Bounding DRAM Interference in COTS Heterogeneous MPSoCs for Mixed Criticality Systems

Mohamed Hassan, *Member, IEEE*, and Rodolfo Pellizzoni, *Member, IEEE*

***Abstract*—Commercial off-the-shelf (COTS) heterogeneous multiple processors systems-on-chip (MPSoCs) are appealing platforms for emerging mixed criticality systems (MCSs). To satisfy MCS requirements, the platform must guarantee predictable timing bounds for critical applications, without degrading average performance for noncritical applications. In particular, this paper studies the main memory subsystem, which in modern MPSoCs is typically based on double data rate synchronous dynamic access memory. While there exists previous work on worst-case DRAM latency analysis, such work only covers a small subset of possible COTS configurations, which are not targeted at MCS. Therefore, we derive a generalized interference delay analysis for DRAM main memory that accounts for a breadth of features deployed in COTS platforms. We then explore the design space by studying the effects of each feature on both the worst-case delay for critical applications, and the bandwidth for noncritical applications.**

***Index Terms*—DRAM, memory, heterogeneous systems, mixed criticality, multiple processors systems-on-chip (MPSoCs), real-time systems, system-on-chip, timing analysis.**

I. Introduction

Many emerging embedded systems consist of tasks with different criticalities, known as mixed criticality systems (MCSs). Examples include self-driving cars, smart grids, and healthcare devices [1]. A major challenge in MCS is the different needs of applications with varying criticalities: critical applications require guaranteed, predictable timing bounds on worst-case execution time, while noncritical applications demand good average time performance. This places conflicting requirements on the hardware platform. Taking the memory subsystem as an example, critical tasks are latency sensitive; thus, they require guaranteed bounds on the maximum latency suffered by memory accesses. In contrast, noncritical tasks opportunistically attempt to achieve as high memory bandwidth (BW) as possible to improve their average-case performance.

Due to their cost, performance, and energy efficiency, multiple processors systems-on-chip (MPSoCs) provide appealing platforms for MCS. In particular, heterogeneous MPSoCs with diverse processing element (PE) types can efficiently address the conflicting requirements of MCS; for example, some existing commercial off-the-shelf (COTS) platforms [2] already include a set of real-time PEs, optimized for predictable timing, as well as a set of high-performance PEs targeted at noncritical applications. However, careful design choices among available COTS platforms, as well as rigorous timing analysis, are both essential to guarantee satisfaction of MCS requirements.

In this paper, we study the main memory subsystem, which in modern MCS is typically based on double data rate synchronous dynamic access memory technology [3]. Many research efforts have focused on providing DRAM solutions for MCS; nonetheless, most of them have adopted the approach of designing a custom memory controller (MC) to provide tight analytical latency bounds [4]–[8], and thus are not compatible with COTS platforms. Recently, two related works have developed latency analyses for COTS DRAM platforms [9], [10]. However, both works cover a small subset of possible COTS configurations, which are not targeted at MCS. For instance, they do not account for features such as heterogeneity in PEs, or different priority levels in the DRAM scheduler, which can be used to provide differentiated service to critical and noncritical applications.

Toward this end, this paper explores a breadth of features deployed in COTS platforms that affect the memory performance of MCS applications both from worst-case delay (WCD) and average-case BW perspectives. The feature set covers heterogeneity in the PEs, operating system (OS)-level memory partitioning, and request scheduling mechanisms at the MC level. Based on this feature model, we explore a set of 144 possible COTS platform instances. To guarantee WCD bounds, we derive a generalized interference delay analysis for DRAM main memory that computes the maximum delay that any memory request from a critical application can suffer due to the activity of other PEs. In addition, we study the effect of each platform instance on the BW provided to noncritical applications through an extensive set of architectural simulations.

Manuscript received April 3, 2018; revised June 8, 2018; accepted July 2, 2018. Date of current version October 18, 2018. This article was presented in the International Conference on Embedded Software 2018 and appears as part of the ESWEEK-TCAD special issue. *(Corresponding author: Mohamed Hassan.)*

M. Hassan was with Intel PSG, Toronto, ON M5S 2X9, Canada. He is now with the University of Guelph, Guelph, ON, Canada (e-mail: mohamed.hassan@uoguelph.ca).

R. Pellizzoni is with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON N2L 3G1, Canada (e-mail: rpellizz@uwaterloo.ca).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2018.2857379

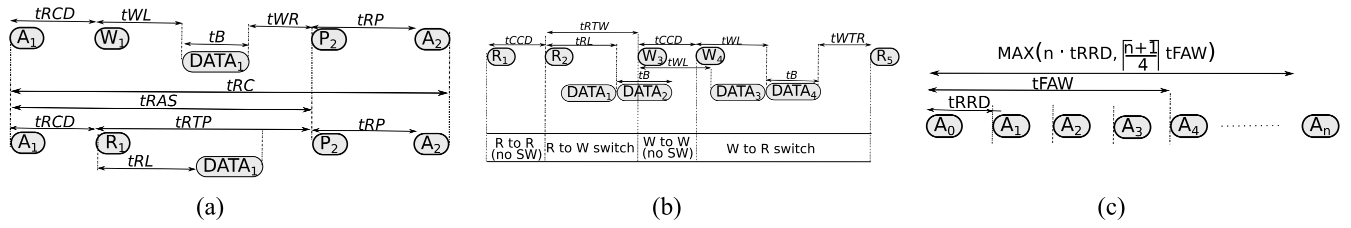


Fig. 1. DRAM timing constraints. (a) Intrabank timing constraints. (b) Interbank CAS timing constraints. (c) Interbank A timing constraints.

The rest of this paper is organized as follows. Section II recaps the operation of DRAM, and Section III reviews related work. Section IV presents the analyzed system features. Sections V and VI contain our main contribution, the derivation of a generalized DRAM delay analysis; Section V introduces the basic building blocks, and Section VI applies them to the varying configuration. Finally, Section VII details our evaluation, and Section VIII provides concluding remarks.

II. Background on DRAM

A DRAM device is an array of memory cells consisting of *banks*, with each bank organized by *rows* and *columns*. A DRAM *rank* consists of multiple banks, and a DRAM *channel* has one or more ranks. Each bank has *a row buffer*, which acts as a cache for the memory array and is able to store a single row. An on-chip MC manages requests from multiple PEs to the off-chip DRAM device. The MC controls the device by issuing DRAM *commands* through a dedicated command bus; one command can be issued per clock cycle. The request data is exchanged between the MC and the device through a separate data bus.

Each request consists of an address and a type, either read or write. The address is mapped by the MC to a specific bank, row and column. If multiple concurrent requests arrive at the MC, an arbiter decides the next request to service. To service a request, the MC issues a set of DRAM commands based on the state of the device: P, A, and CAS.

The P command (*precharge*) stores the data in the row buffer back to the DRAM cells. The A command *activates* a row by fetching it from the DRAM cells to the row buffer. Finally, the CAS command *reads* (R) or *writes* (W) the required column of data from the row buffer. In details, there are three possible scenarios for a DRAM request.

1) If the requested row is already available in the row buffer, the request consists of only a CAS command.
2) The requested bank is idle (i.e., does not have an activated row in the buffer). In this case, the MC issues an A command first to activate the row, followed by a CAS command.
3) The requested row is different from the activated row in the row buffer (a *bank conflict*). In this case, the MC issues all three commands: a) P to precharge the old row; b) A to activate the requested row; and c) CAS to read/write.

The JEDEC DRAM standard [3] defines a set of timing constraints on the three commands that must be satisfied by all MC designs; the value of each constraint depends on the specific DRAM device type and speed. Table I list all constraints for a DDR3 device with one rank; we also show the value of the constraints for the particular device speed we use in the evaluation. Each constraint represents the minimum number of clock cycles that must elapse between the transmission of a command or data and a successive command or data; with the exception of *tFAW*, which represents the minimum distance between four, rather than two, consecutive A commands. We also distinguish between two types of constraints: *intrabank constraints* are applied between data/commands issued to the same bank, while *interbank constraints* are applied between data/commands of the same type (P, A, or CAS) issued to any bank. For ease of exposition, Fig. 1(a) depicts the considered intrabank constraints on the example of consecutive bank conflicts. Fig. 1(b) and (c) illustrates the considered interbank constraints for CAS and A, respectively. There are no interbank constraints for P.

TABLE I
JEDEC Timing Constraints for DDR3-1333H [3]

Parameters	Description	cycles
	Intra-bank constraints	
$tRCD$	ACT to CAS delay	9
tRL	RD to Data Start	9
tRP	PRE to ACT Delay	9
tWL	WR to Data Start	8
$tRAS$	ACT to PRE Delay	24
tRC	ACT to ACT (same bank)	33
tWR	Data End of WR to PRE	10
$tRTP$	Read to PRE Delay	5
	Inter-bank constraints	
$tCCD$	CAS to CAS delay	4
$tRTW$	RD to WR Delay	6
$tWTR$	WR to RD Delay	5
$tRRD$	ACT to ACT (diff bank in same rank)	4
tB	Data bus transfer	4
$tFAW$	Four bank activation window	20

For clarity, in the rest of this paper we will say that a command (as well as the corresponding request) is *intraready* at the current clock cycle if it satisfies all intrabank timing constraints. If the command satisfies all interbank constraints, we call it *inter-ready*. Of course, a command can only be issued if it is both intraready and inter-ready, in which case we say that the command is *ready*. Arbitration between different requests usually follows a first ready-first come first serve (FR-FCFS) scheduling scheme [9]. Under FR-FCFS, requests are first queued into per-bank queues. Within each bank (intrabank arbitration), FR-FCFS prioritizes requests with locality (that target data already available in a row buffer) to increase DRAM throughput. As discussed above, such requests consist of a CAS only, while requests that target non activated rows could consist of all three commands.

Finally, note that the DRAM has to be refreshed periodically to retain stored data. This is achieved by the MC by issuing REF (*refresh*) commands. Similar to previous work [9], [10], we do not account for the delay from the refresh process because it can be often neglected compared to other delays [9], or otherwise, it can be added as an extra delay term to the execution time of a task using existing methods [11], [12].

III. Related Work

Due to its potential impact on predictability and performance, many research efforts in the real-time community have recently focused on managing contention for access to DRAM resources. We distinguish between four main lines of research. First, a large body of work has proposed new MC designs to provide better WCD bounds (see [4] for an overview); some of these works can also support MCS by providing differentiated service to critical and noncritical requestors, e.g., [5]–[8] and [13]. However, since they require a redesign of the MC, they cannot be applied to existing COTS platforms.

A second line of work has focused on OS-level mechanisms to reduce contention in main memory. The approaches in [14]–[16] use virtual memory to partition memory resources among PEs; particularly, interference can be reduced by partitioning banks among the PEs, since a row activated by one PE cannot be precharged due to a request of a different PE. We discuss bank partitioning in Section IV-C. The approach in [17] further limits contention by regulating the maximum amount of memory BW that can be consumed by any individual PE. Third, several works have proposed analytical approaches to bound the delay due to shared memory contention on MPSoCs (see [18]–[22]), by either relying on OS-level mechanisms [20] or information on task structure and scheduling [18], [19], [21], [22]. However, the considered memory models are simple, and cannot accurately capture the behavior of complex arbitration schemes such as FR-FCFS that are employed in COTS MCs to optimize DRAM throughput.

Finally, two previous papers have discussed latency bounds for COTS MCs [9], [10], and are most related to this paper. Reference [9] bounds memory access latencies, while assuming that all PEs are in-order cores. Reference [10], on the other hand, considers out-of-order PEs but it is restricted to systems using bank partitioning. We find two main issues with these approaches, which in turn motivate this paper.

1) *Lack of Support for Different Criticalities:* Both works consider systems where all tasks have the same criticality, which makes them ill-suited for MCS where both critical and noncritical applications are running on the same platform.
2) *Limited Applicability:* Each cited work assumes a specific platform instance with a certain MC architecture and OS configuration. Accordingly, the derived bounds cannot be widely applied to different COTS platforms. In contrast, in this paper, we derive a generalized DRAM delay analysis that can be applied to a variety of COTS systems.

IV. System Model

We consider an MCS comprising P PEs. Similar to previous work [10], we assume that the last-level-cache of all PEs employs a write-back write-allocate policy, such that a write request to DRAM occurs only because of cache eviction of a modified cache line. Additionally, we conduct the exploration for a single-channel single-rank DRAM subsystem, and we denote with N_B the number of banks. Each PE can execute any number of applications, but we assume that all applications mapped to a given PE are either critical or noncritical; we find this to be a common requirement due to considerations of reliability, fault containment, and error handling. Hence, we classify the PEs between a set of P_{cr} *critical* PEs and set of P_{ncr} *noncritical* PEs, with $P = P_{\text{cr}} + P_{\text{ncr}}$. Our main goal is to derive an upper bound on the delay incurred by any memory request of a critical PE under analysis, due to interference caused by memory requests of other PEs. As discussed in [9], [12], and [20], this bound can then be used to either derive the worst-case execution time of single real-time task, or to perform response-time analysis for a multitasking application. Following the same approach as in [9] and [10], we make no assumption on the pattern of computation and memory requests of the PE under analysis, nor of interfering PEs. Additional information on the behavior of tasks and the scheduling algorithm could result in a tighter bound (see [18], [19], [21], [22]), albeit at the cost of compositionality in the analysis. On the other hand, we consider in details the memory behavior of a COTS platform, which depends on the PE architecture, the MC arbitration, and the OS configuration. Hence, in the rest of this section we detail the considered hardware and software features.

A. Memory Controller

We consider alternatives in the way the MC arbitrates among memory requests.

1) Priority: Prior solutions [9], [10] consider a standard FR-FCFS implementation which does not distinguish among PEs. However, modern MCs such as Qualcomm's [23] and Intel's [24] MCs support priority assignments to PEs. We can leverage this feature in commodity MCs by assigning higher priority to requests of critical PEs over requests of noncritical PEs, thus lowering latency bounds for critical applications without the need for hardware changes. For the sake of inclusiveness, we also consider configurations without PE prioritization to match the assumption of previous works, and to cover COTS MCs that do not support this feature.

2) Intrabank Arbitration: As discussed in Section II, FR-FCFS favors intraready requests over non intraready ones. To avoid starving the latter, MCs typically deploy a thresholding mechanism on top of FR-FCFS [9], [24]–[26]. This mechanism can be implemented with various flavors; we assume the implementation considered in [9] (which is also similar to Intel's [24]), where at most N_{thr} intraready requests can be reordered ahead of any other request targeting the same bank. We also cover MCs that do not implement any reordering threshold.

3) Interbank Arbitration: Based on several prior works and patents (e.g., [14] and [27]–[29]), and industry expertise, we assume that the MC deploys a round-Robin (RR) mechanism among banks. In details, the considered RR scheduler selects between intraready commands at the head of the bank queues; when the last command of a request (CAS) is executed, then the corresponding bank is moved to the back of the RR queue. Two possible implementations of RR with regard to interbank constraints are considered.

a) Reordering among all commands: If the scheduled command is not inter-ready, the MC selects the first bank (in the RR schedule) with an inter-ready command whether it is the same type as the stalling command or not. For instance, if the scheduled command is W and it is not inter-ready due to the R-to-W switching constraint, the MC can select an inter-ready R command at the head of another bank. In other words, the MC allows for command reordering among intraready commands across banks regardless of the command type (whether it is P, CAS, and A). Although this aggressive optimization targets performance gains, it can result in unbounded delays in some cases as we show in Section VI.

b) Reordering among commands of different types: The second implementation differs from the previous one in allowing for reordering only across commands of different types. For instance, if the command scheduled by RR is A and it is not inter-ready, it does not stall ready CAS commands of other banks. This implementation still achieves a performance optimization, while preventing the unboundedness of the previous one. For the sake of comprehensiveness, we cover both implementations.

4) Read/Write Arbitration: Some COTS MCs prioritize reads over writes because read accesses are more latency sensitive than write accesses [10], [30]. They queue write accesses into a dedicated write buffer and service them in batches. This mechanism mitigates the turn-around time required to switch the data bus between reads and writes. This is because the controller will be servicing batches of same type (either consecutive reads or writes) instead of alternating between both types. Accordingly, this mechanism can significantly improve memory throughput. In our exploration, we cover both cases: 1) the MC assigns same priority for both reads and writes and 2) the MC employs write batching where it prioritizes reads over writes. Since write batching can be implemented in different ways, for the latter case we consider the watermarking implementation adopted by related work [10], which is similar to the one adopted by COTS MCs [23]: the MC services a batch of W_{btch} writes when the number of buffered writes exceeds a given threshold.

Since writes are generated by the cache due to a write-back of an evicted cache block, we assume that in the worst case, each read request can generate a maximum of one write request.

B. Processing Elements

We consider PE architectures with the following configurations.

1) In-Order PEs (IO-All): All PEs are in-order, where each PE has at most one pending memory request at any given time. This is the architecture assumed by the analysis in [9].

2) Out-of-Order PEs (OOO-All): All PEs are out-of-order. The maximum number of outstanding requests in the MC queues for each PE is bounded by an architectural-dependant constant PR. This is the architecture considered by [10]; here, the value of PR depends on the architecture, in particular on the number of entries in the miss status handling register (MSHR) of the PE; the MSHR keeps track of the cache misses which are currently being handled. For platforms without write batching, PR accounts for both reads and writes. For platforms with write batching, since reads and writes are buffered in separate queues, we consider PR as the maximum number of pending reads only.

3) Mix of In- and Out-of-Order PEs (IO-Cr): The previous configurations do not take the heterogeneous nature of the architecture or the mixed criticality nature of the applications into account. Hence, we also consider COTS platforms that have a mix of in-order and out-of-order PEs, where critical PEs are in-order to achieve high predictability, while noncritical PEs are OOO to achieve high average performance. One example of such platforms is Xilinx Zynq UltraScale+ [2], which comprises two Cortex R5 real-time cores and four Cortex A53 high-performance cores.

C. Bank Partitioning

As previously mentioned, some prior work (e.g., [6]–[8] and [12]) considers DRAM bank partitioning, where banks are partitioned among PEs to reduce bank conflicts and hence improve WCD. Partitioning can be conducted by changing the virtual to physical address translation in the OS [14]–[16]; hence, it can be applied to COTS MCs without hardware changes. We denote with *Part-All* the case where partitioning is applied for all PEs, which is the configuration analyzed in [10] and [20]. The total number of banks assigned to critical and noncritical PEs is denoted as N_{Bcr} and N_{Bncr}, respectively, such that $N_B = N_{\mathrm{Bcr}} + N_{\mathrm{Bncr}}$. On the other hand, we denote with *No-Part* the case where all N_B banks are shared among PEs, so that requests are interleaved over all banks. In addition to these two options, we cover a third alternative, denoted as *Part-Cr*, which we argue is suitable for MCS: here, bank partitioning is used for critical applications, the N_B banks are partitioned among critical PEs, so that no two critical PEs share the same banks. However, noncritical PEs share banks among each other and with critical PEs, and their requests are interleaved across all banks. The intuition behind this novel scheme is as follows.

1) Critical applications care more about WCD; hence, it is okay to suffer an average performance penalty by not interleaving requests of critical PEs, given that partitioning reduces worst-case interference. In contrast, interleaving is important for noncritical applications since they usually require a high average-case BW.

TABLE II
System Model Symbols

Symbol	Description	Instances
P	Number of PEs	all
P_{cr}	Number of critical PEs	all
P_{ncr}	Number of non-critical PEs	all
N_B	Number of DRAM banks	all
N_{Bcr}	Number of banks assigned to critical PEs	$part = Part\text{-}All$
N_{Bncr}	Number of banks assigned to non-critical PEs	$part = Part\text{-}All$
N_{thr}	Intra-bank reorder threshold	$thr = 1$
W_{btch}	Write batch length	$wb = 1$
PR	Number of outstanding requests	$pipe \neq IO\text{-}All$

2) As we will show in Section VI, by leveraging PE prioritization, the interference of noncritical PEs sharing a bank with a critical PE is extremely diminished.

D. Platform Instances

Based on the discussed features, we represent each COTS platform instance with a tuple: $\langle wb, thr, pr, breorder, pipe, part\rangle$, where:

1) $wb \in \{0, 1\}$ equals 1 if a write batching (read/write arbitration) mechanism is employed in the system, and 0 otherwise;
2) $thr \in \{0, 1\}$ indicates whether the MC implements a threshold mechanism on FR-FCFS arbitration to prevent starvation ($thr = 1$) or not ($thr = 0$);
3) $pr \in \{0, 1\}$, indicates whether the MC adopts a fixed priority assignment among PEs ($pr = 1$) or not ($pr = 0$);
4) $breorder \in \{0, 1\}$, indicates whether the MC adopts interbank reordering across all commands ($breorder = 1$) or only across commands of different types ($breorder = 0$);
5) $pipe \in \{IO\text{-}All, IO\text{-}Cr, OOO\text{-}All\}$ denotes the pipeline architecture of the PEs;
6) $part \in \{No\text{-}Part, Part\text{-}Cr, Part\text{-}All\}$ indicates the aforementioned three bank partitioning schemes.

Overall, this results in 144 different platform instances. Each instance is further characterized by a subset of the parameters introduced in this section, which we summarize in Table II.

V. Memory Delay Building Blocks

We now focus on computing a WCD bound for a request under analysis r_{ua} of a critical PE under analysis PE_{ua}. For instances with $wb = 0$, we make no assumption on the type of the request (R/W). For instances with $wb = 1$, we assume that r_{ua} is a read request; write batching is typically employed in architectures where writes do not stall the pipeline of PEs, and hence the delay suffered by write requests does not affect the execution time of applications. Write batching seeks to prevent the write queue from getting full by switching to servicing writes immediately once a threshold is exceeded.[1]

Our approach works as follows: we consider all timing constraints generated by commands of interfering requests of other PEs serviced between the times when r_{ua} arrives and finishes being processed, and all clock cycles where a command cannot be issued due to command bus contention (if multiple commands are ready at the same time, only one can be issued per clock cycle). We then sum them all to obtain the delay bound, after creating a pattern of commands that maximizes the sum. In particular, we assume that the request under analysis suffers a bank conflict, so that it must issue all three commands in sequence: P, A, and CAS. Since the next command in a sequence is issued by the MC when all related timing constraints have elapsed and the command bus is free, the obtained sum is a valid WCD bound. However, the bound could be pessimistic, since in reality, multiple timing constraints could be simultaneously active. For example, consider the time between an A command, and the next A command to the same bank [note this is represented in Fig. 1(a)]. Based on the timing constraints in Table I, the two commands must be separated by tRC. However, note that a CAS, data, and P command must also be issued in between, since a bank must be precharged before being activated again. Hence, based on the table, the two A commands must also be separated by $tRAS + tRP$, and either $tRCD + tWL + tB + tWR + tRP$ for a write or $tRCD + tRTP + tRP$ for a read. In summary, rather than summing all three terms together, we can derive the delay between two consecutive A commands to the same bank as: $\text{MAX}(tRC, \text{MAX}(tRAS, tRCD + tWL + tB + tWR) + tRP)$ for a write, or as: $\text{MAX}(tRC, \text{MAX}(tRAS, tRCD + tRTP) + tRP)$ for a read. Since $tRC = tRAS + tRP$ and $tRTP < tWL + tB + tWR$ for all DDR protocols, the delay between consecutive A commands to the same bank can be simplified to $\text{MAX}(tRAS, tRCD + tWL + tB + tWR) + tRP$.

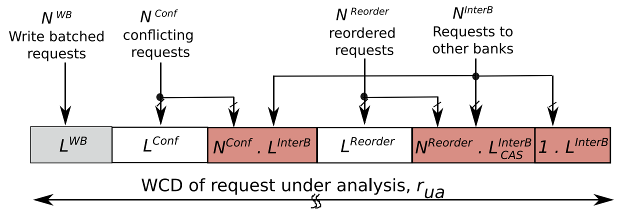


Fig. 2. Analysis approach: number of interfering requests and delay components.

We next discuss how to systematically consider the effects of all interfering requests. Our approach is depicted in Fig. 2, which also summarizes the employed notation: we assume that the maximum number of interfering requests is known; based on such number, in this section, we compute bounds on the maximum delay caused by the interfering requests. We will then show how to derive the number of interfering requests in Section VI, since it depends on the platform instance. More in details, we categorize the interfering requests and associated delay components as follows.

1) For instances with $wb = 1$, N^{WB} represents the maximum number of interfering writes; correspondingly, $L^{WB}(N^{WB})$ represents the maximum delay caused by the N^{WB} writes issued in batches. This is the first block in Fig. 2. The remaining delay components L^{Conf}, $L^{Reorder}$, and L^{InterB} consider the interfering read requests. For

[1]We also assume that the write buffer is large enough to prevent it from becoming full, which would stall the PE. This also aligns with previous work that only considers configurations with write batching [10].

instances with $wb = 0$, the L^{WB} delay is not present since writes are not issued in batches. Instead, the remaining delay components consider both read and writes requests.

2) N^{Conf} is the number of interfering requests of other PEs targeting the same bank as r_{ua}, which are serviced ahead of r_{ua} because they arrived before it. To maximize the corresponding conflict interference delay $L^{Conf}(N^{Conf})$, we assume that all such requests target different rows, resulting in bank conflicts. Hence, each request executes a sequence of P, A, and CAS command. This is the second block in Fig. 2.

3) $N^{Reorder}$ is the number of interfering requests of other PEs targeting the same bank as r_{ua}, and which arrived after r_{ua} but are reordered ahead of r_{ua} due to first-ready intrabank arbitration. As discussed in Section II, such requests comprise a CAS only. Correspondingly, $L^{Reorder}(N^{Reorder}, wb)$ is the intrabank reordering interference delay. The third block in Fig. 2 represents this case.

4) N^{InterB} is the number of interfering requests of other PEs targeting a different bank than r_{ua}, which can interfere with each of the $N^{Conf} + N^{Reorder} + 1$ requests that target the same bank as r_{ua} (including r_{ua} itself). $L^{InterB}(N^{InterB}, wb)$ is the corresponding interbank delay caused on a request comprising a P, A, and CAS (third and last blocks in Fig. 2), while $L^{InterB}_{CAS}(N^{InterB}, wb)$ is the delay on a request consisting of a CAS only (fifth block in Fig. 2). Therefore, the total interbank delay is: $(N^{Conf} + 1) \cdot L^{InterB}(N^{InterB}, wb) + N^{Reorder} \cdot L^{InterB}_{CAS}(N^{InterB}, wb)$.

Based on the discussion above and assuming that the four delay bounds $L^{WB}, L^{Conf}, L^{Reorder}$, and L^{InterB} are correctly computed, we have thus obtained the following result.

Lemma 1: The WCD that r_{ua} suffers due to interference from other PEs is bounded by

$$wb \cdot L^{WB}\left(N^{WB}\right) + L^{Conf}\left(N^{Conf}\right) + L^{reorder}\left(N^{Reorder}, wb\right) + \left(N^{Conf} + 1\right) \cdot L^{InterB}\left(N^{InterB}, wb\right) + N^{Reorder} \cdot L^{InterB}_{CAS}\left(N^{InterB}, wb\right). \quad (1)$$

A. Write Batching Delay L^{WB}

In general, we cannot make any assumption on which banks are targeted by the interfering writes; hence, in the worst case we assume that all interfering writes cause bank conflicts. Since bank conflict requests consist of all three commands, we can determine the delay introduced by each bank conflict write as the time between two successive A commands to the same bank, which we previously computed. Hence, we obtain

$$L^{WB}\left(N^{WB}\right) = N^{WB} \cdot (\text{MAX}(tRAS, tRCD + tWL + tB + tWR) + tRP) \quad (2)$$

B. Conflict Interference Delay L^{Conf}

As previously discussed, in the worst case all N^{Conf} interfering requests can cause bank conflict. Equation (3) computes the total conflict interference delay

$$L^{Conf}\left(N^{Conf}\right) = N^{Conf} \cdot (\text{MAX}(tRAS, tRCD + tWL + tB + tWR) + tRP). \quad (3)$$

C. CAS Latency

The remaining delay components depend on the latency of executing some number N^{CAS} of consecutive CAS commands, which we hence denote as L^{CAS} and compute next. We consider two cases.

1) If write batching is employed, e.g., $wb = 1$, then all requests (outside of already considered write batches) are R. Situation between R1 and R2 in Fig. 1(b) represents this case; since the only interbank timing constraint between two successive R is $tCCD$ as follows:

$$L^{CAS}\left(N^{CAS}, wb = 1\right) = N^{CAS} \cdot tCCD. \quad (4)$$

2) Without write batching, e.g., $wb = 0$, in the worst case requests can alternate types between R and W. This forces the data bus to switch direction, introducing additional timing constraints: each R request suffers from the W-to-R latency [between W4 and R5 in Fig. 1(b)], while each W request suffers from the R-to-W latency [between R2 and W3 in Fig. 1(b)]. Since furthermore the W-to-R latency is always greater than the R-to-W latency for all DRAM devices, the total latency can be as calculated as

$$L^{CAS}\left(N^{CAS}, wb = 0\right) = \left\lceil \frac{N^{CAS}}{2} \right\rceil \cdot (tWL + tB + tWTR) + \left\lfloor \frac{N^{CAS}}{2} \right\rfloor \cdot (tRTW). \quad (5)$$

D. Intrabank Reordering Interference $L^{Reorder}$

As discussed, all $N^{Reorder}$ requests are only CAS commands. Hence, the delay due to intrabank reordering is upper bounded in

$$L^{Reorder}\left(N^{Reorder}, wb\right) = L^{CAS}\left(N^{Reorder}, wb\right). \quad (6)$$

E. Interbank Interference Delay L^{InterB}

We next compute the maximum delay that a request r can suffer due to timing constraints and command bus contention caused by N^{InterB} requests targeting other banks. Remember that interbank timing constraints are applied between commands of the same type. One possible strategy, used in [9], would be to compute the maximum delays L^{InterB}_{PRE}, L^{InterB}_{ACT}, and L^{InterB}_{CAS} suffered by each command of request r, and then sum them together to obtain $L^{InterB} = L^{InterB}_{PRE} + L^{InterB}_{ACT} + L^{InterB}_{CAS}$. However, as shown in [10], this bound is highly pessimistic: since both r_{ua} and the N^{InterB} interfering requests execute the P, A, and CAS commands in the same sequence, the timing constraints are effectively pipelined over three stages. We can take advantage of the pipelining effect using the following theorem, which is formally proven in [10].

Theorem 1 ([10, Th. 1]): Assume that the relative priorities of executed requests remains the same over their P, A, and CAS commands. Then the delay caused by an interfering interbank request to the request under analysis is upper bounded by the maximum delay on a single state, i.e., either the delay caused by the interfering P command to the P command under analysis, or A to A, or CAS to CAS.

Based on Theorem 1, we can obtain the interbank delay by maximizing the delay caused by each of N^{InterB} interfering requests, assuming that each request interferes on only one command

$$L^{InterB}\left(N^{InterB}, wb\right) = \max_{\forall N_{PRE}^{InterB}+N_{ACT}^{InterB}+N_{CAS}^{InterB}\in\mathbb{N}} \left(L_{PRE}^{InterB}\left(N_{PRE}^{InterB}\right) + L_{ACT}^{InterB}\left(N_{ACT}^{InterB}\right) + L_{CAS}^{InterB}\left(N_{CAS}^{InterB}, wb\right)\right)$$
$$\text{where: } N_{PRE}^{InterB} + N_{ACT}^{InterB} + N_{CAS}^{InterB} = N^{InterB}. \quad (7)$$

Here, $N_{PRE}^{InterB}, N_{ACT}^{InterB}$, and N_{CAS}^{InterB} represent the number of interfering requests for the P, CAS, and A command of r, respectively. Finally, Lemma 2 shows that the assumption on relative priorities hold for all platform instances that yield a bounded WCD.

Lemma 2: For a platform instance with $breorder = 0$ or $wb = 1$, the relative priorities of executed requests remains the same over the three commands.

Proof: Recall from Section IV that the MC deploys RR among requests at the head of the bank queues that satisfy their intrabank timing constraints. Hence, the three commands have the same relative priority of the parent request. The only situation in which this priority can change is if the MC prioritizes a command over other commands of same type because it satisfies the interbank timing constraints. This can only happen for CAS commands, since all P commands have the same interbank timing constraints, and the same for A. Due to the bus switching delay, the MC can prioritize an R command over a W command if the previously executed command was an R, or vice versa. This is because R-to-W (W-to-R) delay is larger than R-to-R (W-to-W) delay. If $breorder = 0$, this reordering is prohibited by construction. Under $wb = 1$, only read requests are considered for the interbank interference, while the interference from write requests is accounted for by the write batching interference. Thus, by construction, all CAS commands with regard to (7) have the same interbank timing constraints; thus, no reordering is again possible, and the relative priorities of executed requests remain the same. ■

It remains to compute the individual delay terms.

1) For P: We can simply reuse [10, Lemma 2], which we reproduce below, since the lemma depends only on the behavior of timing constraints and not on the specific MC arbitration.

Lemma 3 ([10, Lemma 2]): The maximum delay caused by N_{PRE}^{InterB} interfering P commands on the one under analysis is upper bounded by

$$L_{PRE}^{InterB}\left(N_{PRE}^{InterB}\right) = 2 \cdot N_{PRE}^{InterB}. \quad (8)$$

The intuition behind the lemma is that there are no interbank constraints on P, but P commands can be delayed due to command bus contention with other commands. However, no more than one A and one CAS command can be issued every four clock cycles, yielding the bound.

2) For A: From Table I, A commands to different banks are limited by two timing constraints: 1) each two A commands to different banks has to be separated by at least *tRRD* cycles and 2) no more than 4 A commands to different banks can be issued every *tFAW* cycles. Fig. 1(c) illustrates both timing constraints for n interfering banks. Furthermore, an A command can be delayed by CAS and P of other banks due to command bus contention: at any clock cycle, the MC can only execute at most one command. Since the number of interfering CAS and P commands is bounded by $2 \cdot N^{InterB}$ as follows:

$$L_{ACT}^{InterB}\left(N_{ACT}^{InterB}\right) = 2 \cdot N^{InterB} + \text{MAX}\left(N_{ACT}^{InterB} \cdot tRRD, \left\lceil \frac{N_{ACT}^{InterB}+1}{4} \cdot tFAW \right\rceil\right). \quad (9)$$

3) For CAS: We use the result in (4) and (5), yielding (10) after adding the effect of command bus contention. Note that we have to consider a sequence of $N_{CAS}^{InterB}+1$ commands, since the first interfering CAS command can also be delayed by previously triggered interbank constraints

$$L_{CAS}^{InterB}\left(N^{InterB}, wb\right) = L^{CAS}\left(N^{InterB}+1, wb\right) + 2 \cdot N^{InterB}. \quad (10)$$

VI. Memory Delay Analysis

In this section, we derive the number of interfering requests $N^{WB}, N^{Conf}, N^{Reorder}$, and N^{InterB} for all 144 COTS platform instances covered by the model in Section IV.

A. General Observations

Before detailing the final equations, we provide some observations that allow us to reduce the number of platform instances that we need to analyze. In the following lemmas, we use the symbol $\times$ (don't care) to denote that a feature does not affect the WCD.

Lemma 4 (Unboundedness of FR-FCFS Without Threshold): For a platform instance with $thr = 0$, if $part = No\text{-}Part$, or $part = Part\text{-}Cr$ and $pr = 0$, then WCD is unbounded.

Proof: part $=$ *No-Part* indicates that PE_{ua} shares banks with other PEs. Since $thr = 0$, in the worst case, those other PEs can always have a ready request to the bank entailing the request under analysis, r_{ua}, to wait indefinitely. For $part = Part\text{-}Cr$, PE_{ua} shares banks with noncritical PEs. In this case, if $pr = 0$, those PEs can also always have ready requests that get scheduled before r_{ua}, which entails its latency to be unbounded. ■

Lemma 5 (Unboundedness of Interbank RR With Reordering): If $breorder = 1$ and $wb = 0$, then WCD is unbounded.

Proof: Under $wb = 0$, the controller does not batch writes together; thus, every access whether it is R or W is handled individually. Let r_{ua} to be a read request that is at the head of its bank queue at time t_0. Moreover, let the request that is picked by the controller at time $t_0 = 0$ according to the

RR policy across banks to be a write request. In addition, all other banks have writes as pending requests. Since the controller is servicing a write request, r_{ua} cannot be inter-ready before $tWL + tB + tWTR$ cycles according to the timing constraints in Table I. However, the write requests from other banks become inter-ready at cycle $tCCD < tWL + tB + tWTR$. Fig. 1(b) depicts these timing constraints for different R/W situations. Since $breorder = 1$, at cycle $tCCD$, the controller will pick one of the inter-ready write requests. Consequently, due to W-to-R turnaround constraint, r_{ua} readiness is pushed to cycle $tCCD + tWL + tB + tWTR$. Similarly, at time $2 \cdot tCCD$, another write request from another bank is inter-ready and further pushes the readiness of r_{ua}. Since this situation can occur indefinitely, r_{ua} suffers from unbounded delay. Same situation occurs if r_{ua} is a write and requests pending at other banks are reads. ■

Lemma 6 (Part-All Effect): If $part = Part\text{-}All$, then $N^{Reorder} = N^{Conf} = 0$, meaning that the request under analysis r_{ua} suffers neither from intrabank reordering nor conflict interference. Furthermore, we have $thr = \times$. Additionally if $wb = 0$, then $pipe = \times$.

Proof: $N^{Reorder}$ and N^{Conf} both count requests of other PEs that target the same bank as r_{ua}. Since $part = Part\text{-}All$ disallows bank sharing among PEs, only requests from PE_{ua} can target the same bank as r_{ua}. Hence, it holds $N^{Reorder} = N^{Conf} = 0$.

Next note that based on the system model in Section IV, thr determines the number of requests that target the same bank as r_{ua} and can be reordered ahead of it; hence, it can only affect the intrabank reordering interference delay. Since such delay is zero, we have $thr = \times$.

Finally, assume that $wb = 0$; then r_{ua} does not suffer from write batching delay. Since furthermore $N^{Reorder} = N^{Conf} = 0$, the only nonzero delay component is the interbank reordering interference. However, $pipe$ cannot affect the number N^{InterB} if interbank interfering requests, since the RR arbitration restricts N^{InterB} based on the number of banks, rather than the number of interfering requests generated based on the pipeline architecture of PEs. Hence, $pipe = \times$. ■

Lemma 7 (Part-Cr Effect): If $part = Part\text{-}Cr$ and $wb = 0$, then r_{ua} suffers neither from intrabank reordering nor conflict interference from other critical PEs, and the worst-case interference is agnostic to the pipeline architecture of critical PEs ($pipe$ of $IO\text{-}Cr$ or $OOO\text{-}All$ has the same effect on WCD).

Proof: Since $part = Part\text{-}Cr$ disallows bank sharing among critical PEs, requests from other critical PEs cannot target the same bank as r_{ua}. Consequently, r_{ua} does not encounter intrabank reordering nor conflict interference from other critical PEs. Recall from the proof of Lemma 6 that $pipe$ affects conflicting and write batching interferences. As a result, if $wb = 0$, having a system with OOO or in-order critical PEs has the same effect on WCD. ■

Lemma 8 (Priority Effect): If $pr = 1$ and $wb = 0$, WCD of r_{ua} is agnostic to the pipeline architecture of noncritical PEs.

Proof: Since $pr = 1$, r_{ua} has higher priority than noncritical requests. Accordingly, regardless of the pipeline architecture of noncritical PEs, r_{ua} incurs an interference from a maximum of one noncritical request, which issued its command just before the arrival of any critical request. The pipeline architecture of the PEs can affect the number of write requests arriving after r_{ua} and hence the write batching delay, but since $wb = 0$, this is not relevant to the platform instance. Thus, WCD is agnostic to the pipeline of noncritical PEs. ■

TABLE III
CONFIGURATIONS.CONFIGURATIONS WITH $wb_j = 0$ AND $breorder = 1$ ARE UNBOUNDED, WHILE FOR CONFIGURATIONS WITH $wb_j = 1$, $breorder = \times$

OS setup	HW setup							
$part_j$	thr_j	pr_j	$pipe_j$ ($wb_j = 0, breorder_j = 0$)			$pipe_j$ ($wb_j = 1, breorder_j = \times$)		
			$OOO\text{-}All$	$IO\text{-}Cr$	$IO\text{-}All$	$OOO\text{-}All$	$IO\text{-}Cr$	$IO\text{-}All$
$Part\text{-}All$	0	0	$confg_1$			$confg_{11}$	$confg_{12}$	$confg_{13}$
	0	1	$confg_2$			$confg_{14}$	$confg_{15}$	$confg_{16}$
	1	0	$confg_1$			$confg_{11}$	$confg_{12}$	$confg_{13}$
	1	1	$confg_2$			$confg_{14}$	$confg_{15}$	$confg_{16}$
$No\text{-}Part$	0	0	$Unbounded$					
	0	1						
	1	0	$confg_3$	$confg_4$	$confg_5$	$confg_{17}$	$confg_{18}$	$confg_{19}$
	1	1	$confg_6$	$confg_7$		$confg_{20}$	$confg_{21}$	$confg_{22}$
$Part\text{-}Cr$	0	0	$Unbounded$					
	0	1	$confg_8$			$confg_{23}$	$confg_{24}$	$confg_{25}$
	1	0	$confg_9$		$confg_{10}$	$confg_{26}$	$confg_{27}$	$confg_{28}$
	1	1	$confg_8$			$confg_{23}$	$confg_{24}$	$confg_{25}$

Lemma 9 (Priority With Part-Cr Effect): If the system adopts a $Part\text{-}Cr$ scheme and the MC prioritizes critical requests ($pr = 1$), then $thr = \times$. Furthermore, if the MC does not deploy write batching ($wb = 0$), then $pipe = \times$.

Proof: Since $part = Part\text{-}Cr$, PE_{ua} shares banks only with noncritical PEs. In addition, since $pr = 1$ and using the same reasoning in the proof of Lemma 8, in the worst case PE_{ua} suffers interference from one noncritical request regardless of the FR-FCFS threshold ($thr = \times$). From Lemma 7, if $part = Part\text{-}Cr$ and $wb = 0$, WCD is agnostic to the pipeline architecture of critical PEs (1). From Lemma 8, if $pr = 1$ and $wb = 0$, WCD is agnostic to the pipeline of noncritical PEs (2). From 1 and 2, $pipe = \times$. ■

Lemma 10 (Write Batching Effect): If the MC implements write batching ($wb = 1$), interbank reordering has no effect on WCD ($breorder = \times$).

Proof: From Section V, $breorder$ only affects the interbank interference component of the WCD. Furthermore from Lemma 2, if $wb = 1$, the value of $L^{InterB}(N^{InterB}, wb)$ in (7) holds independently of the value of $breorder$. Hence, $breorder_j = \times$. ■

Based on the observations established by Lemmas 4–9, Table III maps the 144 platform instances into 25 different *configurations*, where each configuration has a unique WCD bound and represents one or multiple instances. For ease of notation, we represent each configuration as $confg_j = \langle wb_j, thr_j, pr_j, breorder_j, pipe_j, part_j \rangle$, where again we use the symbol $\times$ to denote that a feature can take any value for that configuration since it does not affect the WCD. For example, $confg_1 = \langle wb_1 = 0, thr_1 = \times, pr_1 = 0, breorder_1 = 0, pipe_1 = \times, part_1 = Part\text{-}All \rangle$ denotes the set of 6 instances with no write batching, no threshold mechanism, no interbank reordering, and partitioning for all PEs. Instances which are proved to have unbounded WCD are shown as *unbounded* in Table III.

In the next section, we analyze configurations without write batching ($wb_j = 0$) (configurations 1–10), while Section VI-C covers configurations with write batching (configurations 11–25).

TABLE IV
N^{Conf}, $N^{Reorder}$, AND N^{InterB} FOR CONFIGURATIONS WITHOUT WRITE BATCHING

Configuration	N^{Conf}	$N^{Reorder}$	N^{InterB}
$confg_1$	0	0	$N_B - 1$
$confg_2$	0	0	N_{Bcr}
$confg_3$	$(P-1)\cdot PR$	N_{thr}	$N_B - 1$
$confg_4$	$P_{ncr}\cdot PR + P_{cr} - 1$	N_{thr}	$N_B - 1$
$confg_5$	$P-1$	N_{thr}	$N_B - 1$
$confg_6$	$(P_{cr}-1)\cdot PR + 1$	N_{thr}	$N_B - 1$
$confg_7$	P_{cr}	N_{thr}	$N_B - 1$
$confg_8$	1	0	$N_B - 1$
$confg_9$	$P_{ncr}\cdot PR$	N_{thr}	$N_B - 1$
$confg_{10}$	P_{ncr}	N_{thr}	$N_B - 1$

B. Configurations Without Write Batching

We first illustrate how to derive the number of interfering requests for configuration 3 in Lemma 11, followed by all other configurations in Lemma 12.

Lemma 11: Under $confg_3 = \langle wb_3 = 0, thr_3 = 1, pr_3 = 0, breorder_3 = 0, pipe_3 = \textit{OOO-All}, part_3 = \textit{No-Part}\rangle$, $N^{Conf} = (P-1)\cdot \mathrm{PR}$, $N^{Reorder} = N_{\mathrm{thr}}$, and $N^{InterB} = N_B - 1$ such that based on Lemma 1, r_{ua} encounters a WCD bounded by

$$\begin{aligned}&L^{Conf}((P-1)\cdot \mathrm{PR}) + L^{reorder}(N_{\mathrm{thr}}, 0)\\ &\quad + ((P-1)\cdot \mathrm{PR} + 1)\cdot L^{InterB}(N_B - 1, 0)\\ &\quad + N_{\mathrm{thr}}\cdot L^{InterB}_{CAS}(N_B - 1, 0).\end{aligned}$$

Proof:

1) Since $pipe_3 = \textit{OOO-All}$, each PE can have a maximum of PR pending requests. Also since $pr_3 = 0$, all PEs have the same priority. Hence, in the worst case r_{ua} can suffer conflict interference from PR nonready requests arriving before r_{ua} for each other PE. Thus, $N^{Conf} = ((P-1)\cdot \mathrm{PR})$.
2) Since the MC deploys threshold-based FR-FCFS arbitration, r_{ua} can suffer interference from a maximum of N_{thr} ready requests arriving after r_{ua} to the same bank; hence, $N^{Reorder} = N_{\mathrm{thr}}$.
3) r_{ua} encounters interbank interference from all other banks, therefore, $N^{InterB} = N_B - 1$. Substituting in (1) yields the WCD bound. ■

Lemma 12: Under $confg_j$, where $j \in [1, 10]$, N^{Conf}, $N^{Reorder}$, and N^{InterB} have the values tabulated in Table IV.

Proof: Values for $confg_3$ are already proved in Lemma 11.

1) Under both $confg_1$ and $confg_2$, $part_1 = part_2 = \textit{Part-All}$. Hence, according to Lemma 6, r_{ua} suffers from neither conflict nor intrabank reordering interference; therefore, $N^{conf} = N^{Reorder} = 0$. Since under $confg_1$, $pr_1 = 0$, r_{ua} can suffer interbank interference from all other banks ($N^{InterB} = N_B - 1$). In contrast, $confg_2$ has $pr_2 = 1$; thus, r_{ua} suffers interference from all other critical banks in addition to a maximum of one noncritical bank. Therefore, $N^{InterB} = N_{\mathrm{Bcr}}$.
2) For configurations $confg_4$–$confg_{10}$, r_{ua} can suffer interference from all other banks; thus, $N^{InterB} = N_B - 1$.
3) For configurations $confg_4$–$confg_7$, $confg_9$, and $confg_{10}$, r_{ua} suffers intrabank reordering interference due to the FR-FCFS policy. Since $thr = 1$ for these configurations, a maximum of N_{thr} requests can get reordered before r_{ua}. Accordingly, $N^{Reorder} = N_{\mathrm{thr}}$ for these configurations. On the other hand, since $confg_8$ has $part_8 = \textit{Part-Cr}$ and $pr_8 = 1$, FR-FCFS has no effect on WCD (Lemma 9), and $N^{Reorder} = 0$.
4) For $confg_4$, $pipe_4 = \textit{IO-Cr}$. Accordingly, each critical PE has a maximum of one pending request, while each noncritical PEs can have up to PR pending requests. Therefore, in the worst case r_{ua} suffers conflict interference from $P_{\mathrm{cr}} - 1$ requests from critical PEs and $P_{\mathrm{ncr}}\cdot \mathrm{PR}$ from noncritical ones; hence, $N^{conf} = P_{\mathrm{ncr}}\cdot \mathrm{PR} + P_{\mathrm{cr}} - 1$. Contrarily, $confg_5$ has $pipe_5 = \textit{IO-All}$. Consequently, in the worst case r_{ua} suffers conflict interference from $P-1$ requests ($N^{Conf} = P-1$).
5) For $confg_6$, all PEs are OOO but $pr_6 = 1$. r_{ua} suffers conflict interference from a maximum of one request from noncritical PEs and PR requests from each other critical PE. As a result, $N^{Conf} = (P_{\mathrm{cr}} - 1)\cdot \mathrm{PR} + 1$. The difference between $confg_7$ and $confg_6$ is in the pipeline architecture of critical PEs. Under $confg_7$, critical PEs are in-order with a maximum of one pending request at any time. This reduces N^{Conf} for $confg_7$ to be $N^{Conf} = (P_{\mathrm{cr}} - 1) + 1 = P_{\mathrm{cr}}$.
6) Under configurations $confg_8$–$confg_{10}$, PE_{ua} share banks only with noncritical PEs. Therefore, r_{ua} does not suffer a conflict interference from critical PEs. For $confg_9$, $pr_9 = 0$ and noncritical PEs are OOO. As a result, under $confg_9$, $N^{Conf} = P_{\mathrm{ncr}}\cdot \mathrm{PR}$. Although $confg_{10}$ also has $pr_{10} = 0$, it has $pipe_{10} = \textit{IO-All}$. Hence, each noncritical PE can have a maximum of one pending request at any time. As a result, under $confg_{10}$, $N^{Conf} = P_{\mathrm{ncr}}$. Since $pr_8 = 1$, under $confg_8$, r_{ua} can suffer interference from a maximum of one request from all noncritical PEs and $N^{Conf} = 1$. ■

C. Configurations With Write Batching

We now discuss the WCD for write batching configurations (configurations $confg_{11}$–$config_{28}$). Based on (1) and using a similar procedure as in Lemma 12, we obtain the values of N^{Conf}, $N^{Reorder}$, and N^{InterB} using Table IV, based on the equivalent configuration with $wb = 0$.

We next calculate the maximum number N^{WB} of interfering writes as $N^{WB} = W_{\mathrm{btch}} + N_R^{\mathrm{before}} + N_R^{\mathrm{after}}$. Recall that in the worst case, every time a read request arrives at the MC, one write request can be enqueued in the write buffer. The term W_{btch} accounts for the interference caused by requests arrived before r_{ua}: in the worst case, r_{ua} arrives while the MC has just started to serve a write batch, which consists of W_{btch} requests. The remaining terms N_R^{before} and N_R^{after} accounts for read requests that arrive after the arrival of r_{ua} but before r_{ua} completes; each such read request can add one extra write request to the buffer. N_R^{before} is the number of read requests that arrive after r_{ua} but are executed before it because of the intrabank FR-FCFS and the interbank RR policies. N_R^{before} depends on the configuration, and is determined by Table V. For configurations 11–13 and 23–25, r_{ua} does not suffer any intrabank reordering interference. However, r_{ua} can wait for

TABLE V
N_R^{before} FOR DIFFERENT WRITE BATCHING CONFIGURATIONS

Configuration	N_R^{before}	Configuration	N_R^{before}
$config_{11}$, $config_{12}$, $config_{13}$	$N_B - 1$	$config_{20}$	$N_{thr} \cdot N_B$
$config_{14}$, $config_{15}$, $config_{16}$	N_{Bcr}	$config_{21}$, $config_{22}$	$N_{thr} \cdot N_B$
$config_{17}$	$N_{thr} \cdot N_B$	$config_{23}$, $config_{24}$, $config_{25}$	$N_B - 1$
$config_{18}$	$N_{thr} \cdot N_B$	$config_{26}$, $config_{27}$	$N_{thr} \cdot N_B$
$config_{19}$	$N_{thr} \cdot N_B$	$config_{28}$	$N_{thr} \cdot N_B$

$N_B - 1$ requests from other banks due to the RR policy. Since configurations14–16 support prioritization and have a Part-All scheme, r_{ua} only suffers from $N_{\text{Bcr}} - 1$ requests accessing other critical banks, in addition to a maximum of one noncritical request; thus, $N_R^{\text{before}} = N_{\text{Bcr}}$. For configurations 17–22 and 26–28, a maximum N_{thr} requests can arrive after r_{ua} to same bank and get serviced before it because of FR-FCFS effect, while each of these N_{thr} requests can also wait for N_B requests from other banks because of RR policy. Thus, $R^{\text{before}} = N_{\text{thr}} \cdot N_B$ for these configurations in Table V. On the other hand, N_R^{after} is the number of read requests that arrive after r_{ua}'s arrival but before its completion time, and are executed after r_{ua}. N_R^{after} depends on the number of concurrent requests that each PE can generate as

$$N_R^{\text{after}} = \begin{cases} P \cdot \text{PR}, & \text{if } pipe_j = OOO \\ P_{\text{cr}} + P_{\text{ncr}} \cdot \text{PR}, & \text{if } pipe_j = IO\text{-}Cr \\ P, & \text{if } pipe_j = IO\text{-}All. \end{cases} \tag{11}$$

VII. EVALUATION

We use MacSim [31], a multiprocessor architectural simulator integrated with DRAMSim2 [28]. MacSim models x86 instruction-set architecture and supports in-order and out-of-order PEs. Furthermore, it allows the configuration of the maximum number of pending requests through managing the number of entires in MSHRs. MacSim has a frontend that includes the virtual-to-physical mapping. This enables us to implement partitioning without running an actual OS. We extend this frontend to support the three partitioning schemes discussed in Section IV. DRAMSim2 [28] is a detailed simulator for the DRAM subsystem. It has per-bank queues with FR-FCFS arbitration among requests to same bank. Moreover, it provides a parameter to configure the threshold of FR-FCFS. Finally, it deploys RR across banks. We extend DRAMSim2 to support priority assignment amongst different PEs as well as write batching.

We use benchmarks from the EEMBC-auto suite [32], which include representative applications from the embedded automotive domain. We use the *a2time* and *rspeed* benchmarks as the critical tasks running on critical PEs. Both benchmarks exhibit high instruction dependencies; hence, they are latency sensitive. In addition, we use the *matrix* and *aifftr* benchmarks as the noncritical applications as they are the two most memory-intensive applications in the EEMBC suite. In addition, to stress the worst-case interference delay, we use two synthetic benchmarks: 1) *latency* and 2) *BW* [17]. *Latency* represents a latency-sensitive application, while *BW* represents a BW sensitive application. We use *latency* as a critical application and *BW* as the noncritical applications. Table VI summarizes the setup of the evaluation. Sections VII-A and VII-B

TABLE VI
EVALUATION PARAMETERS CONFIGURATION

PEs	A private 16KB L1 and a shared 1MB L2 cache An in-order PE has a maximum of one pending request to the DRAM An OOO PE has a maximum of 4 pending requests to the DRAM ($PR = 4$)	
	Sections VII-A, VII-B, and VII-D	2 critical PEs and 2 non-critical PEs ($P_{cr} = P_{ncr} = 2$ and $P = 4$)
	Section VII-C	2 non-critical PEs ($P_{ncr} = 2$), while we sweep number of critical PEs
		2 critical PEs ($P_{cr} = 2$), while we sweep number of critical PEs
Bank Partitioning	Through the virtual-to-physical address mapping component at MacSim's frontend Based on the configuration, we enable the corresponding partitioning ($Part\text{-}All$, $Part\text{-}Cr$, or $No\text{-}Part$)	
DRAM	DDR3-1333H [3] with single channel, single rank, and 8 banks	
MC	Based on the configuration, critical PEs can be assigned higher priority than non-critical PEs Per-bank queues with FR-FCFS arbitration Based on the configuration, we enalbe or disable the threshold for FR-FCFS	
	Sections VII-A, VII-B, and VII-C	$N_{thr} = 8$ and write batching is disabled
	Section VII-D	We sweep N_{thr} and write batching is disabled
	Section VII-E	Write batching is enabled
Benchmarks	EEMBC [32]	The two critical PEs execute *a2time* and *rspeed* The two non-critical PEs execute *matrix* and *aifftr*
	Synthetic [17]	Each of the critical PEs execute one instance of the *latency* benchmark Each of the non-critical PEs execute one instance of the *Bandwidth* benchmark

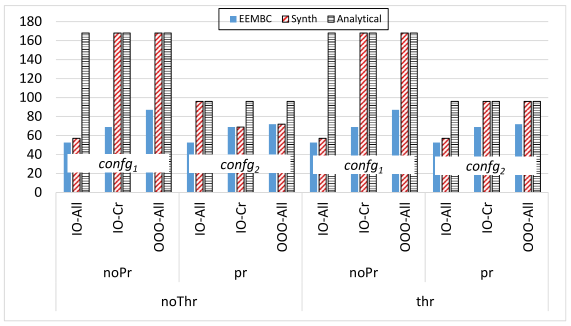
discuss the WCD and BW results of configurations without write batching, while Section VII-E discusses write batching findings. Sections VII-C and VII-D study the effects of number of PEs in the system and the FR-FCFS threshold value on WCD, respectively.

A. Worst-Case Interference Delay of Different MPSoC Configurations

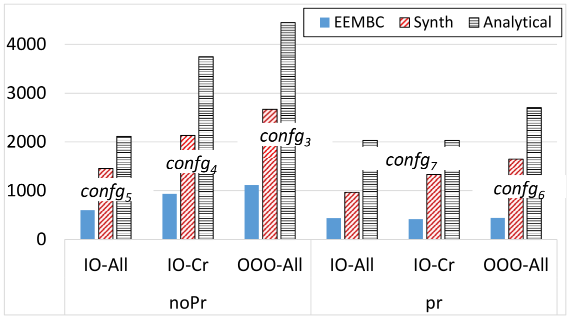
Fig. 3 delineates the WCD for COTS MPSoCs with different set of features. Fig. 3 shows both the analytical latency bounds derived in Section VI, and the observed experimental WCD for both EEMBC and synthetic benchmarks. In *Part-All* [Fig. 3(a)], we depict WCD for configurations that support FR-FCFS threshold as well as configurations that do not support it. While for *no-Part* [Fig. 3(b)] and *Part-Cr* with no priority [Fig. 3(c, i)], we only show configurations with FR-FCFS threshold support since those that do not support the threshold mechanism have unbounded WCD.

Observations:

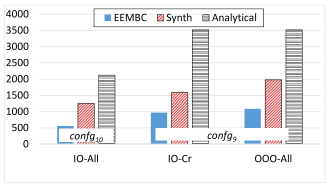
1) Fig. 3(a) shows that both *thr* ($thr = 1$) and noThr ($thr = 0$) exhibit almost identical WCD. This confirms the conclusion of Lemma 6 that in MPSoCs with partitioned banks among all PEs (*Part-All*), the existence of FR-FCFS threshold mechanism has no effect on WCD.
2) Lemma 6 concludes that WCD in (*Part-All*) configurations is independent of the pipeline architecture of PEs. For the same priority setting, Fig. 3(a) illustrates that *IO-Cr* and *OOO-All* have very similar experimental WCD. This is particularly true for the synthetic workloads, where this experimental WCD is also close to the analytical bound since the workloads are more memory stressful. However, *IO-All* has considerably less WCD [compare for example *OOO-All* and *IO-Cr* results for noPr in Fig. 3(a)]. This is because the worst-case scenario for this case assumes that *IO-All* issues a sufficient number of requests in a very short time to saturate the FR-FCFS threshold. This scenario is hard to achieve in practice. This also justifies the big gap between the experimental WCD and the analytical bound for the *IO-All* configurations.
3) Fig. 3(b) shows that if critical PEs have higher priority [pr in Fig. 3(b)], configurations with either *IO-Cr* or *IO-All* exhibit similar WCD. For synthetic workloads, *IO-All* has slightly less WCD than *IO-Cr* for the same



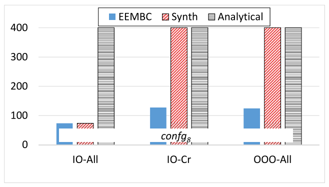
(a)



(b)



(i)



(ii)

(c)

Fig. 3. Worst-case interference delay in *ns* for different system configurations. (a) Systems with *Part-All* partitioning scheme. (b) Systems with *No-Part* partitioning scheme. (i) $thr = 1$ and $pr = 0$. (ii) $thr = \times$ and $pr = 1$. (c) Systems with *Part-Cr* partitioning scheme.

reason as in Observation 2. This confirms the conclusion of Lemma 6 that in MPSoCs with prioritization, the pipeline architecture of the noncritical PEs has no effect on WCD.

4) MPSoCs that partition banks among PEs and assigns critical PEs a higher priority achieve the smallest observed and analytical WCD [pr in Fig. 3(a)]. This corresponds to MPSoCs with configuration 2. In contrast, the maximum observed WCD occurs in MPSoCs that share banks among PEs and assign same priority for all PEs [noPr in Fig. 3(b)]. This corresponds to MPSoCs with configuration 3 (Lemma 11).

5) Mapping critical PEs to different banks (*Part-Cr*) achieves best results for MPSoCs that assigns critical PEs a higher priority [Fig. 3(c, ii)]. This is because *Part-Cr* substantially reduces interference from other critical PEs to only interbank interference leaving noncritical PEs as the only source for intrabank interference. Orthogonally, $pr = 1$ reduces intrabank interference from noncritical cores to only one request at maximum. As a result, the total WCD is remarkably reduced.

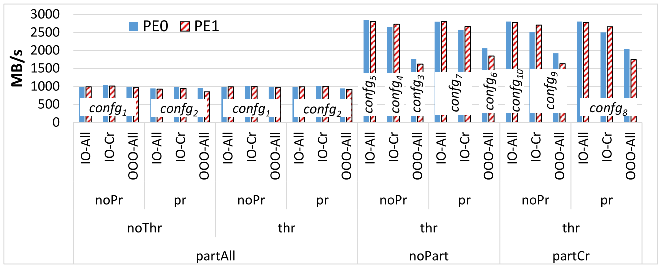
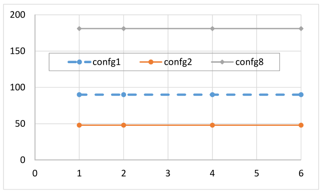
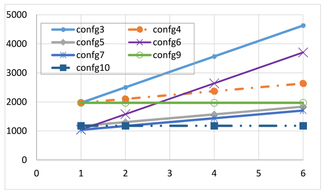


Fig. 4. BW of noncritical PEs.



(a)



(b)

Fig. 5. Sensitivity of WCD to number of critical PEs. Configurations 1, 2, and 8. (b) Configurations 3–7, 9, and 10.

B. Bandwidth of Noncritical PEs

Fig. 4 depicts the BW delivered to the noncritical PEs under different configurations. Although *Part-All* scheme (which is followed by many related works [6]–[8], [10], [12], [20]) is able to substantially reduce WCD of the critical PEs compared to *no-Part* [Fig. 3(a) and (b)], it also significantly reduces the BW delivered to the noncritical PEs (Fig. 4). On the other hand, results show that *Part-Cr* is a viable solution that reduces WCD for critical PEs without compromising the BW of the noncritical PEs. For instance, compare $confg_6$, $confg_2$, and $confg_8$, where all configuration parameters other than partitioning are the same. Compared to $confg_6$ [Fig. 3(b)], $confg_2$ [Fig. 3(a)] reduces WCD of critical PEs by 96%, while $confg_8$ [Fig. 3(c, ii)] reduces WCD by 89%. On the other hand, as Fig. 4 depicts, noncritical PEs encounter a 60% BW degradation under $confg_2$, while this degradation is negligible (only 0.85%) for $confg_8$.

C. Sensitivity to Number of Interfering PEs

We study the scalability of the WCD with the number of coexisting PEs in the MPSoC. Fig. 5 plots the WCD at number of critical PEs of $P_{cr} = 1$ (this is the one under analysis, which means no interfering critical PEs), 2, 4, and 6. The number of noncritical PEs in Fig. 5 is 2. On the other hand, Fig. 6 plots the WCD at number of noncritical PEs of $P_{cr} = 1$, 2, 4, and 6. The number of critical PEs in this case is 2.

Observations:

1) Fig. 5(a) illustrates that WCD of a critical request in configurations 1, 2, and 8 is agnostic to the number of interfering critical and noncritical PEs. This is because configurations 1 and 2 deploy a *Part-All* scheme, while configuration 8 deploys *Part-Cr* with fixed priority.

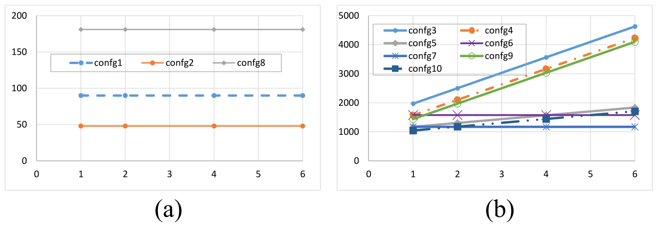


Fig. 6. Sensitivity of WCD to number of noncritical PEs. (a) Configurations 1, 2, and 8. (b) Configurations 3–7, 9, and 10.

These configurations are crucial to consider, especially if the number of PEs in the MPSoC is larger or the task requires a fixed known WCD independent of other PEs.

2) Fig. 5(b) shows that WCD in configurations 9 and 10 is independent of the number of co-existing critical PEs in the MPSoC. This is because both configurations deploy a *Part-Cr* scheme.
3) Similarly, Fig. 6(b) shows that WCD in configurations 6 and 7 is independent of the number of co-existing noncritical PEs in the MPSoC. This is because both configurations prioritize critical PEs over noncritical ones ($pr_6 = pr_7 = 1$).
4) WCD is more sensitive to the number of co-existing critical PEs in MPSoCs with configurations 3 and 6 as compared to configurations 4, 5, and 7. This can be observed by looking at the slope (rate of change) of different plots in Fig. 5(b). The intuition is that MPSoCs with configurations 3 and 7 have critical PEs with OOO pipeline, while in configurations 4, 5, and 7, critical PEs are in-order. In fact, the slope of configurations 3 and 6 is 4× that of configurations 4, 5, and 7. The reason is that PR = 4 for OOO PEs, while effectively PR = 1 for in-order PEs.
5) Similarly, in Fig. 6(b), the slope of plots of configurations 3, 4, and 9 is 4× that of configurations 5 and 10. That is, because configurations 3, 4, and 9 have OOO noncritical PEs, while configurations 5 and 10 have in-order noncritical PEs.
6) A final observation is that partitioning schemes have scalability limitations with the increase of P. *Part-All* requires that $N_B \geq P$ to be able to partition banks across all PEs, while *Part-Cr* requires $N_B \geq P_{\text{cr}}$. Therefore, we do not simulate beyond a maximum of $P = 8$ since the number of banks in the system is 8. Such limitation does not exist in configurations with *no-Part*.

D. Sensitivity to the FR-FCFS Threshold

We study the effect of the maximum number of ready requests that can get reordered before one critical request (N_{thr}). Fig. 7 plots WCD for different configurations at $N_{\text{thr}} \in \{1, 2, 4, 8, 16, 32, 64\}$.

Observations:

1) Configurations 1 and 2 [Fig. 7(a)] have a WCD that is independent of N_{thr}. This is because they eliminate reordering interference from all PEs by mapping them to different banks (*Part-All*).
2) Configuration 8 [Fig. 7(a)] also has a WCD that is independent of N_{thr}. Configuration 8 eliminates reordering interference from critical PEs by mapping them to different banks (*Part-Cr*). In addition, it eliminates reordering interference from noncritical PEs by assigning them lower priority than critical PEs.
3) The rate of change (slope) is the same across configurations 3–7, 9, and 10 in Fig. 7(b). This is because the reordering component (as discussed in Section V) depends only on the value of N_{thr}, and the JEDEC timing parameters.

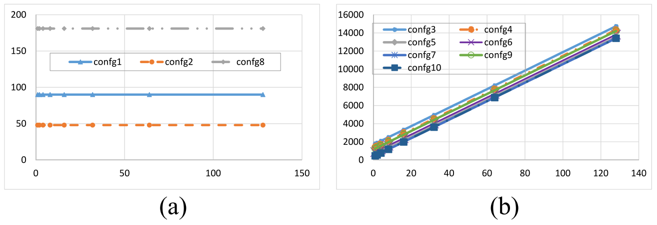


Fig. 7. Effect of different N_{thr} values on WCD. (a) Configurations 1, 2, and 8. (b) Configurations 3–7, 9, and 10.

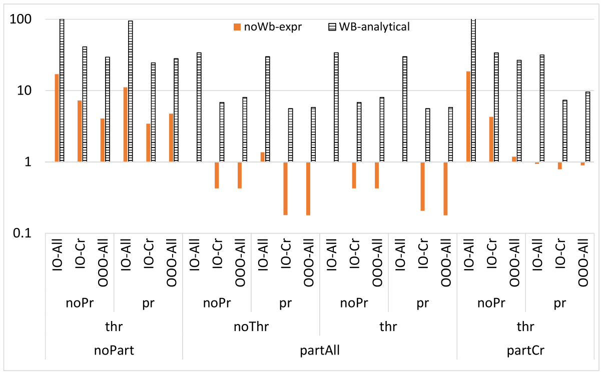


Fig. 8. WCD of write batching versus non write batching.

E. Effect of Write Batching

To study the effect of write batching on WCD, we also execute the same experiment setup in Section VII-A using the BW and latency synthetic benchmarks. Fig. 8 compares obtained experimental WCD results from configurations with no write batching (shown as noWb-expr) with those obtained from configuration with write batching. In addition, Fig. 8 depicts the analytical WCD bounds for write batching configurations obtained from the static analysis conducted in Section VI (shown as WB-analytical). For visibility purposes, all results are normalized to the experimental WCD results with write batching; the y-axis is in log scale.

Observations:

1) Write batching significantly increases the analytical WCD. This is because the per-request analysis (as Section VI-C discusses) has to assume the worst case situation, where a read suffers from the maximum number of writes and all these writes are conflicts. However,

this is a highly pessimistic situation that may never be observed experimentally. Accordingly, there is a big gap between analytical and observed WCD for write batching configurations. Fig. 8 shows that the analytical WCD can be as large as 100× compared to the observed WCD.

2) On the other hand, overall, write batching reduces experimental latencies. Across all configurations, we found that WCD of noWb-expr is 2.84× on average as compared to Wb-expr. As Fig. 8 illustrates, this number can reach up to 10× for some configurations. Consequently, the overall conclusion is that while write batching dramatically increases the pessimism of analytical WCD, it can actually have a positive effect on measured delay. This conclusion aligns with earlier observations in [10], which notes that the pathological arrival patterns induced by write batching prevent the determination of tight delay bounds.

VIII. Conclusion

In this paper, we derived a generalized analysis that bounds the per-request delay that a PE suffers in DRAM due to interference by other PEs. The analysis accounts for a breadth of features found in modern MPSoC platforms. In total, we explore 144 possible COTS platform instances. We then investigate the effect of these features on an MCS, based on both the derived analytical memory delay bound for critical applications, and the average BW for noncritical applications. Our exploration leads to several key observations, of which we highlight the following four.

1) Certain feature configurations result in unbounded delay bounds, and thus should be avoided in MCS.
2) Compared to previous work, leveraging existing features such as PE prioritization can allow the designer to better tradeoff the maximum delay for critical applications and the BW for noncritical ones.
3) There is interdependency among the effects of the features on both the delay and the BW. Existence of some features can countermand the effect of other features. For instance, if the MC supports request prioritization and the OS assigns different banks to critical PEs, the pipeline architecture of all PEs have no effect on delay bounds.
4) While the write batching mechanism works well in the average case, it unfortunately induces pathological cases that result in high bounds on per-request delay. A different read-write reordering mechanism could be designed to improve worst-case bounds [33]. In alternative, one approach to address write batching is to incorporate its effect on the job-driven analysis [9] rather than the request-driven analysis. Job-driven analysis can reduce pessimism by incorporating information on the maximum number of requests produced by other PEs during the execution of the task under analysis; but it comes at the cost of composability, e.g., the delay bound can be affected by changes to other applications running on different PEs.

References

[1] M. Hassan, "Heterogeneous MPSoCs for mixed-criticality systems: Challenges and opportunities," *IEEE Design Test*, vol. 35, no. 4, pp. 47–55, Aug. 2018.

[2] V. Boppana *et al.*, "UltraScale+ MPSoC and FPGA families," in *Proc. IEEE Hot Chips Symp. (HCS)*, 2015, pp. 1–37.

[3] *DDR3 SDRAM JEDEC*, JEDEC Standard jesd79-3b, 2008.

[4] D. Guo, M. Hassan, R. Pellizzoni, and H. Patel, "A comparative study of predictable DRAM controllers," *ACM Trans. Embedded Comput. Syst.*, vol. 17, no. 2, 2018, Art. no. 53.

[5] M. Hassan, H. Patel, and R. Pellizzoni, "A framework for scheduling DRAM memory accesses for multi-core mixed-time critical systems," in *Proc. Real Time Embedded Technol. Appl. Symp. (RTAS)*, Seattle, WA, USA, 2015, pp. 307–316.

[6] L. Ecco, S. Tobuschat, S. Saidi, and R. Ernst, "A mixed critical memory controller using bank privatization and fixed priority scheduling," in *Proc. Embedded Real Time Comput. Syst. Appl. (RTCSA)*, 2014, pp. 1–10.

[7] J. Jalle *et al.*, "A dual-criticality memory controller (DCmc): Proposal and evaluation of a space case study," in *Proc. IEEE Real Time Syst. Symp. (RTSS)*, 2014, pp. 207–217.

[8] D. Guo and R. Pellizzoni, "A requests bundling DRAM controller for mixed-criticality systems," in *Proc. IEEE Real Time Embedded Technol. Appl. Symp. (RTAS)*, 2017, pp. 247–258.

[9] H. Kim *et al.*, "Bounding memory interference delay in COTS-based multi-core systems," in *Proc. IEEE Real Time Embedded Technol. Appl. Symp. (RTAS)*, 2014, pp. 145–154.

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

[11] B. Bhat and F. Mueller, "Making dram refresh predictable," in *Proc. Euromicro Conf. Real Time Syst.*, 2010, pp. 145–154.

[12] Z. P. Wu, R. Pellizzoni, and D. Guo, "A composable worst case latency analysis for multi-rank DRAM devices under open row policy," *Real Time Syst.*, vol. 52, no. 6, pp. 761–807, 2016.

[13] M. Hassan, H. Patel, and R. Pellizzoni, "PMC: A requirement-aware dram controller for multicore mixed criticality systems," *ACM Trans. Embedded Comput. Syst.*, vol. 16, no. 4, 2017, Art. no. 100.

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

[15] N. Kim *et al.*, "Attacking the one-out-of-m multicore problem by combining hardware management with mixed-criticality provisioning," in *Proc. Real Time Syst.*, 2017, pp. 1–12.

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

[17] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha, "Memory bandwidth management for efficient performance isolation in multicore platforms," *IEEE Trans. Comput.*, vol. 65, no. 2, pp. 562–576, Feb. 2016.

[18] S. Schliecker, M. Negrean, and R. Ernst, "Bounding the shared resource load for the performance analysis of multiprocessor systems," in *Proc. Conf. Design Autom. Test Europe*, 2010, pp. 759–764.

[19] K. Lampka, G. Giannopoulou, R. Pellizzoni, Z. W. Pei, and N. Stoimenov, "A formal approach to the worst-case response time analysis of multicore systems with memory contention," *Real Time Syst.*, vol. 50, nos. 5–6, pp. 736–773, 2014.

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

[21] R. Trüb, G. Giannopoulou, A. Tretter, and L. Thiele, "Implementation of partitioned mixed-criticality scheduling on a multi-core platform," *ACM Trans. Embedded Comput. Syst.*, vol. 16, no. 5, 2017, Art. no. 122.

[22] B. Rouxel, S. Derrien, and I. Puaut, "Tightening contention delays while scheduling parallel applications on multi-core architectures," *ACM Trans. Embedded Comput. Syst.*, vol. 16, no. 5, 2017, Art. no. 164.

[23] *Qualcomm Snapdragon 600E Processor APQ8064E Recommended Memory Controller and Device Settings Application Note*, Qualcomm, San Diego, CA, USA, 2016.

[24] *External Memory Interface Handbook Volume 2: Design Guidelines*, Intel, Santa Clara, CA, USA, 2017.

[25] M. Hassan and H. Patel, "MCXplore: Automating the validation process of DRAM memory controller designs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 5, pp. 1050–1063, May 2018.

[26] M. Hassan and H. Patel, "MCXplore: An automated framework for validating memory controller designs," in *Proc. IEEE Conf. Design Autom. Test Europe (DATE)*, 2016, pp. 1357–1362.

[27] B. Jacob, S. Ng, and D. Wang, *Memory Systems: Cache, DRAM, Disk*. Amsterdam, The Netherlands: Morgan Kaufmann, 2010.

[28] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "DRAMSim2: A cycle accurate memory system simulator," *IEEE Comput. Archit. Lett.*, vol. 10, no. 1, pp. 16–19, Jan./Jun. 2011.

[29] M. Calle and R. Ramaswami, "Multi-bank scheduling to improve performance on tree accesses in a dram based random access memory subsystem," U.S. Patent 6 839 797, Jan. 4, 2005.

[30] M. Hassan, A. M. Kaushik, and H. Patel, "Reverse-engineering embedded memory controllers through latency-based analysis," in *Proc. IEEE Real Time Embedded Technol. Appl. Symp. (RTAS)*, Seattle, WA, USA, 2015, pp. 297–306.

[31] H. Kim, J. Lee, N. Lakshminarayana, J. Lim, and T. Pho, "Macsim: Simulator for heterogeneous architecture," Georgia Inst. Technol., Atlanta, GA, USA, Rep., 2012.

[32] J. Poovey, *Characterization of the EEMBC Benchmark Suite*, North Carolina State Univ., Raleigh, NC, USA, 2007.

[33] L. Ecco and R. Ernst, "Tackling the bus turnaround overhead in real-time SDRAM controllers," *IEEE Trans. Comput.*, vol. 66, no. 11, pp. 1961–1974, Nov. 2017.

Mohamed Hassan (M'11) received the M.Sc. degree from Cairo University, Giza, Egypt, in 2012 and the Ph.D. degree from the University of Waterloo, Waterloo, ON, Canada, in 2017.

He was a Research and Development SoC Lead Engineer with Intel PSG, Toronto, ON, Canada. He is an Assistant Professor with the School of Engineering, University of Guelph, Guelph, ON, Canada. His current research interests include real-time embedded systems, multicore architectures, and hardware validation and security.

Rodolfo Pellizzoni (M'05) received the master's degree from Scuola Superiore Sant'Anna, Pisa, Italy, in 2005 and the Ph.D. degree from the University of Illinois at Urbana–Champaign, Champaign, IL, USA, in 2010.

He is an Associate Professor with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada. His current research interests include real-time systems and timing analysis, with a particular focus on HW/SW architectures for timing predictability and safety certification.