DRAMController: A Simulation Framework for Real-Time DRAM Controllers

Danlu Guo and Rodolfo Pellizzoni
University of Waterloo
Email: dlguo, rpellizz@uwaterloo.ca

I. Introduction

The performance of modern computer systems is increasingly limited by the characteristics of DRAM main memory. Simulation-based study has been widely accepted for computer architecture design, and accurately modeling the memory system is essential to provide meaningful full-system results. Memory performance strongly depends on multiple factors including the device timing constraints, the controller architecture, the workload of incoming requests and the scheduling algorithm of corresponding DRAM access commands.

Available DRAM system simulators, such as DRAMSim2 [2] and Ramulator [1], are designed to model the structure and behavior of DRAM devices, but the underlying DRAM memory controller (MC) which processes incoming memory requests is a fairly simplified model lacking modularity and extensibility. The Gem5 full-system simulator also provides a detailed MC model [3], but it is not cycle-accurate. Recently, the real-time community has proposed many innovative DRAM scheduling policies that allow the derivation of better upper bounds on the latency of memory requests. However, the fixed MC models used in the aforementioned DRAM simulators cannot easily support the architecture required by the proposed real-time policies. As a consequence, authors in the real-time community have typically resorted to designing their own MC simulators from scratch. This increases the time required to test and validate new ideas and complicates the process of comparing different designs.

To address such issues, we present DRAMController, an extensible and cycle-accurate object-oriented simulation framework that simplifies the process of testing and comparing new MC designs. DRAMController is based on the observation that while different MCs employ widely varying scheduling schemes, they process memory requests by a set of common functions that are used to implement standard hardware blocks and processing flow. These functions tend to contribute the majority of the code in any simulator, and can thus be reused across designs. We proved the usability of the simulator by successfully implementing several recently proposed real-time MC designs [4], [5], [6], [7], [8], [11], [10], [9], [12] within our framework. DRAMController can be built on any system supporting C++11; the simulator code is available at [http://ece.uwaterloo.ca/~rpellizz/DRAMController].

II. Simulator Overview

DRAMController employs a modular design; Figure 1 illustrates the major hardware blocks implemented in the framework. Each block is constructed independently and the encapsulated data is accessed through a simple interface. In this manner, changes to the behavior of a specific block do not impact the other blocks in the system. Our generalized architecture consists of an address translator, which maps requests to physical memory cells, a command generator which converts requests into access commands, and a request and command schedulers which determine the order of request/command execution. The specific algorithms implemented by these blocks must be customized based on the MC design; DRAMController exploits the benefits of inheritance and polymorphism by providing virtual function interfaces, which minimize the amount of code required to extend the functionality of each block. A MC simulator must also include queues to connect these hardware blocks, and buffer requests and commands. Rather than fixing the structure of the queues as in most other MC simulators, DRAMController provides an easy to configure, modular queue structure. Since DRAM devices are organized in hierarchy levels (e.g., channels, ranks, bank groups, banks), the configurable queue structure allows the designer to construct the queues according to any DRAM level, and furthermore allocate each queue globally or for specific requestors and request types. As a result, after implementing all aforementioned real-time MCs, we observe that the amount of controller-specific code in the framework is only 10%, and the largest amount of code required to implement any one controller is 200 lines.

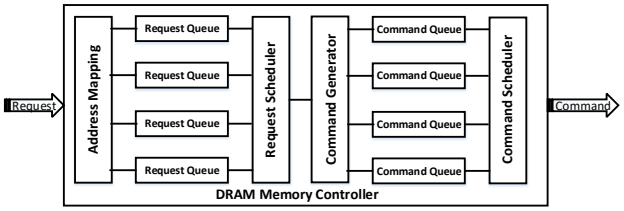

Fig. 1. Generalized Real-Time Memory Controller Architecture.

DRAMController employs a generalized interface that can be accessed by any external source to send memory requests. In particular, we tested the framework using memory request traces generated from full-system simulator. DRAMController also employs an abstract DRAM interface for the DRAM device model, so that the framework is not tied to any specific memory device type. We currently connect to Ramulator as the preferred device simulator, as it supports a wide variety of DRAM standards.

References

[1] Y. K., and W. Y., and O. M., *Ramulator: A Fast and Extensible DRAM Simulator*, CAL, IEEE 2015.
[2] P. R., and E.C. B., and B. J., *DRAMSim2: A Cycle Accurate Memory System Simulator*, CAL, IEEE 2011.
[3] A. H. et al., *Simulating DRAM controllers for future system architecture exploration*, ISPASS, 2014.
[4] M. P., and E. Q., and F. C., and M. V., *An analyzable memory controller for hard real-time CMPs*, ESL, IEEE 2009.
[5] M. H., and H. P., and R. P., *A Framework for Scheduling DRAM Memory Accesses for Multi-core Mixed-time Critical System*, RTAS, 2015.
[6] Y. L., and B. A., and K. G., *Dynamic Command Scheduling for Real-Time Memory Controllers*, ECRTS, 2014.
[7] Z. W., and Y. K., and R. P., *Worst Case Analysis of DRAM Latency in Multi-Requestor Systems*, RTSS, 2013.
[8] L. E., and R. E., *Improved DRAM Timing Bounds for Real-Time DRAM Controllers with Read/Write Bundling*, RTSS, 2015.
[9] H. K., and E. L., and M. Z., *A Predictable and Command-Level Priority-Based DRAM Controller for Mixed-Criticality Systems*, RTAS, 2015.
[10] L. E., and R. E., *Improved DRAM Timing Bounds for Real-Time DRAM Controllers with Read/Write Bundling*, RTSS, 2015.
[11] Y. K., and Z. W., and R. P., *ROC: A Rank Switching, Open-Row DRAM Controller for Time-Predictable Systems*, ECRTS, 2014.
[12] L. E. et al., *A Mixed Critical Memory Controller Using Bank Privatization and Fixed Priority Scheduling*, RTCSA, 2014.