An Enhanced COTS DRAM Controller Design for High-Performance Real-Time Systems

Prathap Kumar Valsan, Heechul Yun
University of Kansas, USA.
heechul.yun@ku.edu

I. Introduction and Motivation

In a modern Commercial-Off-The-Shelf (COTS) multicore architecture, a core often generates multiple concurrent memory requests (thanks to techniques such as non-blocking cache, super-scaler, and out-of-order execution) to hide long off-chip memory access latency. This, together with the increased number of cores, puts a high bandwidth pressure on the main memory subsystem [5].

To cater high memory bandwidth demand, modern main memory (DRAM) is organized into ranks and each rank is divided into multiple banks, which can be accessed in parallel. Each bank is comprised of rows and column; to access data in a row, an activate command (ACT) must be issued to load the data in the row-buffer of the bank, before subsequent read or write commands (RD, WR) can be issued. To access data in a differnt row, a pre-charge (PRE) command must be issued to write-back the data in the row-buffer into the originating DRAM row. Accessing data already in the row buffer (row-hit) is faster than data in a different row (row-miss).

A COTS DRAM controller typically employs an interleaved bank addressing strategy to maximize bank parallelism. Memory requests from the cores (or DMA devices) are buffered inside the DRAM controller's internal buffers and the DRAM controller and issues DRAM commands through the shared command bus connected to the DRAM chips while respecting all the timing constraints specified by the JEDEC standard [6]. The queued memory requests are often re-ordered to maximize memory throughput; the mostly commonly used first-ready first-come-first-serve (FR-FCFS) [11] scheduling algorithm prioritizes: (1) Row-hit requests over row-miss requests; (2) Older requests over younger requests.

Unfortunately, aforementioned COTS memory organization and DRAM controller design are very poor at providing predictable timing due to several reasons: First, each core can access any bank at any time. If, for example, all cores try to access the same bank at the same time, they will suffer a very long delay due to the loss of bank-level parallelism. Second, because the FR-FCFS scheduling algorithm re-orders memory requests in the DRAM controller's buffers to maximize memory throughput, a request from a high priority task can be starved by the requests from the low priority threads.

Because of these reasons, real-time DRAM controller proposals either partition the banks on a per-core basis [12], [7], [10] at the hardware level or increase the granulaity of each memory request so that each request always access all banks in the DRAM [1], [9]. Also, instead of the FR-FCFS algorithm, they use predictable scheduling alrogithm such as round-robin [9] or CCSP [2]. While these real-time DRAM controller desigsns provide predictable memory timing, they generally suffer much decreased average memory thoughput.

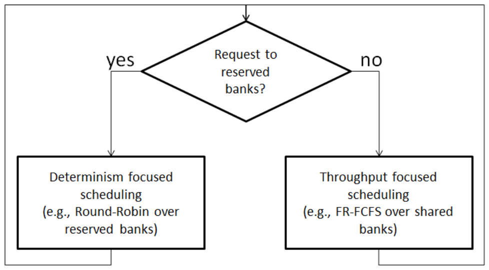
Request to reserved banks?
yes
no
Determinism focused scheduling (e.g., Round-Robin over reserved banks)
Throughput focused scheduling (e.g., FR-FCFS over shared banks)

Fig. 1: Two-level hierarchical memory scheduling algorithm

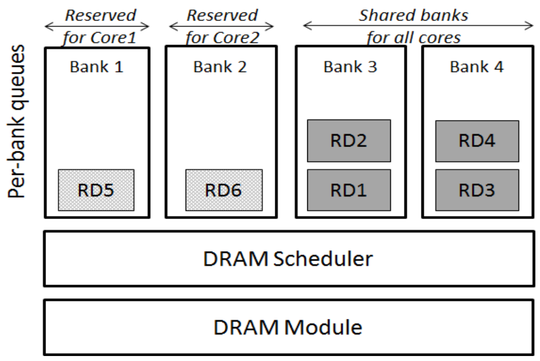

Fig. 2: An example: Requests to reserved banks (Bank 1-2) are prioritized over previously arrived requests to shared banks (Bank 3-4)). (Note: requests are numbered in the arrival time order.)

II. An Enhanced COTS DRAM Controller Design

In this work, we propose a DRAM controller design that can provide high time predictability when needed for real-time tasks but also strive to provide high average performance for non-real-time tasks. We aim to achieve this goal through a close collaboration between the OS and the DRAM controller.

A. OS based DRAM bank partitioning

First, we *partially* partition DRAM banks via a software (OS) based bank partitionnig method [13]. Instead of statically partitioning all banks as in [10], [12], [7], we reserve a small number of banks for each core but share the rest of the banks for all cores. Each core can access both reserved banks (dedicated for the core) and shared banks depending on the physical addresses of the memory pages allocated memory by the OS.

B. *Two-level higherarchical memory scheduling*

The DRAM controller employs the following two-level scheduling algorithm. First, it prioritizes memory requests for the reserved banks over shared banks. Second, arbitration among deterministic memory requests is determined on a round-robin basis over the reserved banks. Third, memory requests to the shared banks are serviced using the standard FR-FCFS algorithm. Figure 1 shows the flowchart of the proposed algorithm.

C. *Example*

Figure 2 shows a running example of the proposed memory contoller design. In this example, six requests, RD1-6 (numbered in their arrival order), are initially in the DRAM controller's request queues. Note that Bank1 (Bank2) is reserved for Core1 (Core2), while Bank 3 and 4 are shared by all cores. Assuming all requests are row-hit requests, if the standard FR-FCFS algorithm is used, the requests will be processed in the arrival order—i.e., RD1, RD2, ..., RD6. In our framework, however, RD5 and RD6 will be prioritized because they are targetting the core-reserved banks. Note that if there are no deterministic memory requests in the buffers, our scheduler works exactly the same as existing COTS memory controllers (e.g., FR-FCFS).

The expected benefits of our approach is three-fold: 1) Real-time tasks can easily allocate memory from the reserved banks (via the OS) to achieve highly predictable timing; 2) Non-real-time tasks can still achieve high average perforamnce by allocating memory from the shared banks. Note that the number of DRAM banks are typically significantly bigger than the number of cores (e.g., 32 banks vs. 4 cores) and most applications do not show performance improvement beyond a certain number of banks [13], [8]; 3) Configuration is highly flexible (via the OS at run-time).

III. Preliminary Evaluation Results

In this section, we present our simulation setup and some preliminary simulation results.

We are currently implementing the proposed DRAM controller design in the Gem5 full-system simulator [3]. We have modelled a Quad core armv7 system. Each core has a private L1 cache (32K-I/32K-D) and all cores share a 1MiB L2 cache (LLC). Both L1 and L2 are non-blocking caches with 4 and 8 MSHRs, respectively, which determine the local and global limit of outstanding memory requests. We use a realistic event-based memory controller [4], which captures important timing and structural constraints of COTS memory controllers and modified it to prioritize OS specified banks (currently hard coded to prioritize bank0). On the simulator, we run a full Linux 3.14 kernel and patched it to use the PALLOC [13]

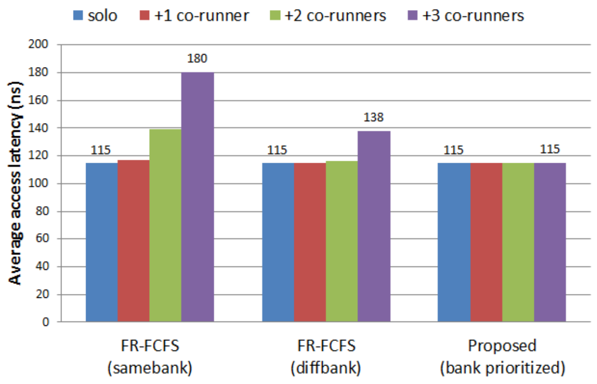

Fig. 3: Average memory access latencies of *Latency* benchmark with memory intensive co-runners.

memory allocator, which allows us to partition DRAM banks through the CGROUP interface.

As preliminary work, we used a micro benchmark *Latency* (linked list travsersal) and memory intensive co-runners to evaluate the proposed two-level scheduling algorithm. We measured the average memory access latency of the Latency benchmark in the presence of memory intensive co-runners; we varied the number of co-runners from 0 to 3. Note that all tasks are single-threaded and each core runs a single task. We repeat the experiment on three different DRAM controller and memory configurations: In FR-FCFS (samebank), all cores access the same DRAM bank to simulate the worst-case scenario; in FR-FCFS (diffbank), each core is assigend its own dedicated DRAM bank; in proposed, memory requests to bank 0 are alwasy prioritized over requests to other banks and the memory pages of Latency are allocated on the bank0 while co-runners are allocated on the rest of the banks using PALLOC.

Figure 3 shows the results. As expected, the proposed scheduling algorithm almost completely eliminate the inference from the co-runners while the standard FR-FCFS shows up to 20% higher interference even after partitioning DRAM banks.

Our future work include 1) predictable handling of write request draining (currently we only focused on the read request queue) in the DRAM controller; 2) OS extension to support more flexible bank partitioning; and 3) desinginig an interface between the OS and the DRAM controller.

References

[1] B. Akesson, K. Goossens, and M. Ringhofer. Predator: a predictable SDRAM memory controller. In *Hardware/software codesign and system synthesis (CODES+ISSS)*. ACM, 2007.
[2] B. Akesson, L. Steffens, E. Strooisma, and K. Goossens. Real-time scheduling using credit-controlled static-priority arbitration. In *Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2008.
[3] N. Binkert, B. Beckmann, G. Black, S. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. Hower, T. Krishna, S. Sardashti, et al. The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 39(2):1–7, 2011.
[4] A. Hansson, N. Agarwal, A. Kolli, T. Wenisch, and A. Udipi. Simulating DRAM controllers for future system architecture exploration. 2014.

[5] J.L. Hennessy and D.A. Patterson. *Computer architecture: a quantitative approach*. Morgan Kaufmann, 2011.

[6] JEDEC. DDR3 SDRAM Standard JESD79-3F, 2012.

[7] Y. Krishnapillai, Z. Wu, and R. Pellizzoni. ROC: A Rank-switching, Open-row DRAM Controller for Time-predictable Systems. In *Euromicro Conference on Real-Time Systems (ECRTS)*, 2014.

[8] L. Liu, Z. Cui, M. Xing, Y. Bao, M. Chen, and C. Wu. A software memory partition approach for eliminating bank-level interference in multicore systems. In *Parallel Architecture and Compilation Techniques (PACT)*, pages 367–376. ACM, 2012.

[9] M. Paolieri, E. Quiñones, J. Cazorla, and M. Valero. An analyzable memory controller for hard real-time CMPs. *Embedded Systems Letters, IEEE*, 1(4):86–90, 2009.

[10] J. Reineke, I. Liu, H.D. Patel, S. Kim, and E.A. Lee. PRET DRAM controller: Bank privatization for predictability and temporal isolation. In *Hardware/software codesign and system synthesis (CODES+ISSS)*. ACM, 2011.

[11] S. Rixner, W. J Dally, U. J Kapasi, P. Mattson, and J. Owens. Memory access scheduling. In *ACM SIGARCH Computer Architecture News*, volume 28, pages 128–138. ACM, 2000.

[12] Z. Wu, Y. Krish, and R. Pellizzoni. Worst Case Analysis of DRAM Latency in Multi-Requestor Systems. In *Real-Time Systems Symposium (RTSS)*, 2013.

[13] H. Yun, R. Mancuso, Z. Wu, and R. Pellizzoni. PALLOC: DRAM Bank-Aware Memory Allocator for Performance Isolation on Multicore Platforms. In *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2014.