.e., the `PRE`, `ACT`, and `CAS` presented in Section 2). The inter-bank scheduler selects a new request to be scheduled when the previous one is completed (if any: otherwise the new request is served upon arrival), i.e., when all its commands have been issued, and the corresponding JEDEC constraints are satisfied. Write requests are enqueued separately and served in batches [18], [19], [24]. This technique is called *write batching* and it is typically used in COTS controllers [21] to improve the throughput (in architectures where writes do not stall the processing pipeline and can be processed asynchronously) by giving precedence to reads over writes while avoiding starvation. This memory controller model is

used next as a baseline MC in our paper. As this work studies different alternatives to implement the MPAM specifications, when required, the MC behavior defined in this section is either enriched or revised in some of its parts, but keeping the overall structure unaltered. The behavior of the baseline MC is formalized by means of a set of rules, which are classified into three categories: **(i)** intra-queue arbitration rules, **(ii)** inter-queue arbitration rules, and **(iii)** write handling rules.

Intra-queue arbitration rules:

MC1. *FCFS:* Requests in each per-bank queue are organized with a *first-come-first-served* (FCFS) order. FCFS prioritizes older requests over newer requests.

MC2. *FR:* In each per-bank queue, row-hits are privileged over row-conflicts, i.e., the requests causing row-hits can pass ahead of older requests causing row-conflicts. The number of requests that can pass ahead of any other request, with respect to the FCFS order, is limited by the constant N_{thr}.

MC3. *FR-FCFS:* The overall scheduling policy applies, in order, first rule **MC2** and then rule **MC1**.

Inter-queue arbitration rules:

MC4. *Round Robin*. The inter-bank arbitration employs the round-robin scheduling algorithm. At most one request is served at each round-robin turn. No inter-bank reordering is allowed.

Write handling rules:

MC5. *Write Batching*. The memory controller enqueues write requests and serves at least N_{wb} of them in a batch as soon as W_{thr} write requests are enqueued (referred to as *watermarking threshold* [24]). The write batch stops after N_{wb} writes if there is at least one pending read, and at least a read request is served afterwards.

For the write batching, we use the same assumptions as in prior work [18], [19], [24]. Namely, in our model: **(i)** the write buffer has a size Q_{write}, **(ii)** $W_{\text{thr}} \geq N_{\text{wb}}$, so that when the watermark threshold is overtaken there are always at least N_{wb} write requests to form a batch, **(iii)** $Q_{\text{write}} - N_{\text{wb}} < W_{\text{thr}}$, i.e., after issuing a batch the overall number of writes fall below the watermarking threshold and, **(iv)** the write buffer is large enough to allow it never becoming full. Furthermore, it is worth highlighting that if a read request targets data for which there exists a pending request in the write buffer, then it is served from the queue without accessing the DRAM. This is required to guarantee data causality.

Definitions. We conclude the section by providing some useful definitions. A request r_x issued by ρ_i suffers *interference* from another request r_y when r_y is served while r_x is pending. Given a request r_y, the interference it can cause to another request r_x is either categorized as *intra-bank* or *inter-bank* interference. Namely, r_y causes *intra-bank* interference to r_x if r_y and r_x target the *same bank* and r_x suffers interference from r_y. Instead, r_y causes *inter-bank* interference to r_x if r_x suffers interference from r_y and they target *different banks*.

The inter-bank interference is further divided into *direct* and *transitive*. A request r_x experiences direct inter-bank interference when it participates to the inter-bank arbitration (i.e., it is at the top of its intra-bank queue) and suffers inter-bank interference. On the opposite, a request r_x experiences transitive inter-bank when it is not at the top of its intra-bank queue and suffers inter-bank interference. The latter phenomenon occurs when r_x suffers inter-bank interference due to other requests causing intra-bank interference to it that in turn are suffering inter-bank interference.

A table of symbols is reported in the appendix, available in the online supplemental material.

4 A Close Look at the MPAM Specification

Next, we proceed in two steps. First, in this section we take a close look to the official MPAM specification as provided by Arm [17] for what concerns memory-bandwidth and memory-priority partitioning. The specification presents several mechanisms to improve the memory access predictability and isolation, but unfortunately only providing a high-level and informal descriptions that leave large room for interpretation to hardware vendors that want to actually implement such mechanisms.

Later, in Section 5, we propose some possible instantiations of the MPAM specification, with the goal of both enabling timing analysis and guiding hardware vendors towards the best design choices to maximize predictability. In our proposals, to focus on solutions that are practically viable and that can be integrated with existing architectural designs with limited efforts, the MPAM mechanisms are combined with other standard design choices typically employed in COTS platforms aimed at maximizing average-case performance and throughput.

Overview. The Memory System Resource Partitioning and Monitoring (MPAM) [17] specification released by Arm for Armv8-A is designed for allowing the partitioning of memory system components (MSCs) shared among different applications and virtual machines. The specification states that each MSC, such as caches, interconnects, and DRAM memory controllers, might support MPAM (page 47 of [17]). The resource partitioning is based on the PARTID, an identifier that can be assigned, for example, to a VM or a task (when the MPAM PARTID Virtualization is enabled). The PARTID is then propagated through the MSCs to allow resource control and monitoring. When a new memory request r arrives in an MSC implementing MPAM, the measured usage of the MSC resources is compared with the control settings configured for the PARTID to which r is associated to. The MPAM architecture defines different control interfaces for MSCs, discussed below.

The MPAM specification provides two main categories of techniques: those related to caches and those related to memory bandwidth regulation.

The first category includes *cache-portion partitioning*, a mechanism that allows allocating storage portions of the cache to partitions, and *cache maximum-capacity partitioning*, which specifies a capacity limit for the storage used by a PARTID in the cache. This category is not studied in this paper. Rather, this work focuses on the second category, which includes the following memory-access regulation techniques: *memory-bandwidth minimum-maximum partitioning*, *priority partitioning*, and *memory-bandwidth portion*

partitioning. These three techniques are extensively reviewed below based on the information provided by the MPAM specification [17].

4.1 Memory-Bandwidth Min-Max Partitioning

The MPAM reference manual defines an interface for the management of memory bandwidth by configuring two values α_i^{M} and α_i^{m} for every partition ρ_i. These values respectively represent the maximum and minimum bandwidth to be assigned to ρ_i (e.g., to a task assigned to ρ_i). They can be specified by writing in the `MPAMCFG_MBW_MAX` and `MPAMCFG_MBW_MIN` registers, available in every MPAM-compliant MSC (e.g., a memory controller or an interconnect) that implements this partitioning strategy. Quoting the MPAM manual "*The control parameters for bandwidth partitioning schemes are all expressed in a fixed-point fraction of the available bandwidth*", which means the user can specify, by means of a percentage, how much of the total bandwidth can be used by each partition.

Since the MPAM specification does not provide a formal definition of bandwidth, in compliance with it, we assume it should be interpreted as the number of memory transactions in a given time interval. The size of such a time interval (for partition ρ_i) is referred to as *accounting window* and denoted as w_i. Based on these premises, the bandwidth used by each partition ρ_i can be measured by counting the number $q_i(t)$ of transactions served in the current accounting window. This value can then be compared with the minimum and maximum budget Q_i^m and Q_i^M, derived by the corresponding bandwidth values α_i^{m} and α_i^{M} as $Q_i^m = \lfloor \alpha_i^{\mathrm{m}} \cdot w_i \rfloor$, and $Q_i^M = \lfloor \alpha_i^{\mathrm{M}} \cdot w_i \rfloor$.

To keep track of the bandwidth usage, it is necessary to define the time interval over which the traffic measurement is performed. To this end, the MPAM manual proposes two possible high-level strategies to define a time window: *fixed* accounting window and *moving* accounting window. The specification also states the vendor is free to implement its custom accounting window strategy, as long as it is "*in line with the schemes described*" above. Since our purpose is to enable timing analysis, which requires well-defined accounting windows, this paper considers the fixed accounting window scheme. Memory traffic is hence measured by counting the requests issued over a fixed period of time that periodically repeats. When an accounting window terminates, a new window begins with no history of the used bandwidth (i.e., if an accounting starts at t^*, $q_i(t^* + k \cdot w_i) = 0, \forall \rho_i \in \mathcal{R}, k \in \mathbb{N}$). The MPAM specification offers a configurable register to customize the size of the window for each PARTID (`MPAMCFG_MBW_WINWD`), expressing its length in microseconds.

Next, we provide details on the maximum-minimum bandwidth partitioning, which may be either jointly or separately applied by an MPAM implementation.

*Memory-Bandwidth Minimum Partitioning (MB-mP).*MB-mP is a bandwidth partitioning strategy based on the design goal of enforcing a minimum bandwidth to every partition.

When using MB-mP, each PARTID ρ_i is characterized by a minimum bandwidth α_i^{m}. A minimum budget Q_i^m is defined to be compared with the current $q_i(t)$ value at runtime. The behavior of MB-mP complies to the following high-level rules, which have been extracted from the MPAM specification:

min1 If, during the current accounting window, partition ρ_i issued less than Q_i^m requests (i.e, $q_i(t) \leq Q_i^m$), requests from PARTID ρ_i are preferentially selected to be served.

min2 If, during the current accounting window, partition ρ_i already issued at least Q_i^m requests (i.e., $q_i(t) > Q_i^m$), "*requests from PARTID ρ_i may compete with other requests, as enabled by other regulation mechanisms implemented*", if any (e.g., maximum-bandwidth partitioning).

As one may note, the MPAM specification leaves a lot of room for interpretation: for example, if $q_i(t) \leq Q_i^m$ holds for multiple PARTIDs, it is not specified how their requests compete with each other. To better specify the behavior of MB-mP, we later discuss some design choices that we deem reasonable in Section 5.

*Memory-Bandwidth Maximum Partitioning (MB-MP).*MB-MP is a bandwidth partitioning strategy based on the design goal of enforcing a maximum bandwidth to every partition. When MB-MP is enabled, each partition ρ_i is characterized by a maximum bandwidth α_i^{M} assigned to it. Furthermore, for each PARTID, a boolean flag $h_i \in \{T, F\}$ can be configured to control how requests are handled when the current bandwidth consumption is above the maximum. In the MPAM specification, this flag corresponds to the `HARDLIM` bit of the `MPAMCFG_MBW_MAX` register.

A maximum budget Q_i^M is defined to be compared with the current $q_i(t)$ value at run time. The behavior of MB-MP complies to the following high-level rules (extracted from the MPAM specification):

MAX1 If, during the current accounting window, partition ρ_i issued an amount of memory requests between Q_i^m and Q_i^M (i.e., $Q_i^m \leq q_i(t) < Q_i^M$), requests from ρ_i are served when "*there are no competing minimum bandwidth requests to serve*". Requests for PARTIDs ρ_j such that $Q_j^m \leq q_j(t) < Q_j^M$ compete with each other to use bandwidth, as allowed by the other implemented MPAM regulation mechanisms.

MAX2 If, during the current accounting window, partition ρ_i already issued at least Q_i^M memory requests (i.e., $q_i(t) \geq Q_i^M$) and $h_i = F$, requests from ρ_i "*compete with other requests to use bandwidth only when there are no competing requests*" to serve for PARTIDs $\rho_j \neq \rho_i$ such that $q_j(t) < Q_j^M$.

MAX3 If, during the current accounting window, partition ρ_i already issued at least than Q_i^M memory requests (i.e., $q_i(t) \geq Q_i^M$) and $h_i = T$, requests from ρ_i are not served and saved for later service, namely, until $t_2 > t$ when $q_i(t_2) < Q_i^M$.

When minimum bandwidth partitioning is not implemented, the rules above still apply considering $Q_i^m = 0$.

Again: these rules are not precise enough in defining the behavior of MPAM to allow hardware vendors to directly implement them. To this end, additional design choices need to be considered, which are crucial to achieve a truly

predictable behavior. Therefore, later in the paper we discuss different alternatives to implement the specification.

4.2 Priority Partitioning

The MPAM reference manual defines an interface for the management of memory requests via the configuration of a fixed priority for the traffic generated by a specific partition.

To model this strategy, a priority π_i is defined to each partition ρ_i. In MPAM, this priority can actually be assigned to the PARTID by writing in the `INTPRI` field of the `MPAMCFG_PRI` register of the MSC. In this paper, since the reference manual describes it as an implementation-defined behaviour, we assume that higher values of π_i correspond to higher priorities.

4.3 Memory-Bandwidth Portion Partitioning

The MPAM reference manual describes a bandwidth partitioning strategy based on the division of the total available bandwidth into *portions*. Portions are not properly defined by the MPAM specification (it just says that *"a portion is a quantum of the bandwidth"*), and every vendor is free to implement a different strategy to enforce it.

Overall, this mechanism is unfortunately largely unspecified by the standard: Section 9.3.3 of the MPAM manual [17] describes it in just five lines (plus a few other lines in Appendix A.3 to describe the bitmap register, available in the online supplemental material).

Discussion. MPAM provides very general and flexible mechanisms that can be applied to different types of MSCs. Nevertheless, the implementation of the MPAM specification by hardware vendors requires exploring several design choices to be carefully evaluated. To better understand how to implement the MPAM specification to maximize predictability and providing recommendations to hardware vendors, we next explore some possible implementations of MPAM at the level of the *DRAM memory*. Then, we derive a memory contention analysis for each of them, which is finally used in the evaluation to compare different design alternatives. In this paper, we focus on the memory bandwidth maximum-minimum partitioning and the priority partitioning. Furthermore, we considered the first policy only for the case in which the `HARDLIM` bit is not set, so that there is no hard limitation of the maximum number of memory transactions that can be issued in an accounting window. This is because such a behavior would introduce a stall for the PARTID under analysis that requires complex analysis techniques and an even richer notation that we cannot fit in this paper. For the same reason, and given its very vague specification in [17], we leave as a future work the consideration of the portion partitioning strategy.

5 Modeling MPAM Mechanisms

Next, we model in detail some solutions to actually implement MPAM for controlling the contention at the DRAM level. We start with the MB-mp/MP partitioning scheme.

5.1 MB-mp/MP Partitioning in the MC

The first design choice for implementing this policy consists in deciding whether to implement it *inside* the MC, i.e., as part of its scheduling policy, or *outside* the MC, making MPAM acting as a filter for requests arriving at the MC. In our proposal, we consider the former solution: indeed, the min-max policy involves a dynamic prioritization of requests from different PARTIDs that, if implemented outside the MC, would be superseded by the internal scheduling policy of the MC.

To study a practical choice from a hardware vendor point of view, it is highly advisable to integrate MB-mp/MP partitioning with the standard mechanisms that can be found within a COTS MC, where requests causing row-hits (also called *open-requests*, as they target a row that is already open, i.e., loaded in the row-buffer) are privileged because they result in a shorter latency and hence maximize throughput. In the baseline memory controller presented in Section 3, this typical design choice is specified by Rule **MC2** (FR).

The baseline memory controller model has to be enriched to cope with the behavior of MPAM's MB-mP/MP mechanisms specified by Rules **min1**, **min2**, **MAX1**, **MAX2**, and **MAX3**. We note that such a behavior can be modeled as an *MPAM-related dynamic priority* attributed to memory requests issued by each PARTID ρ_i. The priority to be assigned to new requests changes over time, over three classes (i.e., three dynamic priority levels), depending on whether $q_i(t) < Q_i^m$, $Q_i^m \leq q_i(t) < Q_i^M$, or $q_i(t) \geq Q_i^M$, as three different priority levels that can be attributed to memory requests. The three conditions correspond to the priority levels $o_i = 3$ (*maximum-priority*), $o_i = 2$ (*medium-priority*), and $o_i = 1$ (*low-priority*). Once a request is assigned a priority, it remains unchanged during its entire lifetime.

To proceed, we need to define a set of accurate rules to describe how the dynamic priority of the MPAM min-max partitioning changes over time, and how the budget of transactions is managed:

- **mM1** *Counter reset.* At the system startup and every w_i time units since then, the counter $q_i(t)$ is set to 0.
- **mM2** *Dynamic priority.* If $q_i(t) < Q_i^m$, memory requests from ρ_i arriving at the MC are assigned with priority $o_i = 3$; If $Q_i^m \leq q_i(t) < Q_i^M$ requests from ρ_i are assigned with priority $o_i = 2$; If $q_i(t) \geq Q_i^M$ requests from ρ_i are assigned with a priority $o_i = 1$; once a request is assigned to a priority, its priority does not change.
- **mM3** *Counter increment.* Every time a memory request from ρ_i enters the MC, $q_i(t)$ is incremented by one.

Building on these priority classes, the intra-bank arbitration behavior is summarized by the rules that refine Rules **MC1** and **MC2** of the baseline memory controller (Section 3).

- **mM4**. *MINMAX.* Each per-bank queue is ordered by MPAM-related dynamic priority. FCFS tie-breaking is used for requests with the same priority.
- **mM5**. *FR-MINMAX.* The overall intra-bank scheduling policy applies, in order, first rule **MC2** and then rule **mM4**.

The resulting intra-queue scheduling behavior privileges, in order, **(i)** row-hits over row-conflicts, and **(ii)**

requests with a higher MPAM-related dynamic priority over requests with a lower priority.

Finally, we left both the inter-bank arbitration (Rule **MC4**) and write batching (Rule **MC5**) policies unaltered with respect to the baseline memory controller of Section 3.

Indeed, implementing the MB-mp/MP partitioning at the inter-bank level would privilege accesses to certain banks, possibly causing long delays to requests targeting such banks. For this reason, we deem a reasonable solution to keep the fair round-robin policy at the inter-bank level. Second, the considered baseline meta-memory controller handles write requests in batches, i.e., in a delayed fashion with respect to when they are issued. Since instead the dynamic priority of the MB-mp/MP policy changes over time, using it in a delayed fashion may make little sense.

5.2 Priority Partitioning in the MC

As discussed in Section 4.2, under priority partitioning each PARTID $\rho_i \in \mathcal{R}$ is assigned to a static priority π_i. Therefore, for each $\rho_i \in \mathcal{R}$ we define the sets of PARTIDs $\rho_j \in \mathcal{R} \setminus \rho_i$ with higher, equal and lower priorities as hp_i, ep_i, and lp_i, respectively. Requests inherit the priority of the issuing PARTID. The priority assigned to each PARTID has an effect in defining the intra-bank scheduling policy. Combining it with the prioritization of row-hits over row-conflicts typically used in COTS memory controllers, we consider two different possible variants for the *intra-bank* behavior of priority partitioning. The first, called *PP-FR*, retains the prioritization of requests directed to open rows and is specified by the following rules:

PP1. *Priority Partitioning.* Requests in each per-bank queue are ordered according to the priority assigned to the corresponding PARTID. FCFS tie-breaking is used for requests with the same priority.

PP2. The overall intra-bank scheduling policy applies, in order, first rule **MC2** (Section 3) and then rule **PP1**.

The second variant, called *FR-PP*, privileges requests directed to open rows only when there is a tie in priority, and is specified by Rule PP1 and the following one:

PP3. The overall intra-bank scheduling policy applies, in order, first rule **PP1** and then rule **MC2**.

As for the previous case, we leave the inter-bank arbitration policy (Rule **MC4**) unaltered with respect to Section 3, to avoid privileging accesses to certain banks. Similarly, we leave the write batching rule un-altered with respect to the baseline MC (Rule **MC5**), as writes are handled asynchronously in MCs with write batching.

6 Contention Analysis: Preliminaries

The analysis approach used in this paper is based on a linear programming formulation. Given an arbitrary schedule $\mathcal{S}$ and an arbitrary time window of length Δ, we define a set of variables and constraints to bound the memory contention experienced by *read* requests in the time window. As detailed in the appendix, available in the online supplemental material, the memory contention analysis can be then integrated with a response-time analysis to compute memory-aware response-time bounds [19], [25].

As noted in prior work [19], when deriving a response-time bound, only the contention due to reads actually delays the task under analysis. Indeed, writes do not stall the processing pipeline and can be handled asynchronously by the MC (if the write buffer is large enough, as assumed in Section 3).

The proposed optimization problem maximizes the delays experienced by memory transactions. Constraints to exclude impossible contention scenarios are enforced so that the solution of the optimization problem yields a safe memory-related delay bound. This approach is highly compositional: each constraint can be proved in isolation (i.e., with local reasoning) and reused whenever the property encoded by the constraint holds in the analyzed setting.

In this work we extensively benefit of this analysis approach: given that the scope of this work is to analyze several different MPAM mechanisms, it is possible to independently study each mechanism, derive the corresponding constraints, and finally plug them in the optimization problem together with those of the baseline MC.

Before proceeding, we introduce some additional notation that is required for the analysis. As in prior work [18], [24], we further introduce the architectural constant $N_{\text{pend}} > 1$ that defines the maximum number of outstanding read requests in the memory controller[1].

The proposed analysis is general enough to be applied under different circumstances: for this reason, we do not tie our analysis to a specific task model. Nevertheless, as an example, in the appendix, available in the online supplemental material, we show how to analyze the memory contention of sporadic real-time tasks under fixed-priority partitioned scheduling, also providing specific definitions of the following functions. We consider each PARTID to be associated with a sequential computational activity that issues memory transactions over time that are modeled as follows. Functions $\mathrm{RD}_{j,u}(\Delta)$ and $\mathrm{WR}_{j,u}(\Delta)$ bound the maximum number of read and write requests, respectively, issued from ρ_j to a bank $b_u \in \mathcal{B}$ in any interval of length Δ. These functions must include all the requests issued in the considered system. Functions $\mathrm{RD}_j(\Delta)$ and $\mathrm{WR}_j(\Delta)$ are defined for the same purpose, but considering all requests irrespectively of the target bank. Similarly, $\mathrm{RI}_{j,u}(\Delta)$ and $\mathrm{WI}_{j,u}(\Delta)$ bound the number of reads and writes from $\rho_j \in \mathcal{R}$ to bank $b_u \in \mathcal{B}$ in any interval of time Δ that are issued by ρ_j when it is the PARTID under analysis. $\mathrm{RI}_j(\Delta)$ and $\mathrm{WI}_j(\Delta)$ bound the same quantities but irrespectively of the target bank. When clear from the context, the dependency on Δ is hereafter omitted for brevity.

1. For example, in [24] this constant is given by the size of the Miss Status Handling Register (MSHR) of the shared last level cache, as in architectures with caches accesses to the DRAM are generated only in correspondence of a cache miss. Therefore, the size of this register determines the maximum number of cache misses that can be handled simultaneously, and consequently the maximum number of outstanding requests in the memory controller.

7 Baseline Analysis Framework

This section presents the baseline analysis framework. In particular, we introduce the modeling variables and the delay bounds that are common to all settings to be analyzed for MPAM. While we liberally take inspiration from the analysis framework of [19], the analysis proposed in this paper is fundamentally different. Indeed, [19] makes use of a different model for the cores and tasks. It assumes at most one pending read at a time from each core, tasks leveraging a three-phase execution model, and non-preemptive execution. Furthermore, it does not consider any bandwidth regulation technology. Conversely, this paper proposes a more general setting where multiple outstanding requests can be in the MC at the same time from different processing elements, as MPAM is designed for Armv8-A processors that leverage the out-of-order execution paradigm, and makes less stringent assumptions on the execution model.

We focus on the contention experienced by *read* requests. Indeed, in both our setting and in [19] writes do not stall the processing pipeline and the write buffer in the MC is assumed to never become full. Consequently, contention on writes issued by ρ_i do not delay ρ_i. Overall, the delay experienced by read requests is composed of **(i)** the contention due to other read requests, and **(ii)** the contention due to other write requests. Similarly to [19], we provide an optimization-based approach for bounding the maximum contention due to reads, and a closed form bound for the contention due to writes. Later, we use the baseline analysis as the capstone to analyze MPAM.

Therefore, we define four different classes of variables of the optimization problem to model contention in an arbitrary schedule $\mathcal{S}$ and in an arbitrary time interval $[t, t+\Delta)$, for each pair of bank $b_y \in \mathcal{B}$, PARTID $\rho_j \in \mathcal{R}$, and requests from ρ_i to b_u. Each class considers a different type of memory contention:

- $X^{\mathrm{IP}}_{n,j,u} \in \{0,1\}$: (*IP: intra-bank promoted*) indicates whether the n-th request issued by $\rho_j \in \mathcal{R}$ to bank $b_u \in \mathcal{B}$ causes *intra-bank* interference to any request of PARTID ρ_i by being promoted by the FR policy.
- $X^{\mathrm{INP}}_{n,j,u} \in \{0,1\}$: (*INP: intra-bank not promoted*) indicates whether the n-th request issued by $\rho_j \in \mathcal{R}$ to bank $b_u \in \mathcal{B}$ causes *intra-bank* interference to any request of PARTID ρ_i not because of the FR policy.
- $X^{\mathrm{CP}}_{n,j,u} \in \{0,1\}$: (*CP: cross-bank promoted*) indicates whether the n-th request issued by $\rho_j \in \mathcal{R}$ to bank $b_u \in \mathcal{B}$ causes *inter-bank* interference to any request of ρ_i by interfering with a request promoted by the FR policy.
- $X^{\mathrm{CNP}}_{n,j,u} \in \{0,1\}$: (*CNP: cross-bank not promoted*) indicates whether the n-th request issued by $\rho_j \in \mathcal{R}$ to bank $b_u \in \mathcal{B}$ causes *inter-bank* interference to any request of ρ_i by interfering with a request that was not promoted by the FR policy.

For all variable definitions, $n \in \{0, \ldots, \mathrm{RD}_{j,u}(\Delta) - 1\}$.

To ease the notation, we further define $X^{\mathrm{I}}_{n,j,u}, X^{\mathrm{C}}_{n,j,u} \in \{0,1,2\}$, such that $X^{\mathrm{I}}_{n,j,u} = X^{\mathrm{IP}}_{n,j,u} + X^{\mathrm{INP}}_{n,j,u}$ and $X^{\mathrm{C}}_{n,j,u} = X^{\mathrm{CP}}_{n,j,u} + X^{\mathrm{CNP}}_{n,j,u}$, to denote the entire intra-bank and inter-bank interference, respectively. Note that a request r_n can interfere with different pending requests from ρ_i under analysis in different ways.

The objective function maximizes the overall memory contention and it is defined as follows:

$$\textbf{max} \quad f = \sum_{\rho_j \in \mathcal{R} \setminus \rho_i} \sum_{b_u \in \mathcal{B}} \sum_{n=0}^{\mathrm{RD}_{j,u}-1} \max_{T \in \mathcal{T}} \left\{ \lambda^T(X^{\mathrm{T}}_{n,j,u}) \right\}, \tag{1}$$

where $\mathcal{T} = \{\mathrm{IP}, \mathrm{INP}, \mathrm{CP}, \mathrm{CNP}\}$ and $\lambda^T(x)$ denotes the contention delay implied by x requests that generate type-T contention[2]. These functions have been established in prior works [18], [19] and are recalled in the appendix, available in the online supplemental material, for the sake of completeness.

First, note that the first summation of the objective function excludes the requests from ρ_i. This is necessary because these requests cannot be a source of interference for ρ_i (e.g., if ρ_i is associated to a task, their duration is already taken into account in the task's WCET). However, they must be included in the analysis, given that their presence may allow additional interference by requests from other PARTIDs. Second, note that the objective function considers the maximum among all possible types of contention an arbitrary interfering request r_n may cause to ρ_i. This is because, while r_n can simultaneously interfere in different ways with multiple pending requests issued by ρ_i, the time spent by ρ_i waiting for the memory controller to serve r_n simultaneously elapses for each request that is interfered and it cannot be longer than the overall time required to serve r_n (e.g., think of a case in which r_n interferes with both a request of ρ_i in the same per-bank queue of r_n and with one in another queue).

7.1 Constraints of Baseline MC

We present several sets of constraints: each of them acts from a different angle to exclude impossible schedules and hence improving the accuracy of the analysis. The proofs are available in Section 3 of the appendix, available in the online supplemental material.

The first constraint bounds the intra-bank interference due to requests promoted by FR scheduling.

Constraint 1. *For each bank $b_u \in \mathcal{B}$:*

$$\sum_{\rho_j \in \mathcal{R}} \sum_{n=0}^{\mathrm{RD}_{j,u}-1} X^{\mathrm{IP}}_{n,j,u} \leq N_{\mathrm{thr}} \cdot \mathrm{RI}_{i,u}(\Delta).$$

The next constraint bounds the inter-bank interference.

Constraint 2. *For each bank $b_u \in \mathcal{B}$:*

$$\sum_{\rho_j \in \mathcal{R}} \sum_{n=0}^{\mathrm{RD}_{j,u}-1} X^{\mathrm{C}}_{n,j,u} \leq \sum_{b_y \in \mathcal{B} \setminus b_u} \left(\mathrm{RI}_{i,y} + \sum_{\rho_j \in \mathcal{R} \setminus \rho_i} \sum_{n=0}^{\mathrm{RD}_{j,y}-1} X^{\mathrm{I}}_{n,j,y} \right).$$

Constraint 3 bounds the inter-bank interference due to promoted requests.

2. The maximum term can be linearized using standard techniques that leverage auxiliary variables and the so-called big-M method [26].

Constraint 3. *For each bank $b_u \in \mathcal{B}$:*

$$\sum_{\rho_j \in \mathcal{R}} \sum_{n=0}^{\mathrm{RD}_{j,u}-1} X^{\mathrm{CP}}_{n,j,u} \leq \sum_{b_y \in \mathcal{B} \setminus b_u} \sum_{\rho_j \in \mathcal{R}} \sum_{n=0}^{\mathrm{RD}_{j,y}-1} X^{\mathrm{IP}}_{n,j,y}.$$

Finally, the last constraint leverages the constant N_{pend}.

Constraint 4. *For each bank $b_u \in \mathcal{B}$:*

$$\sum_{\rho_j \in \mathcal{R}} \sum_{n=0}^{\mathrm{RD}_{j,u}-1} X^{\mathrm{INP}}_{n,j,u} \leq (N_{\mathrm{pend}} - 1) \cdot \mathrm{RI}_{i,u}(\Delta).$$

7.2 Contention Due to Writes in the Baseline MC

Finally, read transactions can also be delayed by write transactions. Lemma 1 introduces a closed-form bound to the delay suffered by the read requests of a ρ_i under analysis due to write requests (Proof in Section 3 of the appendix, available in the online supplemental material).

Lemma 1. *The overall interference suffered by read requests issued by PARTID $\rho_i \in \mathcal{R}$ due to write requests in any time interval of length Δ is bounded by*

$$MC^{\mathrm{wr}}_i(\Delta) = \lambda_{\mathrm{WR}} \cdot \min\{NR(\Delta) \cdot N_{\mathrm{wb}}, \quad NW(\Delta) + Q_{\mathrm{write}}\}, \tag{2}$$

where
$NR(\Delta) = \sum_{b_u \in \mathcal{B}} (\mathrm{RI}_{i,u}(\Delta) + \sum_{\rho_j \in \mathcal{R} \setminus \rho_i} \mathrm{RD}_{j,u}(\Delta))$,
$NW(\Delta) = \sum_{b_u \in \mathcal{B}} \mathrm{WI}_{i,u}(\Delta) + \sum_{\rho_j \in \mathcal{R} \setminus \rho_i} \mathrm{WR}_{j,u}(\Delta)$, *and* λ_{WR} *is the maximum delay generated by a write request (defined in the appendix, available in the online supplemental material).*

8 Priority Partitioning

Priority partition inherits the same variable definitions, objective function, and constraints from the baseline analysis of Section 7. To start, we recall that the sets of PARTIDs $\rho_j \in \mathcal{R}$ (including ρ_i itself) with higher, equal, and lower priorities are denoted with hp_i, ep_i, and lp_i, respectively. In Section 5.2, two variants of priority partitioning have been defined: FR-PP and PP-FR, depending on whether the FR policy prevails on the priority ordering.

We introduce next a constraint, which is common to both FR-PP and PP-FR, to leverage the priority partitioning for limiting the interfering requests due to other PARTIDs.

Constraint 5. *Under both PP-FR and FR-PP, for each bank $b_u \in \mathcal{B}$, $\sum_{\rho_j \in \mathsf{lp}_i} \sum_{n=0}^{\mathrm{RD}_{j,u}-1} X^{\mathrm{INP}}_{n,j,u} \leq \mathrm{RI}_{i,u}(\Delta)$.*

Proof. The LHS counts the number of reads from $\rho_j \in \mathsf{lp}_i$ to b_u that generate intra-bank interference to ρ_i's requests not because of the FR policy. By Rule **PP1**, reads from $\rho_j \in \mathsf{lp}_i$ cannot be placed ahead to those issued by ρ_i in the queue of b_u. However, since requests are served non-preemptively, each read r from ρ_i can be delayed by at most one request from $\rho_j \in \mathsf{lp}_i$ that may have started to be served before r enters the MC. Since ρ_i can issue at most $\mathrm{RI}_{i,u}(\Delta)$ requests to bank b_u, the constraint follows. □

Next, we distinguish between the FR-PP and PP-FR.

Under FR-PP, the FR policy is applied before priority partitioning (Rule **PP2**). Therefore, Constraint 1 of the baseline analysis suffices to encode the FR part of the policy.

Conversely, under PP-FR, the priority partitioning policy prevails over the FR one. This information is used next to derive an accurate constraint on variables $X^{\mathrm{IP}}_{n,j,u}$.

Constraint 6. *Under PP-FR, for each bank $b_u \in \mathcal{B}$, $\sum_{\rho_j \in \mathcal{R} \setminus \mathsf{ep}_i} \sum_{n=0}^{\mathrm{RD}_{j,u}-1} X^{\mathrm{IP}}_{n,j,u} \leq 0$.*

Proof. Due to Rule **PP3**, the priority partitioning policy is applied before the first-ready policy. Therefore, the requests promoted due to the FR policy must have the same priority of those of ρ_i. The constraint follows. □

The interference due to write requests is bounded as in Section 7, as this feature of the baseline MC is left unaltered by the priority partitioning policy.

9 MB-mp/MP Partitioning Analysis

Next, we show how to extend the baseline analysis framework to work under the MB-mp/MP policy of MPAM (Section 5.1). Aiming at bounding the contention experienced by a PARTID $\rho_i \in \mathcal{R}$ under analysis, we start providing some bounds that are useful to formulate fine-grained constraints to limit the memory contention.

9.1 Bounds on the Number of Interfering Requests

We first derive two bounds for the requests issued by every PARTID ρ_j that can be served at a given dynamic priority $o_j \in \{3, 2\}$ in any interval of length Δ. The bounds can be obtained by considering that, in an arbitrary time interval $[t, t + \Delta)$, each PARTID can issue a limited number of requests before its priority class is changed (rules **mM1-mM3**). This is caused by the fact that when ρ_j crosses the thresholds of Q^m_j or Q^M_j transactions in the accounting window w_j, its priority is decreased by MPAM.

Lemma 2. *The number of transactions at priority $o_j \in \{3, 2\}$ of a PARTID $\rho_j \in \mathcal{R}$ in any time interval $[t, t + \Delta)$ is bounded by $I^3_j(\Delta) := (\lceil \Delta / w_j \rceil + 1) \cdot Q^m_j$ and $I^2_j(\Delta) := (\lceil \Delta / w_j \rceil + 1) \cdot (Q^M_j - Q^m_j)$ respectively.*

Proof. We discuss the two bounds separately.

$I^3_j(\Delta)$: At priority $o_i = 3$, the PARTID can issue at most Q^m_j transactions for each accounting window before its priority is decreased to 2 (rules **mM1-mM3**).

$I^2_j(\Delta)$: Similarly, at priority $o_i = 2$, the PARTID can issue at most $Q^M_j - Q^m_j$ transactions for each window before its priority is decreased to 1 (rules **mM1-mM3**).

The lemma follows by noting that the number of requests that can be issued at each priority level during $[t, t + \Delta)$ is composed of: **(i)** at most $\lceil \Delta / w_j \rceil \cdot Q^m_j$ and $\lceil \Delta / w_j \rceil \cdot (Q^M_j - Q^m_j)$ transactions, respectively, for the accounting windows starting within $[t, t + \Delta)$, and **(ii)** at most Q^m_j and $Q^M_j - Q^m_j$ transactions, respectively, for a carry-in accounting window started before time t and completing in $[t, t + \Delta)$. □

9.2 Bounding the Memory Contention

Consider the PARTID $\rho_i \in \mathcal{R}$ under analysis. With respect to the case of Section 7, the MB-mp/MP memory controller analysis needs to re-define variables $X^{\mathrm{INP}}_{n,j,y}$, as they leverage rule **MC1** that is superseded by Rule **mM4**. Furthermore,

we need to introduce some new variables to model the behaviors that are specific to the MB-mp/MP policy.

Therefore, for each partition $\rho_j \in \mathcal{R}$, for each memory bank $b_u \in \mathcal{B}$, and for each pair of MB-mp/MP priorities $o_i, o_j \in \{1,2,3\}$, we define:

- $X_{n,j,u}^{\mathrm{INP},o_i,o_j} \in \{0,1\}$ indicates whether the n-th request issued by ρ_j to bank b_u is served at dynamic priority o_j and causes *intra-bank* interference to at least one request from ρ_i assigned to dynamic priority o_i because of the MB-mp/MP or the FCFS policies.

The objective function is the same used in Section 7, provided that we redefine $X_{n,j,y}^{\mathrm{INP}} = \max_{o_i,o_j \in \{1,2,3\}} \{X_{n,j,y}^{\mathrm{INP},o_i,o_j}\}$.

The maximum is required because, even though a request r_n can simultaneously interfere with multiple pending requests from ρ_i having different dynamic priorities, the time spent by ρ_i waiting for the memory controller to serve r_n simultaneously elapses for each interfered request.

We start introducing bounds for the newly defined variables. Constraint 7 enforces that, given a priority $o_i \in \{1,2,3\}$, an interfering request can have only one priority.

Constraint 7. *For each PARTID $\rho_j \in \mathcal{R}$, for each $b_u \in \mathcal{B}$, for each $n \in [0, \mathrm{RD}_{j,u} - 1]$, for each $o_i \in \{1,2,3\}$:* $\sum_{o_j=1}^{3} X_{n,j,u}^{\mathrm{INP},o_i,o_j} \leq 1$.

Proof. By contradiction, if the constraint does not hold, i.e., more than one variables $X_{n,j,u}^{\mathrm{INP},o_i,o_j}$ is equal to 1 for a given request, then there exists at least one request that interfered with ρ_i while having more than one priority. This is impossible by rule **mM2**, which states that the dynamic priority of any request cannot change over time. □

However, Constraint 7 does not prevent to the *same* (arbitrary) request r_n from ρ_j to interfere, by taking more than one dynamic priority o_j, with *multiple* requests of the PARTID under analysis ρ_i assigned to different priorities (e.g., o_i and another one in $\{1,2,3\} \setminus o_i$). Again, this is clearly impossible because the dynamic priority of r_n does not change over time due to rule **mM2**. Constraint 8 excludes this impossible case.

Constraint 8. *For each PARTID $\rho_j \in \mathcal{R}$, for each $b_u \in \mathcal{B}$, for each $n \in [0, \mathrm{RD}_{j,u} - 1]$, for each $o_j \in \{1,2,3\}$, for each $o_i \in \{1,2,3\}$, for each $o_h \in \{1,2,3\} \setminus o_i$:*

$$X_{n,j,u}^{\mathrm{INP},o_i,o_j} \leq 1 - P_{n,j,u}^{o_h,o_j},$$

$$\textit{where } P_{n,j,u}^{o_h,o_j} = \sum_{o_k \in \{1,2,3\} \setminus o_j} X_{n,j,u}^{\mathrm{INP},o_h,o_k},$$

Proof. First note that, due to Constraint 7, $P_{n,j,u}^{o_h,o_j}$ can be either equal to 0 or 1. Observe that $P_{n,j,u}^{o_h,o_j}$ indicates whether (=1) or not (=0) any request by ρ_i at priority $o_h \in \{1,2,3\} \setminus o_i$ is interfered by the n-th request of ρ_j to bank b_u with a priority $o_k \neq o_j$. If $P_{n,j,u}^{o_h,o_j} = 0$, it means that r_n does not interfere with a priority $o_k \neq o_j$. In this case the constraint has no effect as it simply enforces $X_{n,j,u}^{\mathrm{INP},o_i,o_j} \leq 1$.

Since multiple outstanding requests are allowed by the MC, ρ_i can have simultaneously pending requests assigned to dynamic priorities 1, 2, and 3. By contradiction, assume there exists a request r_n from ρ_j to bank b_u such that r_n takes two different priorities o_j and $o_k \neq o_j$ over time. Assume also that r_n interferes with multiple requests of the PARTID ρ_i under analysis with priorities o_i and $o_h \neq o_i$. It hence means that $X_{n,j,u}^{\mathrm{INP},o_i,o_j} = 1$ and that $P_{n,j,u}^{o_h,o_j} = 1$. Therefore, the constraint reduces to $1 \leq 0$, which is impossible, reaching a contradiction. □

Constraint 9 leverages the bounds of Lemma 2 to limit the number of interfering requests.

Constraint 9. *For each PARTID $\rho_j \in \mathcal{R}$, for each $o_j \in \{2,3\}$, for each $o_i \in \{1,2,3\}$:*

$$\sum_{b_u \in \mathcal{B}} \sum_{n=0}^{\mathrm{RD}_{j,u}-1} X_{n,j,u}^{\mathrm{INP},o_i,o_j} \leq I_j^{o_j}(\Delta)$$

Proof. The LHS counts all the requests of PARTID ρ_j at priority o_j interfering with requests from ρ_i at priority o_i. As stated by Lemma 2, the number of requests that can be emitted by ρ_j at priority o_j is bounded by the RHS. Hence the constraints follows. □

Next, we impose Constraint 10 to limit the interference due to requests at a lower dynamic priority.

Constraint 10. *For each bank $b_u \in \mathcal{B}$, for each $o_i \in \{2,3\}$:*

$$\sum_{\rho_j \in \mathcal{R}} \sum_{n=0}^{\mathrm{RD}_{j,u}-1} \sum_{o_j=1}^{o_i-1} X_{n,j,u}^{\mathrm{INP},o_i,o_j} \leq \mathrm{RI}_{i,u}(\Delta)$$

Proof. The LHS of the inequality counts all the requests issued with a priority $o_j < o_i$. Due to Rule **mM4**, no request from ρ_j can cause intra-bank interference at a priority $o_j < o_i$ if there are pending requests from ρ_i at priority o_i. However, since requests are served non-preemptively, for each request r from ρ_i, at most a single request may contribute to the LHS of the inequality because it is started before r is enqueued. Since ρ_i can issue at most $\mathrm{RI}_{i,u}(\Delta)$ requests to bank b_u, the constraint follows. □

The following constraint leverages the constant N_{pend} to limit the maximum number of interfering requests at the same priority (FCFS tie-breaking).

Constraint 11. *For each $o = o_i = o_j \in \{2,3\}$:*

$$\sum_{\rho_j \in \mathcal{R}} \sum_{b_u \in \mathcal{B}} \sum_{n=0}^{\mathrm{RD}_{j,u}-1} X_{n,j,u}^{\mathrm{INP},o,o} \leq (N_{\mathrm{pend}} - 1) \cdot I_i^o(\Delta).$$

Proof. First note that requests from PARTID $\rho_j \in \mathcal{R}$ served at priority $o = o_i = o_j$ and producing intra-bank interference to pending requests from ρ_i at the same priority need to be issued *before* those of ρ_i due to FCFS tie-breaking (Rule **mM4**). Whenever a request r with priority o is issued by ρ_i, there can be at most other $N_{\mathrm{pend}} - 1$ requests that are pending in the MC with priority o, which can interfere with r because they were issued before.

The LHS counts the overall amount of requests by ρ_j interfering because of the FCFS policy at priority o. These requests are no more than $N_{\mathrm{pend}} - 1$ for each request issued at priority o by ρ_i. The constraint follows by noting that the number of requests issued by ρ_i at priority o in the analysis window is bounded by $I_i^o(\Delta)$ (see Lemma 2). □

9.3 Interference to Requests with Dynamic Priority 1

We now focus on the interference to requests with priority 1. Note that such requests are not addressed by Constraints 10, and 11, while bounding the interference they suffer is of the utmost importance.

Indeed, it may happen that ρ_i starts its execution with the budget of higher priority requests already exhausted, and all its requests enter the MC during this first accounting window. In this case, all of them would have priority 1. By using the previously discussed constraints only, it would be legal for interfering PARTIDs that all their requests enter the MC just a moment before ρ_i and, by consequence, ρ_i would suffer all the possible interference due to the FCFS policy, as if the MPAM's MB-mp/MP policy is not present. Ultimately, this would result in making the MB-mp/MP strategy useless, since the $X^{\mathrm{INP},1,1}_{n,j,u}$ variable would not be bounded by any MB-mp/MP specific constraint.

To address this issue, we need to introduce a mapping between interfering requests and the window in which they enter the MC. In this way, we allow the optimizer to decide how many requests from ρ_i enter the MC in each accounting window, and to match them with interfering requests. To this end, we leverage a typical property of function $\mathrm{RD}_{j,u}(\Delta)$. Indeed, $\mathrm{RD}_{j,u}(\Delta)$, as shown in the appendix, available in the online supplemental material, is a monotonic non-decreasing function, most commonly in a step-wise periodic fashion. Therefore, not all interfering requests are available in all windows (e.g., think of the case in which the PARTID is assigned to a periodic task).

Bounding the interference that can be suffered by requests with the lowest priority is not trivial, and requires introducing additional notation.

For each $b_u \in \mathcal{B}$, for each accounting window with index $z \in \mathcal{W}_i = \{0, \ldots, \lceil \Delta / w_i \rceil\}$, we define the variables:

- $Y_{u,z} \in \mathbb{N}$ denotes the number of requests by the PARTID under analysis ρ_i to bank b_u that enter the MC during the z-th window elapsed in the interval $[t, t + \Delta)$.

We start dividing the total number of requests based on the accounting window they enter the MC.

Constraint 12. *For each $b_u \in \mathcal{B}$: $\sum_{z \in \mathcal{W}_i} Y_{u,z} \leq \mathrm{RI}_{i,u}(\Delta)$.*

Proof. By definition, $Y_{u,z}$ counts the number of requests issued by ρ_i to bank b_u during the z-th accounting window. Provided that $\mathcal{W}_i$ contains all the windows elapsed during the time period Δ, since the windows are not overlapped, it is straightforward that their sum must not be larger than the total number of requests issued by ρ_i to bank b_u. □

A bound on the requests that may be issued at priority 1 can now be derived in a specific constraint, which also introduces the auxiliary term $H^1_{i,u}(\Delta)$.

Constraint 13. *The maximum number of requests by ρ_i to bank b_u that can enter the MC at priority 1 in the analysis window $[t, t + \Delta)$ are bounded by:*
$H^1_{i,u}(\Delta) = \mathrm{RI}_{i,u}(\Delta) - \sum_{z \in \mathcal{W}_i \setminus \{0\}} \min\{Q^M_i, Y_{u,z}\}$

Proof. The total number of requests issued by ρ_i to bank b_u in the analysis window of interest is bounded by $\mathrm{RI}_{i,u}(\Delta)$, irrespectively of their dynamic priority. Among such requests, for each accounting window with index $z > 0$, by Rule **mM2**, Q^M_i requests have to be issued at dynamic priorities 3 and 2 before issuing requests at priority 1. Furthermore, by definition of variables $Y_{u,z}$, no less than $\min\{Q^M_i, Y_{u,z}\}$ requests can anyway be issued at dynamic priorities 2 and 3. The same cannot be argued for the first accounting window ($z = 0$) because it may have started outside the analysis window of interest, i.e., before time t. Hence, $\sum_{z \in \mathcal{W}_i \setminus \{0\}} \min\{Q^M_i, Y_{u,z}\}$ requests of the $\mathrm{RI}_{i,u}(\Delta)$ ones cannot surely enter the MC at dynamic priority 1. The constraint follows. □

Thanks to the auxiliary term $H^1_{i,u}(\Delta)$ introduced by Constraint 13, we can derive Constraint 14, which is analogous to Constraint 11 but considering requests at dynamic priority 1. The proof is analogous and thus is omitted.

Constraint 14. *For each bank $b_u \in \mathcal{B}$, $\sum_{\rho_j \in \mathcal{R}} \sum_{n=0}^{\mathrm{RD}_{j,u}-1} X^{\mathrm{INP},1,1}_{n,j,u} \leq (N_{\mathrm{pend}} - 1) \cdot H^1_{i,u}(\Delta)$.*

9.4 Bounding the Number of Transactions per Window

Lastly, we need a constraint that prevents requests that are available only after a specific offset in the analysis interval (e.g., which are due to jobs that did not start yet if PARTIDs are associated with periodic tasks), from being accounted for in an accounting window that already ended.

We start by bounding the number of requests by ρ_i that can enter the MC at priorities 1 and 2, but only starting from a certain accounting window with index z_0.

Constraint 15. *Let $\overline{\mathcal{W}}_i(z_0) = \mathcal{W}_i \setminus \{0, \ldots, z_0 - 1\}$, if $z_0 \geq 1$, or $\overline{\mathcal{W}}_i(z_0) = \mathcal{W}_i$ otherwise. For each bank $b_u \in \mathcal{B}$, for each $z_0 \in \mathcal{W}_i$, the number of requests by ρ_i that enter the MC after or at the z_0-th window with priority 1 is bounded by:*

$$B_{1,z_0} = \begin{cases} \sum_{z \in \overline{\mathcal{W}}_i(z_0)} B^*_{1,z_0}, & if\ z_0 \neq 0 \\ Y_{u,0} + \sum_{z \in \overline{\mathcal{W}}_i(z_0) \setminus \{0\}} B^*_{1,z_0}, & if\ z_0 = 0, \end{cases}$$

with $B^*_{1,z_0} = \max\{0, Y_{u,z} - Q^M_i\}$. Similarly, those with priority 2 are bounded by:

$$B_{2,z_0} = \begin{cases} \sum_{z \in s\overline{\mathcal{W}}_i(z_0)} B^*_{2,z_0}, & if\ z_0 \neq 0 \\ \min\{Y_{u,0}, Q^M_i - Q^m_i\} + \sum_{z \in \overline{\mathcal{W}}_i(z_0) \setminus \{0\}} B^*_{2,z_0}, & if\ z_0 = 0 \end{cases}$$

with $B^*_{2,z_0} = \max\{0, \min\{Y_{u,z} - Q^m_i, Q^M_i - Q^m_i\}\}$.

Proof. First consider requests at priority 1. We distinguish two cases: **(i)** $z_0 \neq 0$, and **(ii)** $z_0 = 0$. *Case (i).* The accounting windows with indexes in $\overline{\mathcal{W}}_i(z_0)$ must all have started in the analysis interval $[t, t + \Delta)$. By Rule **mM2**, to issue requests at priority 1 in one of the accounting windows with index in $\overline{\mathcal{W}}_i(z_0)$, ρ_i must have before issued Q^M_i requests with priorities > 1 in the same. By recalling the definition of $Y_{u,z}$, the number of requests with priority 1 in such accounting windows is hence bounded $\sum_{z \in s\overline{\mathcal{W}}_i(z_0)} \max\{0, Y_{u,z} - Q^M_i\}$. *Case (ii).* The first accounting window ($z = 0$) may have started before the beginning of the analysis interval $[t, t + \Delta)$. Irrespectively of their priority, no more than $Y_{u,0}$ requests can be issued in the first accounting window. The same reasoning of case (i) applies to the next accounting windows with index in $\overline{\mathcal{W}}_i(z_0) \setminus \{0\}$. Hence the constraint for term B_{1,z_0} holds.

Consider now requests at priority 2 and let us distinguish the same two cases as above. *Case (i).* By Rule **mM2**, no more than $Q^M_i - Q^m_i$ requests with priority 2

can be issued in each accounting window. Furthermore, for the same reasons discussed for requests at priority 1, to issue requests at priority 2 in one of the accounting windows with index in $\overline{\mathcal{W}}_i(z_0)$, ρ_i must have before issued Q_i^m requests with priority 3 in the same. Hence, $Y_{u,z} - Q_i^m$ also yields a safe bound for the number of requests at priority 2 issued in the z-th interval, with $z > 0$. Therefore, the total number of requests with priority 2 in such accounting windows is bounded by $\sum_{z \in s\overline{\mathcal{W}}_i(z_0)} \max\{0, \min\{Y_{u,z} - Q_i^m, Q_i^M - Q_i^m\}$. *Case (ii).* The same reasoning used for case (ii) related to requests with priority 1 can be used after nothing that $\min\{Y_{u,0}, Q_i^M\}$ bounds the number of requests with priority 2 in the first accounting window. Hence the constraint for term B_{2,z_0} holds. □

With Constraint 15 in place, we can bound the interference produced by the requests from ρ_j that entered the MC during the z_0-th accounting window or later. Due to the FCFS policy, these requests can only produce interference to requests from ρ_i that are enqueued after the beginning of the z_0-th accounting window.

Constraint 16. *Let $\overline{\mathcal{W}}_i(z_0) = \mathcal{W}_i \setminus \{0, \ldots, z_0 - 1\}$, if $z_0 \geq 1$, or $\overline{\mathcal{W}}_i(z_0) = \mathcal{W}_i$ otherwise. Let $t_0 = z_0 \cdot w_i$. For each bank b_u in $\mathcal{B}$, for each $z_0 \in \mathcal{W}_i$:*

$$a) \sum_{\rho_j \in \mathcal{R}} \sum_{n=\mathrm{RD}_{j,u}(t_0)}^{\mathrm{RD}_{j,u}(\Delta)-1} \sum_{o=1}^{3} X_{n,j,u}^{\mathrm{INP},o,o} \leq (N_{\mathrm{pend}} - 1) \cdot \sum_{z \in \overline{\mathcal{W}}_i(z_0)} Y_{u,z}$$

$$b) \sum_{\rho_j \in \mathcal{R}} \sum_{n=\mathrm{RD}_{j,u}(t_0)}^{\mathrm{RD}_{j,u}(\Delta)-1} X_{n,j,u}^{\mathrm{INP},1,1} \leq (N_{\mathrm{pend}} - 1) \cdot B_{1,z_0}$$

$$c) \sum_{\rho_j \in \mathcal{R}} \sum_{n=\mathrm{RD}_{j,u}(t_0)}^{\mathrm{RD}_{j,u}(\Delta)-1} X_{n,j,u}^{\mathrm{INP},2,2} \leq (N_{\mathrm{pend}} - 1) \cdot B_{2,z_0},$$

where B_{1,z_0} and B_{2,z_0} are defined by Constraint 15.

The proof is available in Section 3 of the appendix, available in the online supplemental material.

The contention due to writes is bounded analogously as for the baseline MC, as this feature is left unaltered by the min-max policy.

Relaxation. Differently from other problems that require the value of specific variables (e.g., as in decision problems such as task partitioning), the expected output of the optimization problem is only the (maximized) value of the objective function. Therefore, from a practical point of view, the problem was implemented by relaxing all the binary variables representing the interfering requests as real variables in the interval [0,1]. This action does not affect the correctness of the computed response-time bound since, by relaxing the variables, the search space available to the optimizer is enlarged. Therefore, the maximum of the objective function of the relaxed problem can only be higher than or equal to the maximum of the original problem. Hence, the upper bound is still safe. Furthermore, to evaluate the difference between the two models, we performed a set of tests by running the same problem instance with and without relaxation (based on some configurations used in our evaluation, presented in Section 10). In all the tested cases, the optimal solution did not change.

10 Experimental Evaluation

This section reports on experimental evaluation we performed to test the effectiveness of the MPAM mechanisms. To this end, we implemented the memory-aware response-time analysis in the case in which a PARTID is a sporadic real-time task (see the appendix for more details, available in the online supplemental material). The proposed optimization problems have been implemented with IBM CPLEX on a machine with 128GB of memory and two Intel Xeon(R) CPU E5-2640 v4 @ 2.40GHz, with 40 cores in total. The evaluation focuses on a case study derived from the task set proposed by Bosch for the WATERS 2019 Industrial Challenge [27], which includes nine tasks (task $0 - 8$). The task mapping is based on the challenge solution of [28]. Additional details about the tasks and the number of memory requests are available in Table 2 in Appendix, available in the online supplemental material. When considering memory contention in bounding the worst-case response time, most tasks were actually not schedulable with the original setting. We introduced two additional parameters $\xi_R, \xi_W \in [0, 1]$, to scale the number of reads and write requests, respectively. In the evaluation, we explored a vast range of values for ξ_R and ξ_W to evaluate different trade-offs. We considered the JEDEC timing constraints for a DDR3 MC running at 1333 Mhz, which are reported in the appendix, available in the online supplemental material, for the sake of completeness. We configured the MC parameters as in prior work [19], [24], namely, $N_{\mathrm{thr}} = 18$, $N_{\mathrm{wb}} = 18$, $Q_{write} = 64$, and $N_{\mathrm{pend}} = 24$. We consider a DRAM memory with $N_B = 4$ banks.

In our experiments, we compare the memory contention under different configurations: **(i)** `baseline`: the baseline MC without any MPAM mechanism; **(ii)** priority partitioning with `FR-PP`; **(iii)** priority partitioning with `PP-FR`; and **(iv)** `MB-mp/MP`: the MB-mp/MP MPAM mechanism.

The first three charts focus on the comparison between the two variants of the priority partitioning strategy and the `baseline` system. To this end, we explore three different priority assignment configurations: **(i)** `max improvement`, which independently considers the case in which each task is assigned the maximum MPAM priority, while the other tasks are all assigned the same lower priority; **(ii)** `rate monotonic`, which assigns the priority inversely proportional to the tasks' period; **(iii)** `request monotonic`, which assigns the priority inversely proportional to the number of requests per job of each task.

All the values on the y-axis represent the overall memory interference, as a percentage of the memory interference obtained when using the `baseline` configuration (i.e., when MPAM is disabled). Fig. 2 reports on a comparison of the `FR-PP` and `PP-FR` policies under three different priority assignment strategies. The requests' scaling factors are $\xi_R = 0.35$ and $\xi_W = 0.05$. The same experiments were also carried out with other scaling factors and the results are presented in Appendix, available in the online supplemental material. Fig. 2a shows that `FR-PP` and `PP-FR` allows achieving a maximum improvement of up to 21% and 27% to the

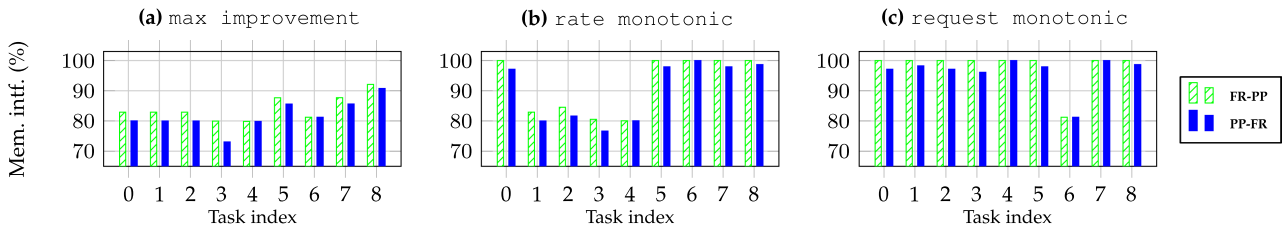

Fig. 2. Memory interference obtained using the FR-PP and PP-FR strategies under different priority assignments, as a percentage of the memory interference obtained with the baseline MC. The scaling factors are $\xi_R = 35\%$ and $\xi_W = 5\%$.

baseline, respectively. Furthermore, it highlights that PP-FR generally performs better than FR-PP in reducing the memory interference delay suffered by the highest-priority task. This is attributed to the fact that under PP-FR the prioritization prevails over requests targeting open rows, which allows posing additional constraints on the intra-bank interference. Thus, increasing the priority has a larger effect when using PP-FR. In each configuration, all other tasks (which have been assigned to the same low priority in this case) showed no improvement.

In Fig. 2b, instead, tasks are assigned priorities according to rate monotonic. In this case, tasks with shorter periods (e.g., tasks $1-4$) achieve a more significant improvement (up to a 24% reduction with task 1 and PP-FR), while tasks with a longer period (e.g., tasks $5-8$) have a very small improvement because they are assigned to lower priorities.

In Fig. 2c, priorities are assigned proportionally to the number of requests per job of each task. In this case, task 6 achieves the most remarkable improvement of 19%.

The next three plots consider the improvement that can be granted to the most memory-intensive task of the benchmark (i.e., τ_6) while varying the parameters of the MB-mp/MP strategy. An additional hybrid task set (details available in the Appendix, Table 3, available in the online supplemental material) was synthesized for these specific experiments, still starting from the WATERS 2019 one. This was deemed necessary to show some relevant configurations where the MB-mp/MP strategy provides some benefits from a worst-case analysis point of view, where in most configurations we observed very few improvements over the baseline. For the sake of completeness, other experiments that use the original task set are available in Appendix, available in the online supplemental material.

In the following, the number of read requests per job for a task τ_j is denoted as $\mathrm{rd}_j = \sum_{b_u \in \mathcal{B}} \mathrm{rd}_{j,u}$, the ratio between Q_j^m and Q_j^M of τ_j as $\gamma_j^{\mathrm{mM}} = Q_{j/}^m Q_j^M$, the ratio between Q_j^M and rd_j as $\beta_j^{\mathrm{M}} = Q_j^M/\mathrm{rd}_j$. In Fig. 3a, we show the effects of the variation of parameter β_j^{M} for each interfering task $\tau_j \neq \tau_6$ (the same value of β_j^{M} is used for all interfering tasks). The plot reports three curves each corresponding to a representative configuration we selected by varying the ratio γ_j^{mM}. All the other parameters were assigned as follows: $Q_6^m = Q_6^M = \mathrm{rd}_j$; $w_6 = 0.03 \cdot T_6$; $w_j = T_j$, for $j \neq 6$. The values of w_i were carefully selected after running a large number of experiments to allow Constraints 12, 13, 14, 15, and 16 to have a remarkable effect in reducing the memory interference suffered by τ_6 itself, thanks to a large number of accounting windows, and hence high-priority requests, for τ_6 compared to the other tasks. Indeed, the MB-mp/MP strategy resulted hard to properly configure to achieve a substantial effect in reducing memory-contention delays, with little improvement under considerably different values of w_i. Fig. 3a shows that τ_6 has a considerable improvement (about 15%) with respect to the baseline MC only when $\beta_j^{\mathrm{M}} = 0$, which implies $Q_j^M = 0$. Indeed, it can be noted from the figure how an increase of β_j^{M} causes a rapid reduction of the improvement. The only exception is for the case with $\gamma_j^{\mathrm{mM}} = 0$, where MPAM is able to provide a 12% gain with respect to the baseline configuration for τ_6 also for larger values of β_j^{M}. Note that in this case $Q_j^m = 0$: hence interfering tasks cannot release any request with maximum priority.

Fig. 3b shows the effect of the variation of γ_j^{mM} for the interfering tasks. The plot reports three curves, for three representative configurations we selected by varying the ratio β_j^M. The results confirm how an increase of the maximum budget causes an up to 12% reduction of memory interference for the task under analysis. It is also interesting to notice how the improvement reaches 0% as soon as $\gamma_j^{mM} = 0.3$. This indicates that is necessary to keep the number of higher priority requests from interfering task really low to allow the MB-mp/MP strategy having a positive effect for τ_6 in the worst-case scenario.

Fig. 3c investigates on how changing the total number of read requests of the interfered tasks (indicated as a percentage of the base case) affects the memory interference limitation provided by the MB-mp/MP. In this case, task 6 has been assigned $Q_6^M = Q_6^m = \mathrm{rd}_6$, and the other tasks have $\gamma_j^{\mathrm{mM}} = 0.2$. Three representative configurations are reported, corresponding to $\beta_j^{\mathrm{M}} \in \{0, 0.2, 0.4\}$. It is interesting to notice how the interference is very close to the baseline when the number of requests is at the two extremes of the tested interval, while they are more distant in the middle (up to a 16% improvement). First, to understand this behavior, please note that, since $Q_6^m = \mathrm{rd}_6$, no other PARTID can issue request at priority higher than ρ_6. We then need to take into account the fact that non-promoted intra-bank interfering requests (i.e., those due to variables $X_{n,j,u}^{\mathrm{INP},o_i,o_j}$ that are affected by MB-mp/MP) with lower or equal priority can interfere for two reasons: **(i)** they can have a lower priority than the interfered request and thus cause a delay due to the non-preemptive service of requests; or **(ii)** they can have an equal priority, and thus interfere because of the FCFS policy. Case (i) causes one interfered request to match at most one interfering request (see Constraint 10), while case (ii) causes each interfered request to be matched with at most N_{pend} interfering requests (Constraints 11 and 16).

When the interfered (i.e., task 6) has a low number of requests, the number of requests due to interfering tasks τ_j /

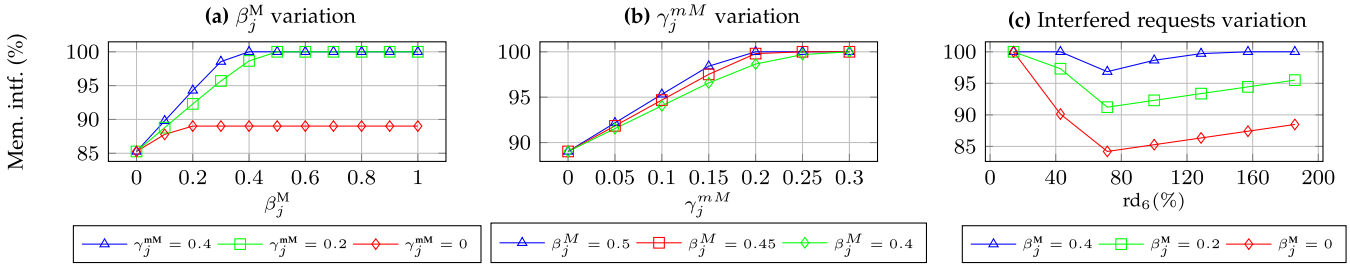

Fig. 3. Memory interference experienced by task τ_6 obtained using the `MB-mp/MP` strategy under different parameters configurations, as a percentage of the memory interference obtained with the `baseline` MC.

$= \tau_6$, i.e., the number of interfering requests available at the beginning of the interval (due to functions $\mathrm{RD}_j(\Delta)$) are enough to interfere with most of the task 6's requests because of the FCFS policy during the first accounting window (case (i)), and the solver does not need to move requests of the interfered tasks in subsequent windows to match more interfering requests. Thus, in this case Constraints 12, 13, 14, 15, and 16 have no effect and the `MB-mp/MP` policy results in being ineffective.

This case does not improve the situation to the `baseline`, which always matches each interfered request with N_{pend} interfering requests (Constraint 4). When task 6's requests increases (rd_6 between 20% and 80%), the solver moves some interfered requests to other accounting windows to increase the interference (due to Constraint 12, 13, 14, 15, and 16), forcing the presence of more requests of τ_6 at high priority, which instead can be interfered with at most 1 interfering requests each (due to case (i) only), instead of N_{pend}. In this way, the interference decreases. However, when task 6's requests grow further (rd_6 over 80%), the interference gets again closer to the `baseline` case. For example, if $\beta_6^{\mathrm{M}} = 0.4$, this is because the number of interfered requests approaches the number of interfering requests, making it irrelevant whether they are matched in a one-to-one or one-to-N_{pend} fashion because the number of interfered requests is anyway enough to match all the interfering ones. In all the configurations in Fig. 3, all other tasks except τ_6 showed no improvement over the baseline.

Another aspect we investigated is the scalability of the proposed optimization problem with respect to the total amount of memory requests issued by the tasks. To better understand its run-time performance, we collected the execution times of the entire response-time analysis. The experiment was repeated by applying five different scaling factors $\xi_R, \xi_W \in \{0.2, 0.4, 0.6, 0.8, 1\}$, with $\xi_R = \xi_W$, to the number of requests of the original task set. As previously stated, with a high number of requests the task set is actually not schedulable; however, during this test, this aspect

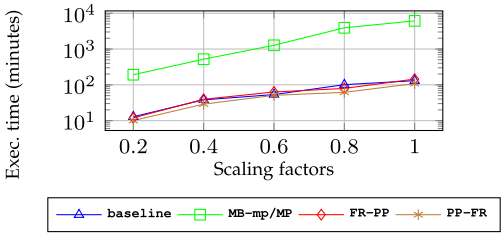

Fig. 4. Running times of the analyses when varying the scaling factors ξ_R and ξ_W, with $\xi_R = \xi_W$.

was taken into consideration. Fig. 4 reports the results, which show how the optimization problem is significantly slower when using the `MB-mp/MP` strategy (up to 102 hours of execution). On the other hand, both the `FR-PP` and the `PP-FR` priority partitioning techniques have total running times similar to the `baseline` approach (up to about 2.5 hours), which is compatible with standard design activities that take place off-line.

11 Related Work

The literature on memory bandwidth reservation mechanisms and methods to control and analyze contention is too vast to be exhaustively discussed within the space limits: therefore, a selection of the most closely related works is reported next.

One of the first proposals to regulate the memory bandwidth is due to Yun *et al.* [2] who implemented a per-core regulator called MemGuard, using performance counters to implement a budgeting mechanism. Farshchi *et al.* [29] proposed a custom hardware component, called BRU, to regulate the memory bandwidth of multiple cores at the same time. Sohal *et al.* [4] implemented a framework for analyzing the memory demand and to predict the timing of real-time workloads on CPUs and hardware accelerators. Several memory-bandwidth regulations mechanisms have been proposed also for hardware accelerators, such as GPUs or FPGAs [8], [9], [10]. Several techniques have been also proposed to improve predictability of cache memories: the interested reader can refer to the survey by Gracioli *et al.* [30]. Other authors [23], [31] proposed methods to reduce contention by adopting bank-aware memory allocations.

Several efforts have been spent in realizing custom memory controller designs to enhance predictability [14], [16], [32], [33], [34]. While being advisable designs in the context of real-time systems, they are not present in COTS platforms, thus strongly limiting their adoption. On the contrary, the MPAM specification has the potential of being present in all Arm platforms, and hence in billions of devices.

Another way of enhancing memory-access predictability is to enforce predictable execution models [11], [12], [35], [36] However, these schemes requires either hardware, OS, or compiler-level support, which may be not always available.

Finally, other works focused on bounding the memory delays in the presence of contention. The most close to our work are those considering DDR memories, i.e., those due to Kim *et al.* [23], Hassan *et al.* [18], [37], and Casini *et al.* [19].

Overall, no previous work studied the effects of the MPAM mechanisms on memory contention.

12 Conclusion

With Arm proposing the MPAM specification, COTS platforms have an unprecedented opportunity to improve their predictability on a large scale. In this paper, we deeply studied the MPAM specification by Arm, highlighting many points in which it is underspecified and leaves room for ambiguities. Then, we explored some possible instantiations of the specification at the level of the DRAM memory controller. For each of them, we derived a memory-contention analysis that has been used to compare different design alternatives.

Lesson Learned. From the extensive experimental evaluation we performed, it resulted in being easy to find configurations of the priority partitioning `FR-PP` and `PP-FR` configurations to foster predictability by reducing the memory contention delays, with improvements up to 41% to the baseline configuration. Conversely, it was hard to achieve reasonable improvements using the much more complex `MB-mp/MP` strategy, with improvements limited to 16% for very specific configurations. As it is common in real-time systems, this showed that *simpler is better* for predictability: `MB-mp/MP` exhibits a higher degree of dynamicity and more parameters to be properly configured, which unavoidably leads to a much more complex analysis and a more pessimistic analysis that arise from the consideration of a larger set of corner cases that cannot be excluded a priori. Nevertheless, we do not exclude the `MB-mp/MP` might lead to improvements in terms of average-case behavior. To assess this, future work will implement this policy into a state-of-the-art DRAM simulator, e.g., [38].

Other possible research directions include the consideration of the `HARDLIM` configuration of `MB-mp/MP` and the portion partitioning strategy.

References

[1] A. Biondi et al., "SPHERE: A multi-SoC architecture for next-generation cyber-physical systems based on heterogeneous platforms," *IEEE Access*, vol. 9, pp. 75446–75459, 2021.

[2] 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.*, 2013, pp. 55–64.

[3] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha, "Memory access control in multiprocessor for real-time systems with mixed criticality," in *Proc. IEEE 24th Euromicro Conf. on Real-Time Syst.*, 2012, pp. 299–308.

[4] P. Sohal, R. Tabish, U. Drepper, and R. Mancuso, "E-warP: A system-wide framework for memory bandwidth profiling and management," in *Proc. IEEE Real-Time Syst. Symp.*, 2020, pp. 345–357.

[5] M. Zini, G. Cicero, D. Casini, and A. Biondi, "Profiling and controlling I/O-related memory contention in COTS heterogeneous platforms," *Softw.: Pract. Experience*, vol. 52, pp. 1095–1113, 2021.

[6] A. Serrano-Cases, J. M. Reina, J. Abella, E. Mezzetti, and F. J. Cazorla, "Leveraging hardware QoS to control contention in the xilinx zynq UltraScale+ MPSoC," in *Proc. 33rd Euromicro Conf. Real-Time Syst.*, 2021, pp. 3:1–3:26.

[7] E. Betti, S. Bak, R. Pellizzoni, M. Caccamo, and L. Sha, "Real-time I/O management system with cots peripherals," *IEEE Trans. Comput.*, vol. 62, no. 1, pp. 45–58, Jan. 2013.

[8] N. Capodieci, R. Cavicchioli, P. Valente, and M. Bertogna, "SiGAMMA: Server based integrated GPU arbitration mechanism for memory accesses," in *Proc. 21st Int. Conf. Real-Time Netw. Syst.*, 2017, pp. 48–57.

[9] F. Restuccia, A. Biondi, M. Marinoni, and G. Buttazzo, "Safely preventing unbounded delays during bus transactions in FPGA-based SoC," in *Proc. IEEE 28th Annu. Int. Symp. Field-Programmable Custom Comput. Machines*, 2020, pp. 129–137.

[10] H. Aghilinasab, W. Ali, H. Yun, and R. Pellizzoni, "Dynamic memory bandwidth allocation for real-time GPU-based SoC platforms," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 39, no. 11, pp. 3348–3360, Nov. 2020.

[11] R. Pellizzoni et al., "A predictable execution model for cots-based embedded systems," in *Proc. IEEE 17th Real-Time Embedded Technol. Appl. Symp.*, 2011, pp. 269–279.

[12] G. Durrieu, M. Faugere, S. Girbal, D. G. Perez, C. Pagetti, and W. Puffitsch, "Predictable flight management system implementation on a multicore processor," in *Proc. Embedded Real Time Softw.*, 2014, pp. 1–9.

[13] F. Restuccia, A. Biondi, M. Marinoni, G. Cicero, and G. Buttazzo, "AXI HyperConnect: A predictable, hypervisor-level AXI interconnect for hardware accelerators in FPGA SoC," in *Proc. 57th ACM/ESDA/IEEE Des. Automat. Conf.*, 2020, pp. 1–6.

[14] B. Akesson, K. Goossens, and M. Ringhofer, "Predator: A predictable SDRAM memory controller," in *Proc. 5th IEEE/ACM Int. Conf. Hardware/Softw. CoDes. System Synth.*, 2007, pp. 251–256.

[15] B. Akesson and K. Goossens, "Architectures and modeling of predictable memory controllers for improved system integration," in *Proc. Des. Automat. Test Europe*, 2011, pp. 1–6.

[16] R. Mirosanlou, M. Hassan, and R. Pellizzoni, "DRAMbulism: Balancing performance and predictability through dynamic pipelining," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp.*, 2020, pp. 82–94.

[17] "Arm architecture reference manual supplement memory system resource partitioning and monitoring (MPAM) for Armv8-A." [Online]. Available: https://developer.arm.com/documentation/ddi0598/latest

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

[19] 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 26th Real-Time Embedded Technol. Appl. Symp.*, 2020, pp. 239–252.

[20] Intel, "External memory interface handbook volume 2: Design guidelines." [Online]. Available: https://www.intel.com/programmable/technical-pdfs/683385.pdf

[21] Qualcomm, "Qualcomm snapdragon 600e processor apq8064e recommended memory controller and device settings application note." [Online]. Available: https://developer.qualcomm.com/qfile/28875/lm80-p0598-6_b_apq8064_recmemcontrollerdevicesettingsappnote.pdf

[22] T. Instruments, "Keystone architecture DDR3 memory controller." [Online]. Available: https://www.ti.com/lit/ug/sprugv8e/sprugv8e.pdf?ts=1662395987524&ref_url=https%253A%252F%252Fwww.google.com%252F

[23] 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.*, 2014, pp. 145–154.

[24] H. Yun, R. Pellizzoni, and P. K. Valsan, "Parallelism-aware memory interference delay analysis for COTS multicore systems," in *Proc. IEEE 27th Euromicro Conf. Real-Time Syst.*, 2015, pp. 184–195.

[25] A. Biondi and M. D. Natale, "Achieving predictable multicore execution of automotive applications using the let paradigm," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp.*, 2018, pp. 240–250.

[26] D. Casini, P. Pazzaglia, A. Biondi, and M. D. Natale, "Optimized partitioning and priority assignment of real-time applications on heterogeneous platforms with hardware acceleration," *J. Syst. Architecture*, vol. 124, 2020, Art. no. 102416.

[27] A. Hamann, D. Dasari, F. Wurst, I. Sañudo, N. Capodieci, P. Burgio, and M. Bertogna, WATERS Industrial Challenge, 2019. [Online]. Available: https://retis.sssup.it/~d.casini/resources/WATERS2019/WATERS_Industrial_Challenge_2019_final.pdf

[28] D. Casini, P. Pazzaglia, A. Biondi, G. Buttazzo, and M. Di Natale, "Addressing analysis and partitioning issues for the Waters 2019 challenge," in *Proc. 10th Int. Workshop Anal. Tools Methodol. Embedded Real-Time Syst.*, 2019, pp. 1–6.

[29] 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.*, 2020, pp. 364–375.
[30] G. Gracioli, A. Alhammad, R. Mancuso, A. A. Fröhlich, and R. Pellizzoni, "A survey on cache management mechanisms for real-time embedded systems," 2015, Art. no. 1–36.
[31] H. Yun, R. Mancuso, Z. 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.*, 2014, pp. 155–166.
[32] M. Paolieri, E. Quinones, F. J. Cazorla, and M. Valero, "An analyzable memory controller for hard real-time CMPs," *IEEE Embedded Syst. Lett.*, vol. 1, no. 4, pp. 86–90, Dec. 2009.
[33] M. Hassan, H. Patel, and R. Pellizzoni, "PMC: A requirement-aware dram controller for multicore mixed criticality systems," vol. 16, no. 4, 2017, Art. no. 100.
[34] D. Guo, M. Hassan, R. Pellizzoni, andH. Patel, "A comparative study of predictable dram controllers," *ACM Trans. Embedded Comput. Syst.*, vol. 17, no. 2, pp. 1–23, 2018.
[35] D. Casini, P. Pazzaglia, A. Biondi, M. Di Natale, and G. Buttazzo, "Predictable memory-CPU co-scheduling with support for latency-sensitive tasks," in *Proc. IEEE 57th Des. Automat. Conf.*, 2020. Art. no. 67.
[36] 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, pp. 850–888, 2019.
[37] M. Hassan and R. Pellizzoni, "Analysis of memory-contention in heterogeneous COTS MPSoCs," in *Proc. 32nd Euromicro Conf. Real-Time Syst.*, 2020, pp. 23:1–23:24.
[38] 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.

Matteo Zini received the graduate (cum laude) degree in embedded computing systems engineering, the master's degree jointly offered by the Scuola Superiore Sant'Anna of Pisa and University of Pisa. He is currently working toward the PhD degree with the Real-Time Systems (ReTiS) Laboratory of the Scuola Superiore Sant'Anna of Pisa. His research interests include the design and implementation of real-time operating systems and hypervisors, cyber-physical systems, and analysis of the effects of memory interference on schedulability.

Daniel Casini (Member, IEEE) received the graduate (cum laude) degree in embedded computing systems engineering, the master's degree jointly offered by the Scuola Superiore Sant'Anna of Pisa and University of Pisa, and the PhD degree in computer engineering from the Scuola Superiore Sant'Anna of Pisa (with honors), working under the supervision of Prof. Alessandro Biondi and Prof. Giorgio Buttazzo. He is assistant professor with the Real-Time Systems (ReTiS) Laboratory of the Scuola Superiore Sant'Anna of Pisa. In 2019, he has been visiting scholar with the Max Planck Institute for Software Systems (Germany). His research interests include software predictability in multi-processor systems, schedulability analysis, synchronization protocols, and the design and implementation of real-time operating systems and hypervisors.

Alessandro Biondi (Member, IEEE) received the graduate (cum laude) degree in computer engineering from the University of Pisa, Italy, within the excellence program, and the PhD degree in computer engineering from the Scuola Superiore Sant'Anna under the supervision of Prof. Giorgio Buttazzo and Prof. Marco Di Natale. He is associate professor with the Real-Time Systems (ReTiS) Laboratory of the Scuola Superiore Sant'Anna. In 2016, he has been visiting scholar with the Max Planck Institute for Software Systems (Germany). His research interests include design and implementation of real-time operating systems and hypervisors, schedulability analysis, cyber-physical systems, synchronization protocols, and component-based design for real-time multiprocessor systems. He was recipient of six Best Paper Awards, one Outstanding Paper Award, the ACM SIGBED Early Career Award 2019, and the EDAA Dissertation Award 2017.

Open Access funding provided by 'Scuola Superiore 'S.Anna' di Studi Universitari e di Perfezionamento' within the CRUI CARE Agreement