Demystifying the Characteristics of High Bandwidth Memory for Real-Time Systems

Kazi Asifuzzaman*, Mohamed Abuelala†, Mohamed Hassan† and Francisco J Cazorla*
*Barcelona Supercomputing Center, Spain
†McMaster University, Canada

***Abstract*—The number of functionalities controlled by software on every critical real-time product is on the rise in domains like automotive, avionics and space. To implement these advanced functionalities, software applications increasingly adopt artificial intelligence algorithms that manage massive amounts of data transmitted from various sensors. This translates into unprecedented memory performance requirements in critical systems that the commonly used DRAM memories struggle to provide. High-Bandwidth Memory (HBM) can satisfy these requirements offering high bandwidth, low power and high-integration capacity features. However, it remains unclear whether the predictability and isolation properties of HBM are compatible with the requirements of critical embedded systems. In this work, we perform to our knowledge the first timing analysis of HBM. We show the unique structural and timing characteristics of HBM with respect to DRAM memories and how they can be exploited for better time predictability, with emphasis on increased isolation among tasks and reduced worst-case memory latency.**

I. Introduction

Advanced software functionalities, like autonomous operation, build on complex Artificial Intelligence (AI) algorithms [1], yield unprecedented memory performance requirements that corresponds to the need to collect enormous amounts of data coming from sensors (e.g. cameras, LiDARs and radars) that must be processed at high rates. The bandwidth requirement of these applications have reached 60GB/s, and advanced applications are projected to require from 400GB/s to 1TB/s [2]. However, the scaling of the off-chip memory bandwidth has not been able to keep up with the increased processing capabilities. Currently, Graphics Double Data Rate (GDDRx) and Dynamic Random Access Memories (DRAMs) are used to provide high bandwidth, but they are still limited due to the narrow interface. Moreover, GDDRx uses high data rate to provide high bandwidth which comes at a considerable power cost [3].

High Bandwidth Memory (HBM) comes as an alternative implementing wider channels (128 bits) providing bandwidth up to 1TB/s [2], while consuming lower power and having higher capacity in comparison to GDDRx. Also, unlike the other variant of 3D-stacked DRAM — Hybrid Memory Cube (HMC), HBM is adopted as a JEDEC standard, implements much wider data bus and resides on the same silicon interposer as the processing unit. HBM, common in GPUs, FPGA-CPUs and System on Chips like the Xilinx UltraScale+, is better equipped to handle increased memory requirements of GPU and accelerator-based architectures [4], [5]. A recent study suggests the combination of low power consumption and high bandwidth make this category of memory ideal for embedded systems as well [6].

Besides the increasing average-performance requirements, applications used in automotive and avionics carry real-time requirements, where the total worst-case execution time (WCET) of all tasks should never exceed their respective dedicated deadlines. Therefore, the consolidation of HBM in critical systems requires careful analysis of its timing predictability properties with emphasis on timing isolation. This enables the safe execution of mixed-criticality software and predictable and tight worst-case memory access latency so that it can be shown that the benefits of HBM in average memory performance remain for worst-case memory performance.

To study the main memory system from the real-time perspective, two system components need to be analyzed: the memory device itself and the on-chip memory controller managing accesses to the device [7]–[9]. In this first analysis of HBM from a real-time systems perspective, we focus on the HBM device to capture its architectural characteristics and functionalities that can affect timing predictability such as device access behavior, timing properties and performance metrics. Additionally, by focusing on the device, the analysis and observations we provide in this paper are general and not limited to a specific scheduling technique deployed by the memory controller. Our analysis leads to key insights as a first essential step opening the door towards designing predictable HBM memory controllers. Overall, we make the following contributions.

1) We analyze HBM's device structure and the changes it brings to its functional and timing behavior compared to the DRAMs. This leads us to identify unique features of HBM from the latency-guarantee perspective that are not present in other DRAM-based memories. These are articulated as a set of observations (Section II).

2) We analyze the impact of the main HBM features to increase isolation or decrease worst-case latency (WCL). To that end, we develop an HBM specific latency formulation for certain HBM features and a set of illustrative time diagrams comparing DRAM and HBM. Our analysis shows that HBM can indeed represent a promising memory protocol for real-time embedded systems (Section III).

3) We perform an empirical comparison of the latest HBM standard (HBM2) and DDR4 DRAM with the state-of-the-art DRAMSim3 [10] simulator integrated with MacSim [11]: a detailed cycle-accurate processor simulator (Sections IV-B and IV-C). Our comparison assesses overall average performance, worst-case performance, and isolation properties using a wide set of representative as well as synthetic benchmarks and kernels.

4) We develop a timing simulation model derived from the JEDEC standards of HBM2 and DDR4 [12], [13]. The purpose of the model is twofold. It allows us to execute synthetically generated traces to further stress HBM2 and DDR4 differences. And it allows us to assess all HBM features including the recently introduced ones in the HBM2 standard [12] (such as pseudo-channels), which are not currently implemented in details in state-of-the-art memory simulators (Section IV-D). We release this model as open-source [14].

The rest of this paper is structured as follows. Section II analyzes the structural organization of HBM. Section III analyzes HBM's fit to real-time domain. Section IV presents experimental results. Section V describes the most relevant related work and Section VI presents the main conclusions of our work.

II. HBM Structure and Features

As the dominant memory technology for several decades, extensive information about DRAM is publicly available making it easier to analyze. This includes the real-time domain with well-documented studies summarizing DRAM analysis and proposals in this field [8], [15], [16]. In contrast, HBM has recently been adopted as an industry standard by JEDEC [12]. Therefore publicly available knowledge

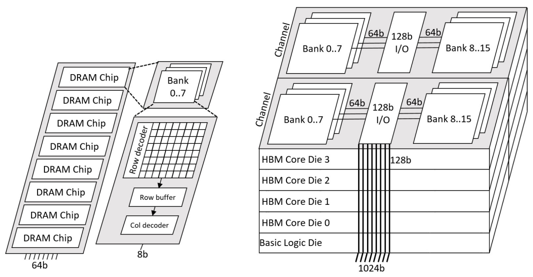

Fig. 1: Organization of DRAM (left) and HBM devices (right).

base of HBM is not nearly as mature as DRAM, making its real-time analysis very challenging. Since the core of HBM uses DRAM chips as its basic building blocks, we start with a brief background about DRAM, its commands, and associated timing constraints.

A. DRAM Background

DRAM is structured in independent channels. Each channel can have one or more ranks, and each rank has a number of banks (left side of Figure 1). Each DRAM bank is a 2-D matrix of memory cells comprising rows and columns. Each bank also has a distinct row buffer caching the most recently accessed row. Accesses to DRAM are performed in the form of DRAM commands. In particular, an *Activate* command (ACT) fetches an entire row from cells into the row buffer, then a Column Address Strobe command (CAS) is used to perform the read (RD) or write (WR) operation. If the row buffer holds a different row than the requested one, a *precharge* (PRE) command is needed to write back the old row to the cells, before activating the new one. The JEDEC DRAM standard mandates specific timing constraints between different commands that must be honored to ensure correct operation. Table I describes the different constraints, while Figure 2 describes the most relevant constraints. For instance, once a read (write) command is issued, tRL (tWL) cycles must elapse before the start of transferring the associated data.

B. HBM Device Organization

HBM organizes DRAM dies into stacks (Figure 1 right), in contrast to the planar layout of conventional DDRx DRAMs (Figure 1 left). This leads to a completely different structure and organization of HBM, enforcing significant changes in operation sequence and timing behavior in contrast to DRAM. These novel properties and features of HBM urge close analysis of the memory standard to determine what advantages HBM can offer for specific computing domains. The basic structure of HBM entails a base logic die, on which several (usually four or eight) DRAM core dies are stacked. The logic die accommodates external communication interface of the stack while stacked DRAM core dies are powered and connected to the logic die by *Through Silicon Vias* (TSVs). TSVs provide internal bandwidth to satisfy the external I/O pin bandwidth of the stack. A DRAM core die accommodates two independent channels, each of which are connected to the logic die with 128 non-shared TSV I/Os.

A standard HBM stack with four/eight dies totals 1024 TSV I/O connections to the logic die from its DRAM dies, see in Figure 1.

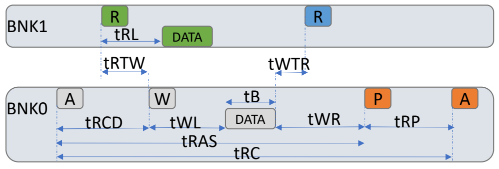

Fig. 2: DRAM commands and timing constraints.

TABLE I: JEDEC DRAM timing description [17]. Values are shown in cycles for DDR4 and HBM both running at 2133MHz. S/L refers to Short (different bank group) and Long (same bank group), respectively.

Parameter	Description	HBM	DDR4	Parameter	Description	HBM	DDR4
$tRCD$	ACT to CAS delay	14	15	$tRTP$ (S/L)	Read to PRE Delay	4/6	8/8
tRL	RD to Data Start	14	15	$tCCD$(S/L)	CAS to CAS delay	1/2	4/6
tWL	WR to Data Start	4	11	$tRRD$	ACT to ACT (diff bank)	6	6
tRP	PRE to ACT Delay	14	15	$tRTW$	RD to WR Delay		
$tRAS$	ACT to PRE Delay	34	36	$tWTR$ (S/L)	WR to RD Delay	6/8	3/8
tWR	Data end of WR to PRE	16	16	$tFAW$	4 bank activation window	30	32

These corresponds to the stack's external 1024 bit wide I/O interface that connects to the computing unit via interconnect circuitry (i.e. memory controllers) [18] (Observation 1). One processing unit can be connected to multiple HBM stacks as depicted in Figure 3a (Observation 2). DRAM on the other hand usually has an interface of 64 bit. This is because the DRAM's chip width is usually 4, 8, or 16 bit as stated in the DRAM chip datasheets. A DRAM channel or DIMM is usually composed of rank(s) where each rank has 8-chips totaling 64bit [7]. Each channel of the HBM interface is independently clocked and has its own command / data interface. Channels do not need to be synchronous to each other (Figure 3b and captured by Observation 3). Each channel features several DRAM banks. However, their organization and access granularity is different than of DRAM's. As stated in Observation 4, unlike DRAM, HBM is not organized into ranks. Instead, each HBM channel has a number of banks (e.g. 8 or 16), and each bank is divided into *half banks* [19] [20]. Each half, under the so called *legacy-mode*, produces 64 bits, so that a full bank produces 128 bits in each access (having the column width of 128 bits). Each set of half banks have 64 dedicated I/Os, see Channel 7 at Core Die 3 in Figure 3c. For DDR DRAMs, when a specific memory location is fetched via row, column and bank address, each chip across the rank supplies either 4, 8, or 16-bit (column width) data from the same location (based on the device type, which is so-called x4, x8, or x16). Hence, assuming 8 chips, this results in a 64 bit width per rank. In contrast, a single access to an HBM (logical) bank supplies 128 bits (Observation 5). In addition to its bandwidth benefits, we show in Section III that this feature can also help in reducing worst-case memory access latency.

Observation 1. *HBM offers wider connections to the processing unit (1024 bits) with respect to DRAM (e.g. 64 bits).*

Observation 2. *A processor can be connected to several independent HBM stacks residing on the same silicon interposer.*

Observation 3. *HBM channels, even those in the same core die, operate independently via private data and address/control signals and buses. Each HBM channel is connected to the logic die via a non-shared 128-bit TSV.*

Observation 4. *While HBM is based on DRAM banks, unlike DRAM it is organized into channels, pseudo channels, (logical) banks and half (physical) banks.*

Observation 5. *In DDR3/4 the access granularity is either 4, 8, or 16 bits per physical bank and 64 bits for the DRAM rank, while in HBM each bank supplies 128 bits.*

C. HBM's Core Memory Cells

Although HBM has a completely different structure w.r.t. conventional DRAMs, it still uses conventional DRAM cells as its memory storage core, i.e., HBM banks are organized (and accessed) the same way as in DRAMs. Accordingly, all commands have the same associated timing constraints as in conventional DRAM, which are dictated by the JEDEC standard both for DRAM [17] and HBM [12].

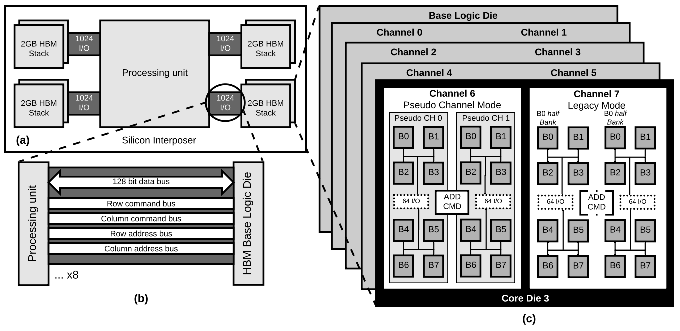

Fig. 3: (a) HBM stacks; (b) Per channel data/bus connections; (c) Internal structure of a 4-die HBM stack with arrangements of banks for pseudo channel mode (Channel 6) and legacy mode (channel 7).

D. Reduced Column-to-Column Timing

Another interesting characteristic of HBM is that, its tCCD is smaller than for DRAM's. tCCD is the timing of minimum burst duration, or the column-to-column timing constraint (i.e. minimum time between column operations). tCCD is mainly constrained by the time required to transfer the data on the data bus. In DDR DRAM, with a burst length of $BL = 8$, it requires $BL/2 = 4$ cycles to transfer the data from one access (i.e. one CAS command). Accordingly, $tCCD = 4$ is the minimum possible value to ensure correct operation. On the other hand, since HBM does not have $BL = 8$ and instead it supports up to $BL = 4$, we observe that $tCCD = 1$ or 2 in most HBM devices (depending on using either $BL = 2$ or $BL = 4$). This leads to Observation 6.

Observation 6. *Compared to DRAM, HBM has a reduced tCCD.*

E. Pseudo Channel Mode

A unique feature of HBM is that HBM channels can be operated in two modes — *legacy* and *pseudo channel*. The former corresponds to the conventional operation mode as described in the previous section. The latter, which is provided in the latest HBM standard (also known as HBM2) [21], further divides each channel into two sub channels formed with each set of *half banks* and 64 I/Os. This is illustrated for Channel 6 and Core Die 3 in Figure 3c. Pseudo channel mode requires the burst length (BL) to be 4, providing $64 \times 4 = 256$ bit or 32B per read/write command for each pseudo channel.

Observation 7. *In HBM2, a single memory access (i.e., CAS command) provides a 32B of data using $BL = 4$.*

Pseudo channels offer some degree of isolation so that accesses to the same channel but different pseudo channel have limited impact on each other. Assuming that each pseudo channel is provided to a different task would also limit the inter-task contention effects.

Observation 8. *Pseudo channels are semi-independent to each other: while they share the row and column command buses and clock inputs, they can decode and execute commands independently.*

F. Dual Command Interface

Driven by cost constraints, DRAM adopts a shared column and row address pins. On the other hand, with its wide I/O interface, HBM deploys separate column and row address pins. Leveraging this architecture, HBM employs dual address/command interface that allows column-related commands (i.e. read and write) to be issued simultaneously with the row-related commands (ACT and PRE) [22].

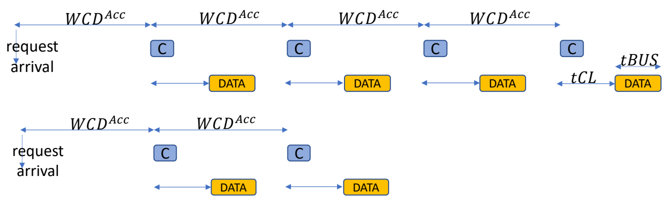

Fig. 4: Effect of bank partitioning.

Observation 9. *HBM has dedicated pins for column address that are separate from row address pins. Hence, read/write commands and addresses can be issued concurrently with row ACT/PRE commands.*

Despite employing separate row/column address and command bus, HBM needs to enforce usual DDR timings for ACT, RD/WR, PRE since these timing constraints are imposed by the internal physical structure of the memory cells, which is still DRAM-based. Hence, this does not change the timing of individual transactions (ACT, RD/WR, PRE). However, this feature can reduce address/command bus conflicts among different transactions as we explain in Section III.

G. Implicit Precharge

In DRAM standard, a row in a bank can only be *activated* after the previously open (active) row has been closed (precharged). Under DRAM close-page protocol, this translates into the sequence ACT-RD/WR-PRE. Under open-page if the row to access is open RD/WR commands are issued, otherwise it is required to issue PRE and ACT. Contrarily, when operating in pseudo-channel mode, HBM controller can ignore the PRE command and issue the ACT directly by leveraging the *implicit precharge* feature (Observation 10). However, it is important to know that all associated ACT-to-PRE and PRE-to-ACT constraints yet have to be satisfied.

Observation 10. *In pseudo channel operation HBM allows a subsequent ACT command to be issued to another row in the same bank without closing the previous row. In this case, the DRAM core itself will issue an implicit PRE command to close the first row before activating the second one.*

III. HBM for Real-Time Systems

In this section, we show how the main observations made on the previous section about HBM operation can be leveraged to (i) either increase isolation properties; and/or (ii) reduce the worst-case memory latency (WCL) bounds compared to existing state-of-the-art commodity-DRAM approaches. In each subsection we describe each feature assuming it is the only difference between HBM and DRAM. *By default <u>all</u> timing parameters remains the same as DRAM* except the ones being analyzed in that subsection. This allows to independently analyze the benefit of each feature as well as simplifies the discussion. Of course, the benefit of different features combine, which we analyze experimentally in Section IV.

A. HBM degrees of isolation

HBM can be leveraged to reduce contention among tasks in memory. We identify several levels of isolation from HBM stacks (Observation 2), HBM channels (Observation 3) and two pseudo-channels per channel (Observation 8). They can be smartly exploited to map the data/instructions of concurrently running tasks to reduce their contention interference as follows. 1) **Stack isolation**. At the top level, requests sent to different HBM stacks suffer no inter-task contention. This is so as each HBM stack operates independently. 2) **HBM (logical) bank isolation**. Similar to regular DRAM systems,

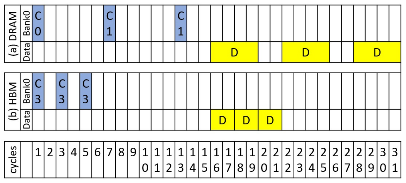

Fig. 5: Reduced tCCD effect.

HBM enables bank isolation so that requests from different tasks can be mapped to non overlapping banks. 3) **Half (logical) bank isolation**. As per Observation 8, pseudo-channels enable request to be sent to different half logical banks (also referred to as physical banks). Pseudo channels offer a reduced degree of isolation compared to channels as they share the row/column address and command buses and clock inputs. Each pseudo-channel has its own 64-bit data interface to the TSV. These isolation levels can be leveraged from the software [23] to increase predictability. Comparing high-level organizational structure, HBM stacks do not exist in traditional DRAMs (since the latter is not 3D stacked); HBM channels match DRAM channels; DRAM ranks do not exist in HBM; HBM bank isolation matches DRAM bank isolation, and HBM half-bank isolation (pseudo-channels) is not present in DRAM.

B. Reconciling the Isolation and Bandwidth Trade-offs

A common approach when using DRAMs for real-time systems is to partition banks among different requestors to minimize interference [8], [24], [25]. The main issue with this approach is that one request can be serialized into multiple accesses to serve the requested data. For instance, under bank partitioning, for the common DRAM data bus width of 16 bit [7] and a maximum burst length of $BL = 8$, a 64B cache line will require $64/(8*2) = 4$ accesses to serve the requested data from a single bank. For WCL analysis purposes, the 64B request will suffer an interference delay of $4 \times WCD^{Acc}$ from other requestors, where WCD^{Acc} is the worst-case interference delay suffered by a single access, see Figure 4(top) [1]. Lemma 1 generalizes this delay.

Lemma 1. *Under bank partitioning where each core is assigned BC private banks, a request with a data size of Y bytes targeting a DRAM with a data bus width of cw bits and a burst length of BL suffers a total WCD due to interference from other requestors that can be computed as shown in Equation 1 [8], [26].*

$$WCD^{tot}_{DRAM} = \frac{Y}{BL \times BC \times cw/8} \times WCD^{Acc} \qquad (1)$$

Note that as Lemma 1 shows, the number of transferred bytes depends on how many banks the request is interleaved across (Equation 1) [8], [15]. We make the following important remarks about Equation 1. 1) cw in modern DRAMs is limited to 4, 8, 16, or 32 bits. 2) BL can be either 4 or 8. 3) BC is the number of banks assigned to the requestor, which is upper bounded by the total number of available banks. In DDR3, a rank has a total of 8 banks, while in DDR4 it increased to 16. And 4) the value of WCD^{Acc} depends on the memory controller architecture [8]. For HBM, we build on Observation 1, 5, and 7 to tighten the WCD in Lemma 1. Observation 7 is based on the fact that each pseudo channel in HBM has a bus width of 64 bit (8 bytes). A $BL = 4$ results in 32B per access. Based on Observation 7, HBM can be utilized to deploy

[1]'A', 'C' and 'P' refers to ACT, CAS and PRE commands respectively; and the following number, if any, indicates the bank. 'D' represents data burst.

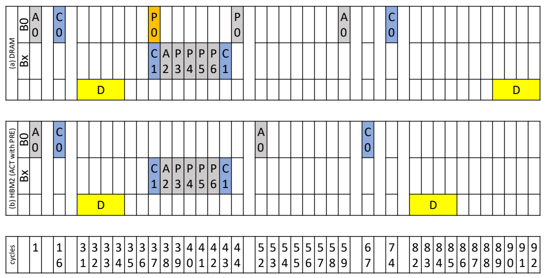

Fig. 6: HBM *Internal PRE* feature and its effect on latency.

bank partitioning to provide isolation among different requestors, while mitigating the effects of the reduced interleaving. To illustrate this observation, Figure 4(bottom) presents the same example used previously in DRAM but using HBM instead. Since HBM provides 32B per single access, a 64B cache line size will consume two accesses. This reduces the total WCD suffered by a memory request to half of the DRAM's case (compared to DRAM's column width of 16 bits).

Lemma 2. *Under bank partitioning scheme where each core is assigned a single private bank, a request with a data size of Y bytes targeting HBM suffers a total WCD due to interference from other requestors that can be computed as $WCD^{tot}_{HBM} = Y/32 \times WCD^{Acc}$*

C. Reducing CAS Latency

One of the main components of the WCD is the CAS latency [23], [27], defined as the latency affecting open requests targeting data available in the row buffer. For a sequence of N^{OP} row-open requests of same type (and hence composed of N^{OP} CAS commands), the total access latency of the sequence is $L^{CAS} = (N^{OP} - 1) \times tCCD + tCL + tBUS$, where tCL is the time between the CAS command and the start of its corresponding data transfer, while $tBUS$ is the required time to transfer the data ($= BL/2$). For this CAS latency, HBM (Observation 6) offers a significant advantage over regular DRAMs. This is because HBM has $tCCD = 1$ or 2 compared to $tCCD >= 4$ for DRAM. Accordingly, HBM can reduce the CAS latency component to at least half of its DRAM value. Figure 5 illustrates this by showing a sequence of three consecutive CAS commands. In case of DRAM (Figure 5a), $tCCD = 4$, which leads to a total of 31 cycles of service time of the sequence. In contrast, HBM (Figure 5b) with $tCCD = 2$ services the same sequence in only 21 cycles. In both scenarios we assume $tCL = 15$ (recall that we keep DRAM parameters except the one being analyzed).

D. Reducing Bus Conflicts

Implicit Precharge. Per Observation 10, HBM implements an implicit precharge that allows the controller to issue an ACT command to a bank while another row is already active. The internal circuitry of HBM device takes care of precharging the open row and maintaining correct operation. Figure 6 captures how this feature can lead to a reduced memory access latency by eliminating the PRE bus conflict. Each scenario shows three rows corresponding to the address/commands sent to of Bank0 (row1); address/commands sent to the other banks (row2); and the data over the bus, respectively. The scenario assumes that Bank0 has opened a row with the ACT command at cycle 1 and accesses it at cycle 16. While this row is open, another request at cycle 37 arrives to the same bank but different row. Hence, it has to close the open row by a PRE command before accessing the requested row. The PRE command at cycle 37 already satisfies the intra-bank constraint of CAS-to-PRE. Additionally, PRE commands do not have associated inter-bank constraints. Hence, it can theoretically be issued immediately to the DRAM device. However, there are other ready commands to the remaining 7 DRAM banks (C1, A2, P3, ... in the Figure). This delays the PRE command by 7 cycles to cycle 44. In this case, we say that the PRE command suffered a bus conflict delay (this is why it is shown in orange at cycle 37 indicating that it was not issued). For the HBM case, the controller does not need in the first place to issue that PRE command at cycle 37. It only needs to issue the ACT command after satisfying all timing constraints, namely, tRAS (36 cycles) and tRP (15 cycles). So, the next ACT command occurs at cycle 52. That request finishes at cycle 85 compared to 92 for the DRAM case. Again note that all timings are DRAM based, e.g. single-cycle ACT, other than the particular features analyzed.

Dual Command Bus Since the *Implicit Precharge* feature helps in removing the need to issue the PRE command, and hence eliminates its associated bus conflict delays, it remains to be discussed the bus conflicts in ACT and CAS commands. We now discuss how the dual command/address buses (feature from Observation 9) can help in eliminating these conflicts by an illustrative example. Figure 7 draws a scenario in which several ACT commands are sent to the memory device (namely to Bank 0, 1, and 2). Concurrently, Bank3 has multiple requests to the same open row that happens to arrive at the same time when the ACT commands to other banks become ready. Assuming that ACT commands have higher priority than CAS commands, each of the CAS commands to Bank3 has to be delayed by one cycle due to the command bus conflict. Note that a similar scenario would occur if CAS commands have higher priority, but in that case the ACT commands are the ones that are going to be delayed and hence suffer the bus conflict. On the HBM case, on the other hand, none of these conflicts occur since CAS and ACT commands can be issued simultaneously.

E. Reducing ACT Latency

One of the largest DRAM timing constraints affecting the WCD is the four-bank window constraint, $tFAW$. No more than 4 banks can be activated in a rank within a $tFAW$ time window. Figure 8 draws an illustrative example of 8 requests targeting different banks. When a single-rank DRAM is used, Figure 8a, the first four ACTs (A0–A3) are issued separated by the $tRRD$ constraint, which is 6 cycles in the used DRAM example. However, the fifth ACT command cannot be issued until the $tFAW = 32$ constraint is satisfied. This causes the idle gap in the data bus between cycles 53 and 62 in Figure 8a. Hence, the sequence of the 8 requests finishes at cycle 84.

Intuitively, using a DRAM with more than one rank and interleaving the ACT commands among them solves the $tFAW$ effect on the memory delays. While it is true that the $tFAW$ constraint does not apply to ACTs across ranks, rank interleaving in commodity DRAMs is a source for another delay that can result in a total suffered delay larger than the one added by the $tFAW$. This is so as the DRAM standard mandates that data transfers from different ranks has to be separated by at least $tRTR$ cycles. The $tFAW$ constraint in the single-rank case adds an extra delay of $tFAW - 4 \times tRRD$. This equals to $32-24 = 8$ cycles to the ACT commands in our used DDR device. On the other hand, the $tRTR$ constraint in the dual-rank case can add a delay of $7 \times 2 = 14$ cycles between the CAS commands compared to the single-rank case. Overall, this is a trade-off between ACT latency ($tFAW$) and CAS latency ($tRTR$).

In contrast to commodity DRAMs, HBM has no concept of ranks (Observation 4). Instead, it introduces the pseudo channel concept, where each channel can be divided into two pseudo channels as explained in Section II. ACTs targeting two different pseudo channels do not have to conform to the $tFAW$ constraint, while ACTs to the same pseudo-channel have to; this constructs our next observation.

Observation 11. *HBM's large $tFAW$ constraint does not apply to ACT commands targeting different pseudo channels.*

Unlike accesses to different ranks in DRAMs, there are no timing constraints that enforce a gap between data transfers from two pseudo channels (Observation 3). In other words, there is no $tRTR$-like constraint. Accordingly, by cleverly interleaving ACTs among the two pseudo channels, it is possible to stream data on the data bus without suffering the large $tFAW$ constraint. In Figure 8b, we apply this observation to our 8 requests sequence. As the figure illustrates, HBM enables the sequence to terminate at cycle 76, which is an example of how the effect of $tFAW$ on the WCL can be mitigated.

F. HBM Drawback: Two-cycle ACT commands

One drawback in HBM that affects the access latency as well as the total WCD upon accessing the device is that HBM requires the ACT command to consume two-cycles in the command bus compared to a single-cycle DRAM's ACT command. This also affects all the ACT-related timing constraints since the standard imposes that all these constraints have to be considered from the second cycle of the command and not the first one [12]. To illustrate the effect of this feature, in Figure 8c we show actual dual-cycle ACT commands of HBM (compared to a single-cycle ACT in Figure 8b). As the figure illustrates, now the same 8-request sequence finishes at cycle 84 similar to DRAM even though it does not suffer the $tFAW$ constraint. The intuition behind this result is that with the two-cycle ACT commands, an additional cycle per ACT command is suffered by the sequence. This adds 8 cycles to the single-cycle ACT HBM in Figure 8b entailing the sequence to finish at cycle 84.

IV. Evaluation

In this section we assess the worst-case performance, predictability (Sections IV-B and IV-D) and average-case performance (Section IV-C) of DDR4 and HBM2 . We model a DDR4 DRAM as the reference predictable DDR technology. To our knowledge there are no worst-case analysis developed for GDDR5/6.

A. Experimental Setup

Simulation Environment. We use the MacSim CPU simulator [11] integrated with DRAMSim3 [10] as its off-chip memory. In particular, we model a single channel DDR4-2133 with one rank, and eight banks; and an HBM2 stack with four dies, each with two channels and each channel having 16 banks. Both HBM2 and DDR4 devices are running at the same frequency (2133MHz). Details of the timing constraints are listed in Table I, while structural and system configurations are shown in Table II.

The core has an L1 cache with 16KB and a last-level cache (LLC) with 256KB. Unless otherwise stated, we use a cache line size of 64B for DDR4 and 512B for HBM since DDR4 can only transfer up to 64B per single transaction, while HBM provides transactions up to 512B. In this way we match the cache line size with the off-chip memory transaction size to guarantee that all the cache line bytes will be transferred by one request; and hence, maximize performance. Later in this section we show that using 512B cache line sizes for DDR4 do not affect our results.

Benchmarks. We aim at modeling real-time applications with varying memory demands. These are common in real-time on-board

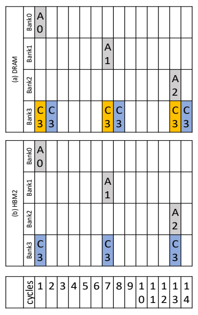

Fig. 7: HBM Dual command feature.

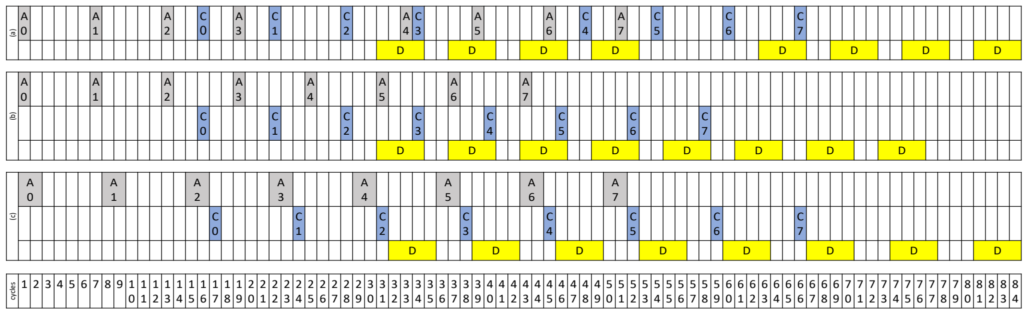
Fig. 8: Effect of HBM's pseudo-channel on $tFAW$ constraint. (a) DRAM, (b) HBM with pseudo-channel. (c) HBM with pseudo-channel but with actual two-cycle ACT command.

TABLE II: Cache and memory configuration parameters we use.

Cache Parameters {DDR4,HBM2}			Memory Structural Parameters		
PARAMETER	**L1**	**LLC**	**PARAMETER**	**DDR4**	**HBM2**
Cache Size	16KB	256KB	bankgroups	2	4
sets	{32, 4}	{256, 32}	banks_per_group	4	4
Associativity	8	16	rows	65536	32768
bank	1	1	columns	1024	64
Line size	{64, 512}	{64, 512}	device_width	16	128
			BL	8	4

space systems that encompass control applications and payload applications. While both have real-time and predictability requirements, control application are much less memory intensive [28].

We model control type of applications with benchmarks from EEMBC Autobench (EEMBC) [29] to mimic control functionalities of production automotive, industrial, and general-purpose systems.

We model payload applications by using a *Matrix Multiplication* (MXM) kernel that we configure with different total memory footprints: $2MB$, $4MB$, and $8MB$. MXM is one of the most common kernels for many functionalities like object detection libraries like YOLOv3 [30] and accounts for more than 65% of YOLO's execution time [31]. We also use the *Bandwidth* (BW_read and BW_write) and Latency micro-benchmarks from the IsolBench suite [32].

B. Worst Case Memory Latency

Determining the worst-case memory latency (WCL) of a task is primitive towards calculating its total worst-case execution time (WCET) since $WCET = WCCT + WCL$, where $WCCT$ is the worst computation time of the task on the processor. Since $WCL = WCL^{perReq} \times NumReqs$, two metrics are needed: the worst-case latency suffered by a single request WCL^{perReq}, and the worst-case total number of memory requests issued by the task $NumReqs$. In this section, in order to evaluate the behavior of HBM2 compared to DDR4 from a real-time perspective, we delineate both metrics.

1) Total Number of Read Requests: In most modern architectures, write requests to DRAM are only due to cache evictions of dirty cache line (cache write backs). Therefore, they do not stall the processor pipeline and hence, are not in the critical path of the task's WCET [27]. Therefore, we focus in this experiment on comparing the total number of read requests issued to both HBM2 and DDR4, which is shown in Figure 9. From Figure 9, we note that there is an overall significant reduction in the number of issued read requests for HBM compared to DDR4 with an average reduction of 6.5$\times$ for the EEMBC benchmarks and up to 8$\times$ for the BW benchmarks in Figure 9 (right). This is so since HBM by leveraging the wide interface (a total of 1024 bits) is able to transfer $512B$ per single transaction compared to the $64B$ transaction size of DDR4. When applications exhibit a high locality pattern, fetching larger data to their caches allows them to enjoy more cache hits and hence decrease the number of times they need to access the off-chip memory.

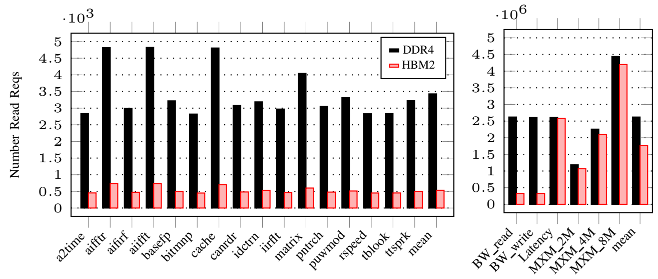

Fig. 9: Read requests of DDR4 vs HBM2 for EEMBC BMs (left), Synthetic and MXM BMs (right). Note the different scales.

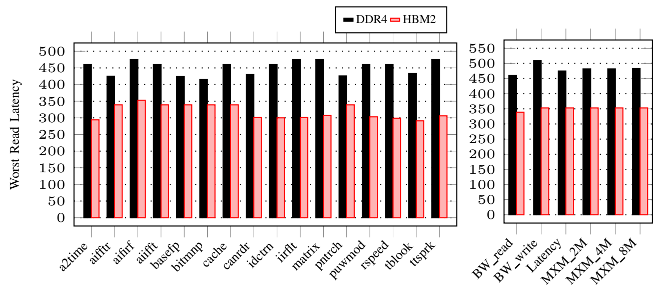

Fig. 10: Worst latency of read request DDR4 vs HBM2 for EEMBC BMs (left), Synthetic and MXM BMs (right).

Some of the benchmarks, namely Latency and MXM, exhibit a relatively smaller reduction in the number of requests. Investigating these benchmarks, we find that due to their access pattern, they suffer a lot of cache conflicts. As a result, they do not really benefit from the large request size that is brought to their cache hierarchy, which results in more requests issued to the off-chip memory.

2) Worst-Case Per-Request Read Latency: Figure 10 shows the observed worst-case latency, which is the maximum latency of a single request observed during the execution of the corresponding benchmark. Results show that HBM consistently introduces lower WCL compared to DDR4. HBM's WCL is in the range 291 - 353 cycles and DDR4's WCL is in the range 415 - 480 cycles in case of DDR4. Interestingly this the behavior for both control type of benchmarks (EEMBC) and payload (MxM and BW and Lat). This leads us to conclude that despite the aggressiveness of the application accessing memory, HBM2 provides reduced per-request WCL. We also evaluated 512B lines for DDR4 with 64B sectors in each line requested on demand. Our results confirm that the number of DRAM requests and latency per request is roughly unaffected.

3) Memory Isolation Opportunities: In all previous experiments, we assume that a request to HBM is interleaved across all the channels to utilize the wide interface of 1024 bits, and hence, transfer

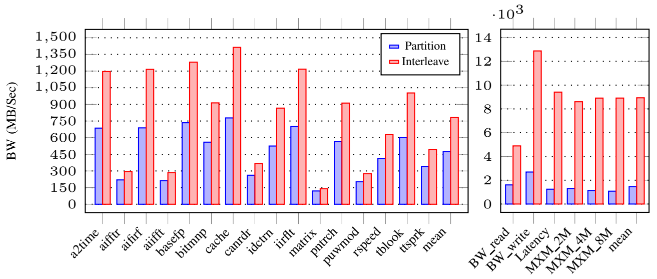

Fig. 11: Channel partitioning vs interleaving in HBM2 for EEMBC BMs (left), Synthetic & MXM BMs (right). Note the different scales.

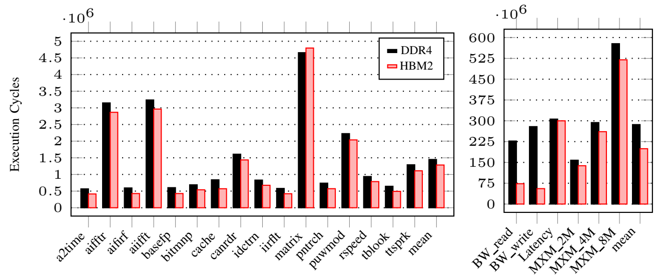

Fig. 12: Execution cycles of DDR4 vs HBM2 for EEMBC BMs (left), Synthetic and MXM BMs (right). Note the different scales.

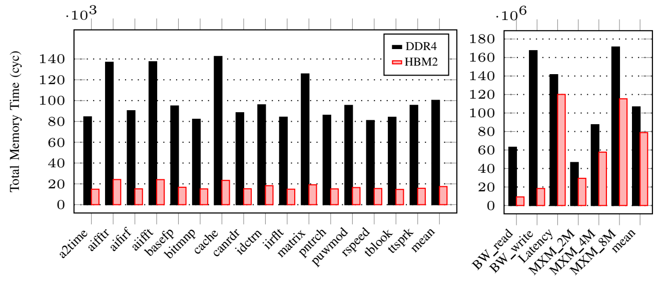

Fig. 13: Total off-chip memory time for DDR4 & HBM2 for EEMBC BMs (left), Synth. & MXM BMs (right). Note the different scales.

$512B$ per request. This is the common approach in high-performance systems to increase the off-chip memory bandwidth. However, a common approach in real-time systems is to enforce isolation by partitioning the off-chip memory among requestors to minimize memory interference [8]. As discussed in Section III-A, HBM offers different degrees of isolation. Nonetheless, it is well established that achieving isolation by partitioning off-chip entities among different requestors comes at the cost of performance. Although as we discussed in Section III-B, HBM can reconcile this trade-off, it is still expected that the trade-off is not fundamentally resolved. Therefore, in this section, we evaluate the BW loss incurred if the application accesses a single HBM channel (resembling channel isolation) compared to being interleaved across all the channels. Figure 11 shows our findings. We can see that achieving isolation comes at the expense of a BW degradation of 15%–45% (35% on average) for the EEMBC benchmarks and 67%–87% (78% on average) for the BW-intensive synthetic benchmarks. The bandwidth-latency trade-off is a use-case dependent and, hence, depends on the running set of the applications. For example, if the number of contending requestors is less than the number of available channels, bandwidth can improve by assigning multiple (yet exclusive) channels. Deciding the exact ideal compromise point of this trade-off is not the focus of the paper.

C. Average-Case Performance

The potential utilization of HBM in real-time systems targets domains requiring both guarantees and considerable average performance. We next compare HBM2 and DDR4 average performance.

Figure 12 depicts the execution time of all the benchmarks using DDR4 and HBM. Despite EEMBC benchmarks presented in Figure 12 (left) are not memory intensive, HBM2 shows better performance than DDR4 with an average improvement of 12%. On the other hand, for memory-stressing synthetic benchmarks (Figure 12 (right)) the performance gap between HBM2 and DDR4 is clearer with an improvement up to $4\times$ for the BW_write benchmark. The Latency benchmark shows comparable performance for HBM2 and DDR4 with HBM2 better execution time of only 3%. For MXM we see improvements of 1.17%.

As expected, for benchmarks with limited memory utilization or reduced locality, the memory performance is not a key factor in application performance. For instance, the Latency benchmark has a random pointer-chasing-like pattern that does not benefit from HBM's high bandwidth since it exhibits very low locality. In order to assess the memory behavior, Figure 13 shows the total memory time spent by different benchmarks accessing the off-chip memory. We see that the gap between HBM2 and DDR4 significantly increases compared to Figure 12. For instance, HBM shows on average a $5\times$ less memory time compared to DDR4 for the EEMBC benchmarks. The gap reaches up to $6\times$ for EEMBC benchmarks and the MXM benchmarks, and up to $9\times$ for the BW_write synthetic benchmark. This confirms that HBM2 consistently improve the memory performance of DDR4, while the benefit of this depends on how the performance or programs depends on memory performance.

D. Synthetic Experiments

Memory simulators, including DRAMSim3 [10], RAMulator [33], and GEM5 [34], have been mainly used for the evaluations of techniques for high-performance systems, and naturally model elements that affect average performance, while overlooking some details of the features that might have an impact on the worst-case latency. For instance, we find that the pseudo-channel mode is either not supported at all or is partially (and abstractly) implemented by these simulators. Abstracting HBM features can support investigations related to average-case behavior; however, analysis for real-time systems mandates a complete and accurate modeling of the timing behavior of HBM. It is also worth mentioning that although there has been a recent DRAM simulator targeting real-time systems [35], it unfortunately does not support HBM in its current version.

For this purpose, we develop a C++ simulation model derived from the JEDEC standard both for HBM2 and DDR4, which we release as open-source. It implements the state machine of the various timing constraints for both protocols taking into account all the features we covered in this paper including the recent pseudo-channel feature of HBM2. A final important note is that all the results presented in Sections IV-B and IV-C are not affected by any means by the missing pseudo-channel modeling of HBM2 in DRAMSim3 since all the experiments are assuming HBM2 used in the legacy mode where a request is interleaved across all the channels and accesses the 16 logical banks of each channel as illustrated in Section II.

In addition to the developed simulation model, we also developed specific tests to stress each of the covered features and feed these tests to the simulator to assess the behavior of the two protocols. We develop four different synthetic tests for four different features as shown in Figure 14: *Dual_CMD*, *partition*, *Reduced_tCCD*, and $tFAW$. Each of these four tests performs 1,024,000 read accesses. 1) *Dual_CMD* is a test that measures the impact of the dual command

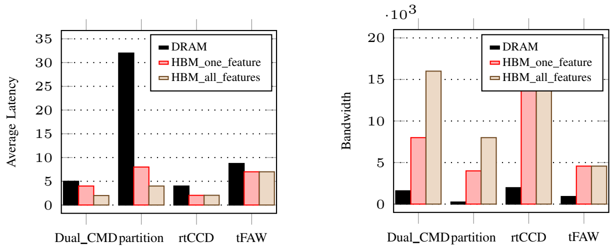

Fig. 14: Isolated and combined analysis of the impact of different HBM features: avg. request latency(left), overall bandwidth (right).

feature (Section III-D). This is done by issuing a stream of open-row requests to one bank; hence, consisting mainly of CAS commands. Simultaneously, an interfering stream of ACT commands to other banks is crafted such that there is always a ready interfering command in the same cycle as each of the CAS commands. 2) *Partition* is a test where the emulated core running the trace is assigned a single bank and each request has a data size of $64B$. This is useful to study the impact of the wide data bus width of HBM on latency (Section III-B). 3) *Reduced_tCCD* is a test of open requests targeting same row, and hence, consisting mainly of CAS commands. Unlike *Dual_CMD*, there is no interfering streams to avoid bus conflicts in order to focus only on the reduced tCCD feature of HBM (Section III-C). 4) *tFAW* is a test stressing the tFAW constraint by issuing close-row requests to different banks; hence, containing ACT commands; hence, assessing the pseudo-channel feature (Section III-E).

Figure 14 shows both the average per-request latency and the bandwidth, respectively for both DRAM (DDR4 in the experiments) and HBM. For each test we use two different models of HBM. The first one (*HBM_one_feature*) only models the considered feature in the corresponding test while all other properties are exactly the same as DRAM. We do this for the sake of studying the effect of this specific feature in isolation. The second model (*HBM_all_features* or simply *HBM*) captures all the features of a regular HBM. For instance, for *Reduced_tCCD* test, *HBM_one_feature* is completely identical to DRAM parameters except $tCCD$ value, which models the HBM's one, while *HBM_all_features* models all the parameters of HBM regardless of the test.

The following conclusions can be obtained from Figure 14. 1) *Dual_CMD* and *Reduced_tCCD* features notably contribute to bandwidth improvement – this complies with our analysis presented in Section III-D and III-C, respectively. Due to *Dual_CMD* feature, HBM can issue ACT and CAS commands in parallel, effectively eliminating bus conflicts, therefore suffering minimum stall cycles and improving sustained bandwidth. This feature also slightly improves execution time for HBM. 2) *Reduced_tCCD* also allows HBM to issue consecutive CAS commands in shorter time, therefore increasing throughput, as well as contributes to achieve reduced execution time. 3) The bank *partition* feature (analyzed in III-B) enables HBM to provide 32B per single access, therefore filling a 64B cache line in just two CAS commands, in comparison to four for DRAM – resulting in a huge reduction in execution time. This feature also considerably improves bandwidth. 4) Feature *tFAW* refers to the timing requirement to open maximum four active windows in a certain time frame as explained in Section III-E. Since HBM improves this requirement, we see it achieves increased bandwidth and reduced execution time w.r.t DRAM. Combined with the analysis in Section III, these results provide insights on the benefits of all the HBM features including the recently introduced ones in HBM2.

V. Related Work

HBM. Some initial works present HBM as an emerging memory standard that can provide bandwidth superior to 256GB/s as well as offers lower power consumption [22]. More recent works compare HBM and DDR for high performance systems [4]. [18] presents the challenges of capacity scaling of HBM device by stacking more DRAM dies to make a taller stack. Other works focus on studying the power consumption of HBM2 compared to DRAM [36], which confirm that HBM2 consumes significantly less energy than DDR4, for per-bit transmitted. While we recognize the importance of energy-consumption in embedded systems, in this first work, we focus on HBM features affecting time predictability. At the software level, several techniques propose HBM's application-specific improvements. For instance, Maohua [37] et al. implement a convolutional neural network (CNN) and breadth-first search (BFS) – two data intensive applications on a HBM-enabled GPU platform. Bingchao et al. describes *pseudo channel mode* and *dual-command* features of HBM, and concludes that these features do not significantly contribute to an average-case performance improvement [4].

DRAM memory predictability. There exists an extensive literature to handle memory contention in real-time systems. A commonality in these proposals is that they do not propose hardware changes to the memory device. Instead they address contention with i) software solutions that increase the isolation among tasks in memory; and/or ii) hardware changes of the memory controller. Regarding the former we refer the reader to [23] for a detailed summary of the state of the art. Software solutions build around approaches to increase isolation, e.g. via bank partitioning among processors (e.g. [38]), and controlling access counts (e.g. [39]). A comparison of hardware solutions is presented in [8] covering a wide set of works on DRAM controller designs for predictability and/or balancing predictability and performance (e.g. [24], [28], [40]). Beyond DDR DRAMs, Hassan [9] identifies that DDR DRAMs suffer inherent limitations to achieve reasonable predictability and results highly variable access latencies with over pessimistic bounds. The study proposes to use Reduced Latency DRAM (RLDRAM) to address these challenges.

Summary. While several works analyze HBM and predictability in DRAM-based systems, in this work we bring these two worlds together and tackle for the first time HBM usage in real-time systems.

VI. Conclusions

We analyzed some distinctive functional and architectural features of HBM and their benefits in terms of isolation and reduction on memory worst case latency. We empirically showed the benefits of some HBM features via a reference DRAM memory simulator. We also developed a worst-case timing model derived from the JEDEC standards of HBM and DDR4 capturing HBM features not currently properly modeled in DRAM simulators. Overall, this work provides a good understanding of the benefits of HBM over DRAM for real-time systems in worst-case performance and isolation and sets a solid basis for future techniques to fully embrace HBM in real-time systems.

VII. Acknowledgments

This work has been partially supported by the Spanish Ministry of Science and Innovation under grant PID2019-107255GB-C21/AEI/10.13039/501100011033; the European Union's Horizon 2020 Framework Programme under grant agreement No. 878752 (MASTECS) and agreement No. 779877 (Mont-Blanc 2020); the European Research Council (ERC) grant agreement No. 772773 (SuPerCom); and the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

[1] R. Pujol, H. Tabani, L. Kosmidis, E. Mezzetti, J. Abella, and F. J. Cazorla, "Generating and exploiting deep learning variants to increase heterogeneous resource utilization in the NVIDIA xavier," in *31st Euromicro Conference on Real-Time Systems, ECRTS , Stuttgart, Germany*, S. Quinton, Ed. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[2] M. Jung, S. A. McKee, C. Sudarshan, C. Dropmann, C. Weis, and N. Wehn, "Driving into the memory wall: the role of memory for advanced driver assistance systems and autonomous driving," in *Proceedings of the International Symposium on Memory Systems, MEMSYS, Old Town Alexandria, VA, USA*, B. Jacob, Ed. ACM, 2018.

[3] B. Oh, N. S. Kim, J. Ahn, B. Li, R. G. Dreslinski, and T. N. Mudge, "A load balancing technique for memory channels," in *Proceedings of the International Symposium on Memory Systems, MEMSYS, Old Town Alexandria, VA, USA*, B. Jacob, Ed. ACM, 2018.

[4] B. Li, C. Song, J. Wei, J. H. Ahn, and N. S. Kim, "Exploring new features of high-bandwidth memory for gpus," *IEICE Electronic Express*, 2016.

[5] M. Zhu, Y. Zhuo, C. Wang, W. Chen, and Y. Xie, "Performance evaluation and optimization of hbm-enabled GPU for data-intensive applications," *IEEE Trans. Very Large Scale Integr. Syst.*, 2018.

[6] M. Jung, C. Weis, and N. Wehn, *The Dynamic Random Access Memory Challenge in Embedded Computing Systems*, 2021.

[7] B. L. Jacob, S. W. Ng, and D. T. Wang, *Memory Systems: Cache, DRAM, Disk*. Morgan Kaufmann, 2008.

[8] D. Guo, M. Hassan, R. Pellizzoni, and H. D. Patel, "A comparative study of predictable DRAM controllers," *ACM Trans. Embedded Comput. Syst.*, 2018.

[9] M. Hassan, "On the off-chip memory latency of real-time systems: Is DDR DRAM really the best option?" in *IEEE Real-Time Systems Symposium, RTSS, Nashville, TN, USA*. IEEE Computer Society, 2018.

[10] S. Li, Z. Yang, D. Reddy, A. Srivastava, and B. Jacob, "DRAMsim3: a Cycle-accurate, Thermal-Capable DRAM Simulator," in *IEEE Computer Architec. Letters*, 2020.

[11] H. Kim, J. Lee, N. B. Lakshminarayana, J. Sim, J. Lim, and T. Pho, "Macsim: Simulator for heterogeneous architecture," 2012.

[12] JEDEC, "High Bandwidth Memory DRAM (HBM1, HBM2)," 2020.

[13] J. S. S. T. Association *et al.*, "Jedec standard: Ddr4 sdram," *JESD79-4, Sep*, 2012.

[14] "HBM DDR Synthetic Model." [Online]. Available: https://gitlab.com/FanosLab/hbm_ddr_synth_model

[15] B. Akesson and K. Goossens, *Memory controllers for real-time embedded systems*. Springer, 2011.

[16] S. Goossens, K. Chandrasekar, B. Akesson, and K. Goossens, *Memory controllers for mixed-time-criticality systems*. Springer, 2016.

[17] D. S. JEDEC, "JEDEC jesd79-3b," 2008.

[18] A. F. Farahani, S. Gurumurthi, G. H. Loh, and M. Ignatowski, "Challenges of high-capacity DRAM stacks and potential directions," in *Proceedings of the Workshop on Memory Centric High Performance Computing, MCHPC@SC, Dallas, TX, USA*. ACM, 2018.

[19] J. Kim and Y. Kim, "HBM: memory solution for bandwidth-hungry processors," in *IEEE Hot Chips 26 Symposium (HCS), Cupertino, CA, USA*. IEEE, 2014.

[20] D. U. Lee et al, "22.3 A 128Gb 8-High 512GB/s HBM2E DRAM with a Pseudo Quarter Bank Structure, Power Dispersion and an Instruction-Based At-Speed PMBIST," in *IEEE International Solid- State Circuits Conference - (ISSCC)*, 2020.

[21] JEDEC, "High Bandwidth Memory (HBM) DRAM," 2013.

[22] H. Jun, J. Cho, K. Lee, H.-Y. Son, K. Kim, H. Jin, and K. Kim, "Hbm (high bandwidth memory) dram technology and architecture," in *IEEE International Memory Workshop (IMW)*, 2017.

[23] M. Hassan and R. Pellizzoni, "Analysis of memory-contention in heterogeneous cots mpsocs," in *32nd Euromicro Conference on Real-Time Systems, ECRTS*, 2020.

[24] L. Ecco, S. Tobuschat, S. Saidi, and R. Ernst, "A mixed critical memory controller using bank privatization and fixed priority scheduling," in *IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications, Chongqing, China*, 2014.

[25] L. Ecco and R. Ernst, "Improved DRAM timing bounds for real-time DRAM controllers with read/write bundling," in *IEEE Real-Time Systems Symposium, RTSS, San Antonio, Texas, USA*. IEEE Computer Society, 2015.

[26] Z. P. Wu, Y. Krish, and R. Pellizzoni, "Worst case analysis of dram latency in multi-requestor systems," in *IEEE Real-Time Systems Symposium (RTSS)*, 2013.

[27] H. Yun, R. Pellizzoni, and P. K. Valsan, "Parallelism-aware memory interference delay analysis for COTS multicore systems," in *27th Euromicro Conference on Real-Time Systems, ECRTS Lund, Sweden*. IEEE Computer Society, 2015.

[28] J. Jalle, E. Quiñones, J. Abella, L. Fossati, M. Zulianello, and F. J. Cazorla, "A dual-criticality memory controller (dcmc): Proposal and evaluation of a space case study," in *Proceedings of the IEEE 35th IEEE Real-Time Systems Symposium RTSS, Rome, Italy*. IEEE Computer Society, 2014.

[29] J. A. Poovey, T. M. Conte, M. Levy, and S. Gal-On, "A benchmark characterization of the EEMBC benchmark suite," *IEEE Micro*, 2009.

[30] H. Tabani, R. Pujol, J. Abella, and F. J. Cazorla, "A cross-layer review of deep learning frameworks to ease their optimization and reuse," in *2020 IEEE 23rd International Symposium on Real-Time Distributed Computing (ISORC)*. IEEE, May 2020, pp. 144–145.

[31] F. Fernandes dos Santos, L. Draghetti, L. Weigel, L. Carro, P. Navaux, and P. Rech, "Evaluation and mitigation of soft-errors in neural network-based object detection in three gpu architectures," in *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE, Jun. 2017, pp. 169–176.

[32] H. Yun, *IsolBench*. [Online]. Available: https://github.com/CSL-KU/IsolBench

[33] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A fast and extensible DRAM simulator," *IEEE Comput. Archit. Lett.*, 2016.

[34] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, 2011.

[35] R. Mirosanlou, D. Guo, M. Hassan, and R. Pellizzoni, "Mcsim: An extensible dram memory controller simulator," *IEEE Computer Architecture Letters (CAL)*, 2020.

[36] S. S. N. Larimi, B. Salami, O. S. Unsal, A. C. Kestelman, H. Sarbazi-Azad, and O. Mutlu, "Understanding power consumption and reliability of high-bandwidth memory with voltage underscaling," 2020.

[37] M. Zhu, Y. Zhuo, C. Wang, W. Chen, and Y. Xie, "Performance evaluation and optimization of hbm-enabled GPU for data-intensive applications," 2018.

[38] H. Yun, R. Mancuso, Z. P. Wu, and R. Pellizzoni, "PALLOC: DRAM bank-aware memory allocator for performance isolation on multicore platforms," in *20th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS, Berlin, Germany*. IEEE Computer Society, 2014.

[39] A. Agrawal, G. Fohler, J. Freitag, J. Nowotsch, S. Uhrig, and M. Paulitsch, "Contention-aware dynamic memory bandwidth isolation with predictability in COTS multicores: An avionics case study," in *29th Euromicro Conference on Real-Time Systems, ECRTS Dubrovnik, Croatia*, M. Bertogna, Ed. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[40] H. Kim, D. Broman, E. A. Lee, M. Zimmer, A. Shrivastava, and J. Oh, "A predictable and command-level priority-based DRAM controller for mixed-criticality systems," in *21st IEEE Real-Time and Embedded Technology and Applications Symposium, Seattle, WA, USA*. IEEE Computer Society, 2015.