Managing DRAM Interference in Mixed Criticality Embedded Systems

Mohamed Hassan
Electrical and Computer Engineering Department
McMaster University, Canada
Email: mohamed.hassan@mcmaster.ca

***Abstract*—Modern embedded systems such as those in autonomous vehicles, drones, and robots have a mixed criticality nature. They run both safety-critical tasks with tight timing constraints as well as non-critical tasks with high average performance demands. Traditional solutions deployed in both general-purpose computing as well as low-end embedded systems are no longer suitable. In this work, we study the challenges facing the deployment of commercial-off-the-shelf memory systems for mixed-criticality embedded systems and we provide potential solutions to enable such deployment.**

I. Introduction

The presence of embedded systems is significantly increasing in many emerging domains such as healthcare, autonomous vehicles, manufacturing automation, and Internet-of-Things (IoT). Embedded systems in these domains have unique characteristics and face new challenges that distinguish them from traditional embedded systems. One of the key challenges is the requirement to deploy tasks with various criticalities, comprising what is known as Mixed Criticality Systems (MCS). The criticality of a task is a reflection of the potential consequences upon failure to meet the requirements of this task. Accordingly, industry standards define distinct criticality levels, where each criticality level introduces certain assurance measures against failures to meet requirements of tasks running in this criticality level. Manufacturers have to go through a rigorous certification process to meet those measures. For instance, the DO-178C avionics standard defines five Design Assurance Levels (DAL) for software considerations in airborne systems, while the ISO 26262 automotive standard defines four Automotive Safety Integrity Levels (ASIL) for systems deployed in automobiles.

Such mixed criticality environment is challenging for embedded systems. This is because tasks of different criticalities, and hence different requirements are required to simultaneously run on the same hardware platform and share same resources due to area and cost considerations. Moreover, tasks of different criticality usually have conflicting requirements. Tasks with higher criticality are often latency sensitive with strict worst-case temporal requirements, which if broken, can lead to severe consequences or even life losses. In contrast, tasks with lower criticality are often throughput sensitive which require high average performance and can tolerate infrequent deadline misses [1], [2].

Ideally, a hardware architecture that achieves both low latency and high throughput will be able to accommodate these different types of tasks. Nonetheless, in reality, the architecture has a trade-off between these two metrics: improving one usually comes at the expense of deteriorating the other. Considering Dynamic Random Access Memories (DRAMs), which is the commodity off-chip memory in most computing systems nowadays [3]: on the one hand, achieving high throughput (i.e. memory bandwidth in this case) requires memory controllers to deploy complex optimizations including multiple reordering and arbitration levels, prioritizations, and adaptive policies. On the other hand, these optimizations result in highly pessimistic worst case delay (WCD), which hinders the ability of the architecture to run higher-criticality tasks.

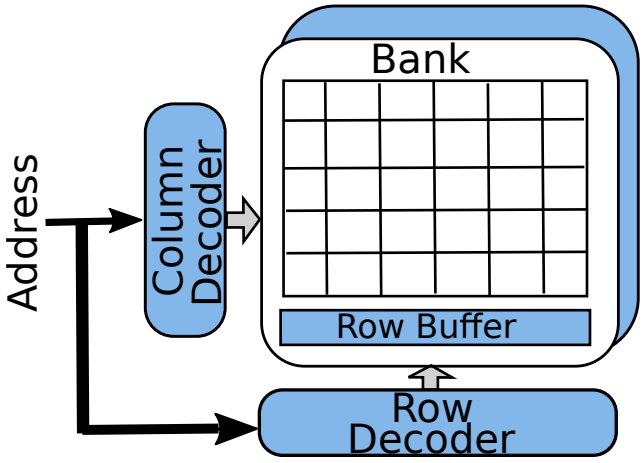


Fig. 1: DRAM internal architecture

In this paper, we focus on DRAMs as a vital component in modern embedded systems. We study existing architectures found in both general-purpose systems both analytically and experimentally, and investigate their limitations upon adopted in MCS.

Based on this study, we highlight potential directions to manage DRAM accesses to achieve both low latency and high average performance in MCS.

II. DRAM Operation

A. DRAM Internal Architecture

The DRAM is organized as banks. As Figure 1 delineates, each bank consists of memory cells structured as an array of

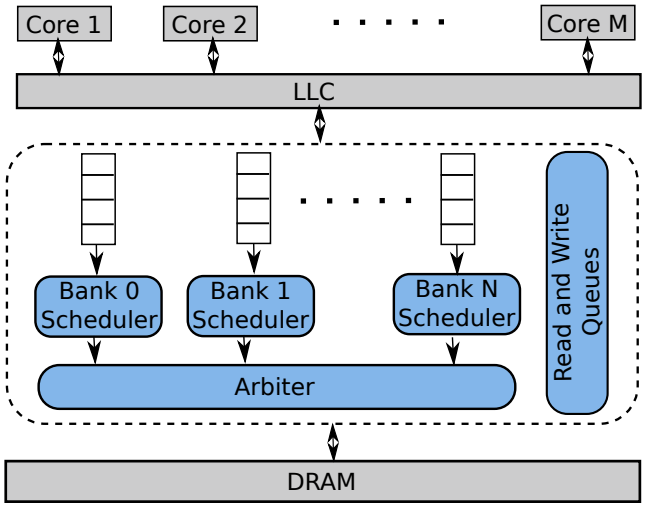


Fig. 2: Memory controller architecture

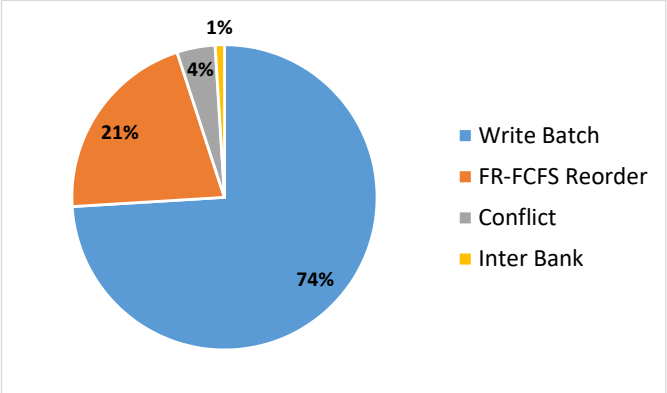


Fig. 3: DRAM internal architecture

columns and rows. Each bank also has sense amplifiers, which in addition to amplifying signals from DRAM cells, work as a small cache holding the most recently accessed row of that bank. This cache is usually referred to as the *row buffer*. The memory address is provided to the DRAM in two stages. First, the row segment of the address along with a row address strobe (RAS) are provided to the DRAM device to fetch the requested row from the cells array to the row buffer. This operation is known as *row activation* (ACT). Then, the column segment along with a column address strobe (CAS) are provided to the DRAM to read/write (RD/WR) certain columns from the row buffer. Exploiting data locality, accesses to data that is already available in the row buffer do not require the first stage. Those requests are referred to as *row hits* and they encounter a lower latency. Contrarily, requests to different data than the one in the row buffer have first to write the data in the row buffer back to the DRAM cells before it can fetch its own data. This operation is known as *row precharging* (PRE). Those requests are referred to as *row conflicts* and they encounter a higher access latency. Requests to the off-chip DRAM are managed through the on-chip memory memory controller, which we explain its details in the next subsection.

B. Memory Controller Architecture

Figure 2 depicts the architecture of a baseline memory controller found in Commercial Off-The-Shelf (COTS) CPU chips. The memory controller translates the request address into the DRAM bank, row, and column segments to access specific DRAM cells. It also translates the request into DRAM commands that execute the DRAM access as needed: ACT, PRE, RD, or WR. All these commands have to adhere to certain timing constraints dictated by the DRAM standard. For details about these timing constraint, we refer the reader to the DRAM standard in [4].

To increase performance, the memory controller performs multiple optimizations including reordering row hits over row conflicts in the same bank, known as First Ready-First Come First Serve (FR-FCFS) arbitration in the bank scheduler shown in Figure 2. Nonetheless, to prevent starvation for conflict requests, COTS controllers usually put a threshold on the maximum number of row hits that can be consecutively serviced from same row before closing it, we call this threshold, N_{thr}. In addition, since read requests are usually more critical for performance than write requests as they can stall the processor pipeline, the memory controller reorders reads over writes deploying what is known as write batching. In write batching, writes are buffered into a separate queue and are not serviced until they form a designated batch size, which we call N_{wb} throughout this paper. This also helps in mitigating the data bus switching from reads to writes and vice versa, which has huge overheads.

III. Analysis of Major DRAM Interference Sources

We study the memory controller architecture in COTS platforms to highlight the main sources of interference and investigate their effect in both the WCD as well as the system BW. Using the analysis in [5], We decompose the total memory delay into its components covered in Section II. Figure 3 depicts the percentage that each of these components contributes to the total WCD. As the figure illustrates, the major cause of large memory delays in COTS controllers is the reordering. This reordering is implemented in two components: the per-bank FR-FCFS arbitration, and the global write batching arbitration. Therefore, this paper focuses on these two components, which are found almost in all modern memory controllers.

A. FR-FCFS Reorder Effect

1) Effect on WCD: In worst case, a request waits for the maximum of (N_{thr}) requests reordered ahead of that request. Since these N_{thr} requests are targeting an open row by definition, they are separated by only $tCCD$ cycles and their total delay is $N_{thr} \cdot tCCD$. However, since each of

these requests also can suffer from inter-bank interference from other $NB-1$ banks. Therefore, the total delay is:

$$N_{thr} \cdot NB \cdot tCCD.$$

B. Effect on BW

We also study the effect of reordering on the system BW. Recall that reordering is originally implemented in COTS to enable high memory bandwidth. Ideally, if there are enough number of open requests to saturate the N_{thr} threshold, the memory pattern will be: open a row, service N_{thr} open requests to that row, and then close that row and open another one and repeat the pattern. Under this ideal situation, the peak bandwidth is calculated as

$$\frac{(N_{thr} \cdot NB + 1) \cdot 64}{N_{thr} \cdot NB \cdot tCCD + tRC}.$$

The intuition behind this calculation is that the first request has to open the row, so it consumes the large tRC delay. Afterwards, all requests to same row are open and consume $tCCD$ cycles. Each of these requests is 64B.

C. Write Batching Reorder Effect

1) Effect on WCD: In worst-case, a read request has to wait for a complete write batch before it can get serviced. In addition, all the N_{WB} write requests of the batch are row conflicts. Therefore, each of these requests according to the DRAM standard has in worst case to take $tRCD + tWL + tB + tWR + tRP$ cycles. As a result, the write batch worst effect on WCD is:

$$N_{WB} \cdot (tRCD + tWL + tB + tWR + tRP).$$

2) Effect of BW: Recall that to calculate the possible peak BW, we have to consider the ideal situation. Ideally, the memory controller has enough number of buffered requests to keep alternating between read and write batches: it service number of reads, afterwards, it services a write batch, and then switch back to the reads. This will minimize the bus turnaround delay, which is the main target of write batching. In addition, in the ideal case and unlike the WCD situation: these requests are open requests. Since most COTS architectures deploy a write-back write-allocate policy, writes to DRAM only occurs because of cache eviction of a modified cache line. Hence, the number of writes at the DRAM controllers are at maximum equal to the number of reads. Accordingly, each write batch, a number of read requests equal to the write batch is serviced before switching back to writes. From this discussion, we dervie the peak BW of write batching as follows:

$$\frac{(N_{WB} \cdot 2 \cdot 64)}{N_{WB} \cdot 2 \cdot tCCD + (tWL + tB + tWTR) + tRTW}.$$

The $(tWL + tB + tWTR)$ term represents the bus switching overhead from writes to reads, while the $tRTW$ term accounts for the read to write switching overhead.

IV. Evaluation Results

A. Evaluation Setup

We use MacSim [6], a multicore simulator to simulate a dual-core COTS platform. For the DRAM subsystem, we integrate DRAMSim2 [7] into MacSim. DRAMSim2 is a detailed DRAM simulator. It already implements a FR-FCFS arbitration with reorder threshold. We extend DRAMSim2 to also implement write batching scheme with parametric batch size. Without loss of generality, we use a DDR3-1600 DRAM device with a single rank and a single channel. However, the observations made in this paper equally applies to other DRAM devices. To model the mixed criticality nature, we simulate a heterogeneous system. We configure one core with an out-of-order pipeline to represent a high-performance non-critical core, while the other core is configured with an in-order pipeline, which is a common configuration in real-time critical cores to enable analyzability.

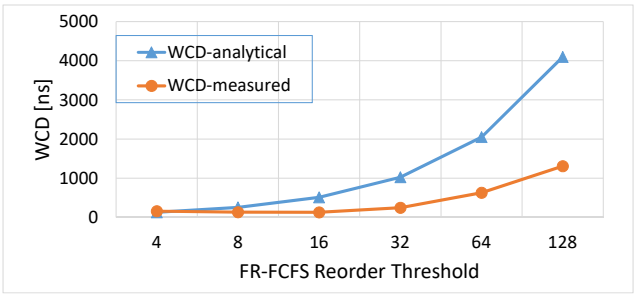
We use two applications: BW and latency [8]. In our setup, BW represents a non-critical application that is bandwidth-sensitive, while latency represents a critical application that is latency-sensitive. Each application runs in one of the two cores and share the same memory controller and off-chip DRAM with the other application. We study the effect of the two parameters detailed in Section III: reorder threshold and write batching in both BW and WCD.

B. Effect of FR-FCFS Reorder Threshold

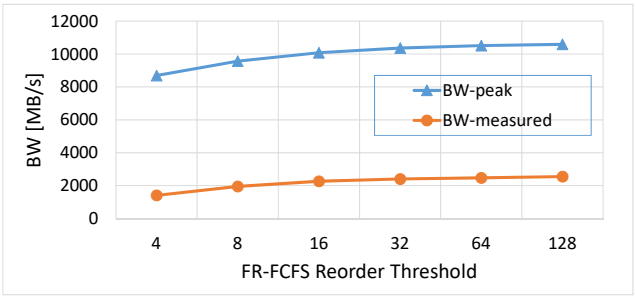
Figure 4 shows both the experimental and analytical results across various values of FR-FCFS reorder threshold. We make the following observations from the figure. 1) As anticipated in Section III, increasing the threshold value, the system BW increases (Figure 4b) at the expense of also an increase in the WCD (Figure 4a). 2) There is a big gap between the observed experimental bandwidth and the ideal peak bandwidth. As Figure 4b delineates, on average, the experimental BW is about 20% to 30% of the peak BW. The reason for this gap is that to achieve the peak BW (as explained in Section III), the memeory controller has to pressure the DRAM such that there are always N_{thr} pending requests to the open row all the time. This is usually not practically possible in traditional applications, which have a mix of open and close/conflict requests, and thus, leverage a lower BW. 3) With increasing the threshold value, the gap between the measured and analytical WCD increases. The intuition behind this observation is that to encounter the analytical WCD for higher threshold values, a higher number of requests have to be pending on the same time (equal to N_{thr}). As explained in observation 2, this becomes less practical with higher N_{thr} values. Accordingly, such analytical WCD is not observed in the experiments.

C. Effect of Write Batching

Figure 5 depicts the experimental and analytical results across various values of high watermark of the write batching mechanism. The low watermark is set to 0 in our experiments such that we can completely control the write batching through the higher watermark. From Figures 5a and 5b, we make the

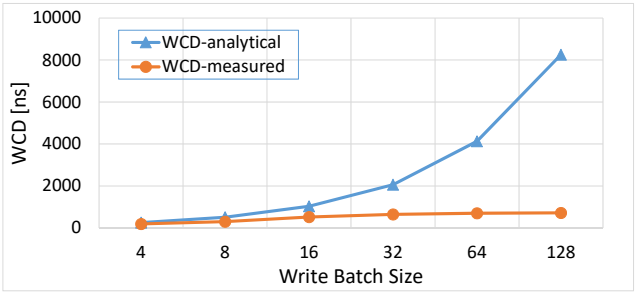


(a) Worst-case delay.

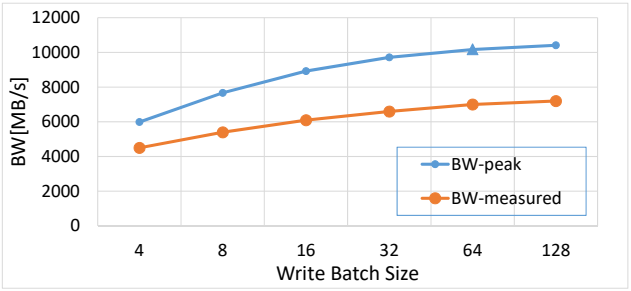


(b) Bandwidth.

Fig. 4: Effect of FR-FCFS reorder threshold.



(a) Worst-case delay.



(b) Bandwidth.

Fig. 5: Effect of Write Batching.

following observations. 1) Similar to the FR-FCFS threshold, increasing the write batch size, while increasing the BW, it also increases the WCD encountered by any request to the DRAM. As explained in Section III, increasing the write batch size, decreases the effect of the bus turnaround delay between request types (switching from read to write and vice versa); thus, it increases the effective BW. On the other hand, since in worst case, a read request has to wait for a complete write batch before it can get serviced, increasing the write batch size also increases the WCD. 2) By increasing the write batch size, the gap between the observed and analytical WCD increases (Figure 5a). This is because higher write batching values will require the applications to push more requests to the memory controller to saturate the write buffer, which may not occur in the application behavior. For instance, for write batch of 128, to observe the analytical WCD, a read has to arrive, where there are pending 128 writes in the write queue. In addition, the analytical WCD assumes that all these 128 writes are row conflicts. However, this is a pathological case that is extremely rare to occur on reality. With higher number of writes in the queue, it is more likely that some of these writes hit on the same row. Therefore, the actual observed WCD will be less than the analytical one. 3) The same observation holds for the BW in Figure 5b.

V. Related Work

The memory subsystem is considered the main bottleneck in many computing systems [3]. Accordingly, researchers have investigated solutions to mitigate this bottleneck in general-purpose computing (e.g. [9], [10]). Most of these solutions aim at increasing DRAM performance at the expense of latency predictability. Therefore, they are ill-suited for MCS with strict timing constraints. Recently, multiple solutions are introduced to reconcile this problem to achieve predictability at the first place, which are surveyed in [11]. The problem with these solutions is that they require a complete redesign of the memory controller, which entails them not applicable for COTS platforms. Three previous papers have investigated latency bounds for COTS memory controllers [5], [12], [13], among which only [5] takes the mixed criticality nature of modern embedded systems into account. Unlike our proposed study, all the three works, however, focus mainly on latency bounds of critical requests and not the system bandwidth. Recently, [14], [15] investigated Reduced-Latency DRAM (RLDRAM) as an alternative to the Double-Data-Rate (DDR) DRAM in the context of real-time systems [14], [15]. Unlike those works, this work focuses on commodity COTS platforms that utilize DDR DRAMs.

VI. Conclusion

We investigate the major sources of memory interference in COTS architectures upon deploying mixed criticality systems. We identified two main contributors to this interference: write batching and FR-FCFS arbitration. Both heavily affects the system predictability because of their deployed reordering mechanism. WE study their effects on both memory delay

and bandwidth. Our study shows that although theoretically increasing the amount of reordering done helps increasing the system peak (ideal) bandwidth), after certain point there are diminishing returns in practice. In addition, such increase will also significantly increase worst-case delays. Therefore, we conclude that reaching a compromise point between bandwidth and worst-case delay to meet all requirements of mixed criticality systems is the potential direction to explore. Such point will be application-dependent. We provide analytical equations as well as experimental evaluation to help designers deciding this point.

REFERENCES

[1] M. Hassan and H. Patel, "A framework for scheduling DRAM accesses for multi-core mixed-time critical systems," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2015.

[2] M. Hassan, H. Patel, and R. Pellizzoni, "PMC: A requirement-aware DRAM controller for multi-core mixed criticality systems," in *ACM Transactions on Embedded Computing Systems (TECS)*, 2016.

[3] O. Mutlu and L. Subramanian, "Research problems and opportunities in memory systems," *Supercomputing frontiers and innovations*, vol. 1, no. 3, pp. 19–55, 2015.

[4] D. S. Standard, "Jedec jesd79-3," 2007.

[5] M. Hassan and R. Pellizzoni, "Bounding dram interference in cots heterogeneous mpsocs for mixed criticality systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2323–2336, 2018.

[6] H. Kim, J. Lee, N. B. Lakshminarayana, J. Sim, J. Lim, and T. Pho, "Macsim: A cpu-gpu heterogeneous simulation framework user guide," *Georgia Institute of Technology*, 2012.

[7] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "Dramsim2: A cycle accurate memory system simulator," *IEEE computer architecture letters*, vol. 10, no. 1, pp. 16–19, 2011.

[8] 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 *2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2013, pp. 55–64.

[9] O. Mutlu and T. Moscibroda, "Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared dram systems," in *ACM SIGARCH Computer Architecture News*, vol. 36, no. 3. IEEE Computer Society, 2008, pp. 63–74.

[10] K. K.-W. Chang, D. Lee, Z. Chishti, A. R. Alameldeen, C. Wilkerson, Y. Kim, and O. Mutlu, "Improving dram performance by parallelizing refreshes with accesses," in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2014, pp. 356–367.

[11] D. Guo, M. Hassan, R. Pellizzoni, and H. Patel, "A comparative study of predictable DRAM controllers," *ACM Transactions on Embedded Computing Systems (TECS)*, 2018.

[12] 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 *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2014.

[13] H. Yun, R. Pellizzoni, and P. K. Valsan, "Parallelism-aware memory interference delay analysis for cots multicore systems," in *2015 27th Euromicro Conference on Real-Time Systems*. IEEE, 2015, pp. 184–195.

[14] M. Hassan, "On the off-chip memory latency of real-time systems: Is ddr dram really the best option?" in *2018 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2018, pp. 495–505.

[15] ——, "Reduced latency dram for multi-core safety-critical real-time systems," *Real-Time Systems*, pp. 1–36, 2019.