CramSim: Controller and Memory Simulator

Michael B. Healy
IBM T.J. Watson Research Center
Yorktown Heights, NY 10598
mbhealy@us.ibm.com

Seokin Hong
IBM T.J. Watson Research Center
Yorktown Heights, NY 10598
seokin.hong@ibm.com

ABSTRACT

The explosion of digital data and the high computation demands of data analysis have made the memory system a major contributor to the performance and power consumption of modern computing systems. Both industry and academia have proposed innovations such as new memory architectures, interfaces, devices, and topologies, that has lead to a vast increase in the design space of memory systems. In this paper, we present CramSim, which is a flexible, extensible, and scalable simulation framework designed to help efficiently explore the vast design space of memory systems. CramSim is designed on top of SST (Structural Simulation Toolkit) to decouple basic functional blocks of the memory system into separate components. This modular design eases modeling of individual components and evaluation of various configurations of memory systems.

CCS CONCEPTS

•Hardware → Semiconductor memory; •Computing methodologies → Modeling and simulation;

KEYWORDS

Memory, DRAM, Simulator

1 INTRODUCTION

For decades, DRAM (Dynamic Random Access Memory) has been the sole storage cell of computer main memory. Advances in manufacturing technology have improved the capacity and power consumption of DRAM. Also, DRAM vendors have continuously increased DRAM bandwidth by using performance enhancing technologies such as DDR (Double Data Rate) I/O signaling and by increasing I/O frequency. However, DRAM technology faces scaling challenges in density, power, and bandwidth due to fundamental limitations of the physical cell design. Meanwhile, memory-intensive applications are increasingly requiring more memory capacity and bandwidth.

To overcome these challenges, both industry and academia are exploring new memory systems that utilize emerging memory technologies such as 3D-stacked memory (HMC and HBM) and Non-Volatile Memory (FLASH, PCRAM, and STT-MRAM). Hybrid or multi-level memory systems that combine emerging technologies with conventional DRAM have been proposed to meet the

memory capacity, bandwidth, and power consumption requirements of future computing systems. Multi-level memory systems obviously have a larger design space to consider, including the organization, controller, interface, and memory device mix. The community needs a simulation infrastructure that is flexible, extensible, and scalable to study and explore this design space.

Several DRAM simulators have been developed to evaluate current and new DRAM standards. DRAMSim2 [8], the most popular simulator, provides a cycle-accurate DRAM model and a simple programming interface, but it only supports the DDR2/3 standards and a simple memory controller model. The USIMM simulator [1] provides multiple memory controller models, but it only supports the DDR3 standard. Recently, an extensible DRAM simulator, called Ramulator [5], was developed. Even though it supports a variety of DRAM standards (e.g., DDR3/4, LPDDR3/4, GDDR5, and 3D-stacked DRAM) through its strong extensibility, it only offers a simple memory controller model. DRAMsys [4] also supports a wide range of standards and has a flexible controller architecture. However, it is not easy to integrate DRAMsys into other open-source full-system simulators because this simulator is implemented in SystemC TLM. Most full-system simulators, such as Gem5, have their own memory system model [3]. Even if these models are fast and provide reasonable accuracy, they are not portable to other full-system simulators. Also, none of the existing memory simulators are designed to simulate a large-scale memory system comprising a large number of controller and memory models.

In this paper, we present a new simulation infrastructure called CramSim (ContRoller And Memory SIMulator). CramSim is developed on top of Sandia's SST (Structural Simulation Toolkit) framework [7] to assist in the design space exploration of memory systems, in particular for hybrid or multi-level memory systems. Using SST's modular interface, CramSim decouples the primary functional blocks of the memory system into multiple component objects and enables configuration of the memory system by initializing and connecting the components with a Python file. CramSim can run with a request trace file in standalone mode, or it can run with other SST hardware components such as the processor core, on-chip cache, crossbar, and network router. We have also demonstrated CramSim working with the popular gem5 system simulator [2].

2 CRAMSIM ARCHITECTURE

CramSim is designed to run with the SST framework to achieve its three design goals: flexibility, extensibility, and scalability.

Flexibility. CramSim provides plenty of flexibility in configuring the memory system to facilitate efficient exploration of the memory system design space. CramSim consists of one or more independent lanes which can have different configurations such as the number of channels, memory device type, scheduling policy,

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

MEMSYS 2017, October 2–5, 2017, Alexandria, VA, USA

© 2017 Copyright held by the owner/author(s). 978-1-4503-5335-9/17/10...\$15.00
DOI: 10.1145/3132402.3132408

Figure 1: CramSim Architecture

and address mapping as shown in Figure 1. In each lane, there are two components: controller and memory. The controller component models a general memory controller architecture while the memory component models the physical characteristics of the corresponding memory device. A memory system is configured by initiating components with the relevant parameters and by defining connections between them with a Python script.

Extensibility. CramSim decouples key functional blocks of the memory system into separate components. Since each component is isolated from one another, we can implement each component separately, which enables easy extension of CramSim to support new memory standards, controller architectures, and memory devices.

Scalability. The SST framework supports parallel simulation with MPI (Message Passing Interface) to speed up simulation of large-scale systems. CramSim is thus able to simulate large-scale memory systems where there are a large number of memory controllers and device models.

In the following subsections, we describe each component of CramSim in more detail.

2.1 Transaction Dispatcher

The transaction dispatcher receives memory transactions from one or more processor cores or I/O devices. It arbitrates the transactions and allocates them to each lane depending on the specified allocation policy.

2.2 Controller

The memory controller manages data movement between memory devices and processor cores with five essential functions: prioritizing memory requests, converting memory requests to device commands, mapping physical addresses to device addresses, scheduling device commands, and issuing commands while meeting memory access protocol requirements. In CramSim, these functions are implemented in pluggable sub-component objects inside the controller component so that we can easily build and evaluate various memory controller models that use different management policies and algorithms.

Address Mapper The major role of the address mapper is to convert the physical address of memory requests into device addresses. For DRAM devices, the physical address is mapped to a set of channel, rank, bank, row, and column indexes. Since the address mapping is a significant factor in determining system performance, CramSim provides a highly flexible address mapper that can specify the mapping policy at the bit granularity.

Transaction Scheduler The transaction scheduler orders the candidate transactions (memory requests) for each channel for dispatching in a way that maximizes performance, fairness and maintains QoS (Quality of Service). CramSim currently includes two transaction scheduler models having different transaction queue structures: one has a unified queue while another has split read and write queues. Both transaction scheduler models offer FCFS (First-Come-First-Serve) and FR-FCFS (First Ready, First-Come-First-Serve) [6] scheduling policies.

Transaction Converter The transaction converter converts a transaction into memory commands and then stores the commands in a command queue. For example, a read request is converted to “ACTIVATE” and “READ” commands for DRAM devices. It can also be set up to generate “PRECHARGE” commands to model row-buffer management policies. The current CramSim provides a transaction converter model supporting close-page, open-page, and pseudo-open-page row-buffer policies for DRAM devices.

Command Scheduler The command scheduler picks memory commands from command queue in a way that maximize the bank-, rank-, or channel-level parallelism while enforcing the timing constraints of memory devices. CramSim currently provides two command scheduling policies: bank round-robin and rank round-robin. The bank round-robin policy searches for issuable commands in the per-bank queues for a given rank, and then moves to the next rank. The rank round-robin policy searches through the per-bank queues of all ranks for a given bank.

Device driver The device driver maintains state information for the memory devices and provides that information to the command scheduler. This state information enables the scheduler to identify commands that can be dispatched to the memory device in any cycle. The device driver also handles device-specific control mechanisms. For example, it can be configured to periodically generate “REFRESH” commands to model the refresh operation of DRAM devices.

2.3 Memory

The memory component models the physical characteristics of memory devices and their internal functionalities that are hidden from the memory controller. It collects internal events of memory devices such as row activations, row buffer hits, bank precharges, etc. The collected data is used to calculate power consumption of memory devices along with their power models. CramSim currently provides a memory component model for DRAM (DDR3/4 and HBM). The component includes an equation-based power model similar to that used in DRAMSim2 and USIMM. The memory component also tracks completion of commands and sends responses to the controller component. The memory component can be configured to store data on write requests and to provide

Table 1: Simulated Throughput (Transactions/cycle)

Simulator	DDR3 (1ch)		DDR4 (1ch)		HBM (1ch)		HBM (8ch)	
	rand	seq	rand	seq	rand	seq	rand	seq
CramSim	0.13	0.22	0.14	0.17	0.18	0.44	1.41	3.13
Ramulator	0.13	0.24	0.13	0.19	0.17	0.46	1.34	3.68

Table 2: Runtime (sec.) with DDR4 for random trace

Simulator	1 ch	2 ch	4 ch	8 ch
CramSim	7875	8921	10708	13153
Ramulator	4024	7866	15550	30789

that data on read requests to work with full-system simulators that require memory storage.

2.4 Configuration

CramSim configures the memory system organization, memory controller, and device types with a Python file. The timing and power parameters of each component can be directly specified within the Python configuration file, included in a separate configuration file, or provided in the command line. Currently, CramSim provides configuration files for DRAM devices (i.e., DDR3/4 and HBM).

3 VALIDATION AND EVALUATION

To validate the basic functionality of CramSim, we developed an equation-based analytical framework called VeriMem. The VeriMem suite is a set of simple read and write request streams that we can easily reason about the expected performance of the memory system. Each stream includes requests that hop among various structures within the DRAM: rows, columns, banks, etc. with varying frequencies. Each stream is thus impacted by a discreet and known set of timing parameters. The VeriMem suite enabled us to work out many bugs in the early development stages of CramSim, which now passes all tests. The VeriMem suite is also used regularly to ensure that new code has not introduced any timing bugs. In addition, we compared the simulation results of CramSim with that of IBM’s internal simulation model of the DRAM systems for complex request traces.

We also compared CramSim with Ramulator [5], which is one of the state-of-the-art memory simulators, to evaluate its simulation accuracy and scalability. Table 1 shows the simulated transaction throughput of three DRAM standards (DDR3, DDR4, and HBM) for 100M random and sequential traces. As we can see in the table, two simulators estimate almost same throughput for each DRAM standard. To evaluate the scalability, we increase the number of channels and linearly increase the number of the per-cycle transaction according to the number of channels. In the case of CramSim, we configured the multi-channel memory system to have multiple lanes that simulate each channel, and then we run CramSim in the parallel simulation mode with MPI. As shown in Table 2, CramSim consumes only 67% more time when simulating an 8-channel

configuration with heavy traffic than when simulating a 1-channel configuration.

4 CONCLUSION

In this paper, we have introduced CramSim, a flexible, extensible and scalable simulation framework for memory systems. Thanks to its modular design, each component of the memory systems can be modeled separately and is easily pluggable into the simulation framework. CramSim enables efficient exploration of the significant design space of future memory systems.

REFERENCES

- [1] U ARCH, N Chatterjee, and R Balasubramanian. Usimm: the utah simulated memory module. *University of Utah Tech. Rep.*, 2012.
- [2] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D Hill, and David A Wood. The Gem5 Simulator. *ACM SIGARCH Computer Architecture News*, 39(2):1–7, August 2011.
- [3] Andreas Hansson, Neha Agarwal, Aasheesh Kolli, Thomas Wensich, and Anirudha N. Udupi. Simulating DRAM controllers for future system architecture exploration. In *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 201–210. IEEE, 2014.
- [4] Matthias Jung, Christian Weis, and Norbert Wehn. DRAMSys: A Flexible DRAM Subsystem Design Space Exploration Framework. *IPSS Transactions on System LSI Design Methodology*, 8:63–74, 2015.
- [5] Y Kim, W Yang, and O Mutlu. Ramulator: A fast and extensible DRAM simulator. *IEEE Computer Architecture Letters*, 15(1):45–49, 2016.
- [6] Scott Rixner, William J Dally, Ujval J Kapasi, Peter Mattson, and John D Owens. Memory Access Scheduling. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pages 128–138, New York, NY, USA, 2000. ACM.
- [7] A F Rodrigues, K S Hemmert, B W Barrett, C Kersey, R Oldfield, M Weston, R Risen, J Cook, P Rosenfeld, E CooperBalls, and B Jacob. The structural simulation toolkit. *ACM SIGMETRICS Performance Evaluation Review*, 38(4):37–42, March 2011.
- [8] P Rosenfeld, E Cooper-Balis, and B Jacob. DRAMSim2: A Cycle Accurate Memory System Simulator. *IEEE Computer Architecture Letters*, 10(1):16–19, 2011.