Simulation based comparative study of open system memory simulators that model DRAM and upcoming NVM.

Vansh Bajaj, *California State University Long Beach,*
and Professor Xiaolong Wu, *California State University Long Beach*

Abstract—In this paper I present a comparative analysis of a number of open source memory simulators that model a number of DDR standards and hybrid DRAM + NVM . The goal of this project is to implement these simulators on a testbench in both full system and trace based simulations . I then go on to compare these simulators on the basis of experimental evaluation of performance metrics such as simulated clock cycles , Request throughput , runtime , memory and power consumption as well as qualitative metrics such as ease of integration , supported standards of simulation , degree of flexibility , ease of use , extensibility etc. Finally , I perform a comparative evaluation of all the available simulators to determine which of these is the most optimal memory simulators for researchers developing new standards.

Index Terms—DRAM,DRAMSim2,Ramulator,NVMain,Gem5,cycle-accurate,DRAMPower,Standalone,Integrated.

1 Introduction

In the recent years we have witnessed an explosion of new proposals for DRAM interfaces of different architectures , level of details and features . Hence simulation has become a vital tool for researchers and developers in the computer architecture community and at the forefront of these innovations should be memory simulators which are used to evaluate the strengths and weaknesses of each new proposal . Unfortunately, many CPU simulators overlook the need for accurate models of the memory system as they tend to include overly simplistic models of memory which in turn fails to take into account the highly complex nature of modern memory systems . A typical DDR memory controller reorders and schedules requests multiple times while keeping track of dozens of timing parameters. Despite the fact that main memory is growing both in complexity and as a system-level bottleneck, 'cycle accurate' CPU simulators often attach a fixed latency to memory accesses, significantly under-reporting the real effect of the memory system. This effective lack of attention on memory system simulators is itself an impediment to industrial development and academic research In this paper I plan to test some of the publicly available memory system simulators on a number of simulated characteristics as well as qualitative characteristics . The structure of paper is as follows – The first section will describe each of the memory simulators used in this paper . The second section will deal with system simulation validation and evaluation . Finally the third section will focus on the comparative analysis and the conclusion .

Segment	DRAM Standards and Architecture
Commodity	DDR3(2007), DDR4(2012)
Low-Power	LPDDR3(2012),LPDDR4(2014)
Graphics	GDDR5(2009)
Performance	eDRAM,RLDRAM3(2011)
3D –Stacked	WIO(2011),WIO2(2014),MCDRAM(2015),HBM(2013),HMC1.0(2013),HMC1.1(2014)
Non Volatile Memory	PCRAM,STT-RAM,MRAM

TABLE 1: DRAM Segment and Architectures

1.1 Background

Modern computer system performance is increasingly limited by the performance of DRAM-based memory systems. More complex algorithms, coupled with larger data sets, lead to a sharp growth in computational requirements in domains ranging from embedded systems to servers. As a result, there is great interest in providing accurate simulations of DRAM based memory systems as part of architectural research. Unfortunately, there is great difficulty associated with the study of modern DRAM memory systems, arising from the fact that DRAM-system performance depends on many independent variables such as workload characteristics of memory access rate and request sequence, memory-system architecture, memory-system configuration, DRAM access protocol, and DRAM device timing parameters. DRAM based memory systems are impacted primarily by two attributes: row cycle time and device datarate. Presently, DRAM row cycle times are decreasing at a rate of approximately

Simulators	DRAM Standards
DRAMSim2	DDR2,DDR3
Ramulator	DDR3,DDR4,LPDDR3,LPDDR4,GDDR5,WIO1,WIO2,HBM
NVMain	DDR3,LPDDR3,LPDDR4
Gem5	DDR3,LPDDR3,WIO

TABLE 2: DRAM Simulators and supported standards

2 RAMULATOR

Ramulator [1], a fast and cycle-accurate DRAM simulator that is built from the ground up for extensibility Ramulator is based on a generalized template for modeling a DRAM system, which is later infused with the specific details of a DRAM standard . Due to this modular design Ramulator is able to provide out-of-the-box support for a wide array of DRAM standards: DDR3/4, LPDDR3/4, GDDR5, WIO1/2, HBM. Ramulator is based on a fundamental principle that is -

DRAM can be abstracted as a hierarchy of state-machines, where the behavior of each state-machine — as well as the aforementioned hierarchy itself — is dictated by the DRAM standard in question.[1]

2.1 Architecture

Ramulator decouples the logic for querying and updating the state-machines from the implementation specifics of any particular DRAM standard. RAmulator's architecture is built around a collection of lookup tables which are computaionally inexpensive to update and query. This and the extremely modular design are the reasons Ramulator outperforms all the other simulators by a large factor.

Memory Controllers in Ramulators maintain three queues of memory requests . They are - *read, write,* and *maintenance.* . The read/write queues are populated by demand memory requests (*read, write*) which is generated by an external source whereas the maintenance queue is populated by other types of memory requests (*refresh, powerdown, selfrefresh*) generated internally by the memory controller as they are needed.

Each DRAM class is represented by the following template-

```
 // DRAM.h
 template <typename T>
 class DRAM {
DRAM <T>* parent;
 vector <DRAM <T>*>
 children;
 T::Level level;
 int index;
// more code...
};
```

2.2 Simulation

Since Ramulator provides a generalized skeleton of DRAM , it is capable of being infused with the specifics of any supported standard . To demonstrate this idea I describe how DDR4 support was added to Ramulator:

1) Copy *DDR3.h/cpp* to *DDR4.h/cpp*.
2) Add *BankGroup* as an item in *DDR4.h/cpp*.
3) Edit 20 entries in the lookup-tables — 1 in prerequisite, 2 in transition,and 17 in timing.

According to Ramulator the difference between any two DRAM standards is simply the difference in their lookup tables.

3 DRAMSIM2

DramSim2 [3] is a cycle accurate DDR2/DDR3 simulator. It is implemented in C++ as an object oriented model of a memory system that includes a detailed model of the memory controller that issues commands to a set of DRAM devices attached to a standard memory bus .

The DRAMSim2 core is wrapped in a single object called MemorySystem. A memory system object requires two ini files: a device ini file and a system ini file. A device ini file contains parameters that describe a specific DRAM device such as the timing constraints and power consumption of the device. These parameters can be found on manufacturer data sheets which are available on their websites. The DRAMSim2 package contains several device ini files for Micron DDR2/3 devices of varying densities and speed grades. The system ini file consists of parameters that are independent of the actual DRAM device. These include parameters such as the number of ranks, the address mapping scheme, debug options, row buffer policies, memory controller queue structures, and other simulation details.

3.1 Simulation

DRAMSim2 can be compiled either as a standalone binary or as a shared library. In standalone mode, DRAMSim2 simulates commands read from a memory trace on disk. In the shared library mode, DRAMSim2 exposes the basic functionality to create a MemorySystem object and add requests to it. DRAMSim2 relies on no external libraries; thus, it is easy to build on any platform that has the GNU C++ compiler installed.

DRAMSim2 attempts to model a modern DDR2/3 memory controller in a general way. Requests from the CPU are buffered into a transaction queue in execution order. These transactions are converted into DRAM commands and placed into a command queue which can have different structures such as per rank or per rank per bank. The memory controller maintains the state of every memory bank in the system and uses this information to decide which request should be issued next. The memory controller is free to issue requests from the command queue out of order as long as it doesn't schedule writes ahead of dependent reads or violate timing constraints.

In open page mode, DRAMSim2 will keep rows open and prioritize requests to already open rows. This can decrease the overhead of precharging rows that might potentially be used in the near future. Alternatively, closed page mode will precharge rows immediately after each request; this can be beneficial for workloads with poor locality. DRAMSim2 can be configured to store data on writes and return data on reads. Many CPU simulators and other front-end drivers are not concerned with the actual data, so data storage is disabled by default to reduce the memory usage of the simulator. As reads and writes complete, the simulator keeps track of the bandwidth and latency of the requests. These statistics are averaged over an epoch, the length of which is configurable by the user. During each epoch, the simulator outputs detailed bandwidth, latency, and power statistics both to a .vis file and the console . DRAMSim2 memory controller also models the effects of DRAM refresh. Modeling refresh is important since refresh has become a major source of variance in the latency of memory requests.

4 NVMain

NVMain [2] is a cycle accurate memory simulator for modeling not only commodity DRAMs but also emerging memory technologies, such as die-stacked DRAM caches, non-volatile memories (e.g., STT-RAM, PCRAM, and ReRAM) including multi-level cells (MLC), , and hybrid non-volatile plus DRAM memory systems.

4.1 Architecture

In NVMain 2.0, a bank is no longer the most basic memory object. Instead, sub-array are defined as the basic blocks to support various sub-array-level parallelism (SALP) [6] modes seamlessly. The sub-array is selected similar to other memory objects using the address translator. This object also contains the MLC write, endurance, and fault models. NVMain 2.0 also allows for fine grained refresh, including all-bank refresh as in DDR, per-bank refresh as in LPDDR, or bank-group refresh as in DDR4.

A distributed timing model is used whereby all timings related to a specific memory object are tracked by that object itself. For example, the Sub-Array tracks the most commonly found memory timing parameters (e.g., tRCD, tCAS, tRP), while the rank objects keep track of rank timing parameters (e.g., tRTRS). All timing parameters are considered before any memory commands can be issued and the worst case value is taken. Therefore, the distributed timing model is functionally equivalent to other simulators using monolithic memory controllers which track all timing values.

4.2 Simulation

Prior NVM simulations have been done by simply reusing existing DRAM simulators with NVM timing parameters. Nonetheless, such simulation is unable to capture the unique features in NVMs, which desires correct endurance modeling, fault recovery, and MLC operation with high accuracy in terms of energy and latency. DRAM simulators typically ignore data being written since timing and energy parameters are agnostic to these values. In contrast, the innovations on endurance improvements, fault recovery mechanisms, and MLC are commonly studied in NVM systems. Although this work is not designed for comparing endurance mechanisms, many of these techniques employ some form of data encoding, which changes the data value ultimately written to the memory cells.

5 GEM5

The gem5 [4] simulator was the result of a merger of the best aspects of m5 and gems simulators. M5 provides a highly configurable simulation framework, multiple ISAs, and diverse CPU models. GEMS complements these features with a detailed and flexible memory system, including support for multiple cache coherence protocols and interconnect models. Many academic and industrial institutions such as Princeton, MIT,ARM,HP and others combined their efforts to create GEM5.

5.1 Architecture

Flexibility is built in the architecture of gem5 . Flexibility is achieved through object oriented design . All major simulation components in the gem5 simulator are SimObjects . They share common behaviors for configuration , initialization , statistics and checkpointing . SimObjects include models of concrete hardware components such as processor cores, caches, interconnect elements and devices, as well as more abstract entities such as a workload and its associated process context for system-call emulation. Every SimObject is represented by two classes, one in Python and one in C++ which derive from SimObject base classes present in each language. The Python class definition specifies the SimObject's parameters and is used in script-based configuration. The C++ class encompasses the SimObject's state and remaining behavior, including the performance-critical simulation model.However 85 percent of gem5 is written in C++. Gem5 also employs Domain Specific Languages (DSL) which provide a powerful and concise way to express solutions by leveraging knowledge from common space. Mainly two DSL's are employed by the gem5 –

- ISA DSL – Inherited from M5 , This language unifies the decoding of binary instructions and the specification of their semantics. The CPU models use a C++ common base class to describe instructions . The ISA DSL allows users to specify C++ code compactly. It also allows for the specification of class templates which are a more generalized that C++ templates. They cover broad categories of instructions such as register-to-register arithmetic operations . ISA also provides for the specification of a decode tree that combines opcode decoding with the creation of specific derived classes as instances of previously defined templates.
- Cache Coherence DSL – Gem5 employs SLICC to implement a variety of cache coherence protocols . SLICC defines the cache , memory and DMA controllers as individual per-memory-block state machines that form the overall protocol . SLICC allows different protocols to incorporate the sme underlying state transition mechanisms with minimal programmer ef-

fort.
The gem5 SLICC defines protocols as a set of states , events transitions and actions within each transition specify the operations that must be performed . The SLICC is implemented in python in the gem5 architecture. , all configuration parameters are specified as input parameters and gem5 SLICC automatically generates the appropriate C++ and Python files.

6 DRAMPower

DRAMPower[5] is an open source tool for DRAM power and energy estimation for DDR2/DDR3/DDR4, LPDDR/LPDDR2/LPDDR3 and Wide IO DRAM memories. Even though DRAMPower is an energy estimation tool and not a memory simulator . It was used briefly with Ramulator to observe the power consumption of the simulator. The DRAMPower uses two types of traces depending upon the type of interface used-

- Command Traces Command Traces can be logged in a file placed in the traces folder in the subdirectory.
  The format it uses is : *<timestamp>*,*<command>*,*<bank>*.
  Timestamp is in clock cycles , supported commands are showed in the *MemCommand.h* , bank is the target bank number. For non bank specific commands bank is set to 0.
- Transaction Traces
  If the transaction-level interface is being used , a transaction trace can be logged. The format it uses is - *<timestamp>*,*<transaction type>*,*<address>* For example , a transaction trace looks like – '10,WRITE,0x50089. Timestamp is in clock cycles (cc) and maximum row-bank-column-BI-BC-BGI-BL Here, BI gives the degree of bank interleaving, BC gives the burst size (count), BGI gives the degree of bank group interleaving (for DDR4) and BL gives the burst length used by the device.

7 Experimentation and Evaluation

The simulators was performed on a testbench with the following specifications-

- Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz, 2808 Mhz, 4 Core(s), 8 Logical Processor(s)
- BIOS Version/Date American Megatrends Inc. F.10, 11/1/2017
- Embedded Controller Version 40.24
- BIOS Mode UEFI
- Installed Physical Memory (RAM) 8.00 GB SODIMM
- Available Virtual Memory 7.18 GB

Each of the simulators were executed on the Ubuntu 18.04.1 LTS system OS.

As simulators for memory controllers and DRAM systems , each of these simulators must be supplied with a stream of memory requests from external memory sources. Each of the simulators provides two modes of operations

- Standalone mode – Memory trace is fed manually by the user.
- Integrated mode – Memory traces are fed from an execution driven engine that is the gem5.

In the following section I present the results from operating all the above mentioned simulators in standalone mode , validate their correctness and compare then of quantitative metrics which are – Simulated Clock Cycles , Runtime , Req/sec(Throughput),Memory Consumption(MB) and Average Power Consumption. Each of the simulators must simulate any given stream of memory request using a legal sequence of DRAM commands, honoring the status transitions and the timing parameters of any supportable standards. A synthetic trace file was created for each of the simulators . The read and writes are in the ratio of 9 : 1 and minority requests which are refreshes , power-down and self refreshes.

7.1 Results

Each of the simulators were build and executed in the same environment with no background applications running to get as accurate observations as possible. Tabulated results are presented on the next page.

Simulators	Standards	Simulated Clock Cycles	Runtime(sec)	Throughput(Req/sec)	Memory Consumption (MB)	Average Power Consumption
Ramulator+DRAMpower	DDR3	783894347	787 sec	57521522	2.5	58.30 mW
	DDR4	559048774	856	57257286	2.46	57.55 mW
	LPDDR2	798168856	568	51734212	8	41.78 mW
	LPDDR3	888903015	849	25156530	1.2	43.25 mW
	GDDR5	294319976	743	57413079	0.5	64.67 mW
DRAMSim2	DDR3	714882177	1982	52000	5.8	1.183 W
	DDR2	714882177	1982	52000	5.8	1.183 W
NVMain 2.0	Hybrid DRAM + NVM	757400272	4528	15480	3864	2.52 W
Gem5	DDR3	727554122	21052	8564	4862	1.95 W

TABLE 3: Comparison of simulators

Some of the inferences from the comparison are –

- All four simulators are yield almost the same number of simulated clock cycles with slight discrepancies in which standard is being implemented and the difference in how their memory controllers make scheduling descions .
- Ramulator has the shortest runtime taking only 787 sec for the DDR3 standard. It is approximately 2X faster than the next fastest simulator – DramSim2 which takes 1982 sec.
- Ramulator has the highest throughput of all the simulators due to shortest runtime. The least memory consumption was observed in the GDDR5 standard implemented in Ramulator.
- Ramulator also had the least Power Consumption by a large factor as compared to other simulators.
- Ramulator also has the most extensibility that the rest of the simulators on the market , supporting the most amount of DRAM standards.

8 Conclusion

In this paper I compared the top open source DRAM simulators available in the market on the basis of Qualitative and Quantitative metrics. However , all of these simulators don't have the support or the community needed to extensively develop these simulators and make them more efficient and cycle-accurate. Simulators are at the forefront of developing new technologies . DRAM simulators should be given the same support as some of the full system and architecture simulators since new DRAM standards are being actively proposed. Experimental standards should be provided an active environment to test them easily and compare with existing and other new standards.

At the moment Ramulator has proven to be the most effective simulator among the most popular ones.It has a clear advantage in speed , accuracy efficiency and extensibility as well as its comprehensive list of supportable DRAM standards.

References

[1] K. Yoongu,Y. Weikun and M. Onur *Ramulator: A Fast and Extensible DRAM Simulator*, Carnie Mellon University , Peking University , 2016.

[2] P. Matt, Z. Tao, X. Yuan *NVMain 2.0: A User-friendly Memory Simulator to Model (Non-)Volatile Memory Systems*, The Pennsylvania State University,2015.

[3] R. Paul, B. Elliot, J. Bruce, *DRAMSim2: A Cycle Accurate Memory System Simulator*, University of Maryland College Park, 2011.

[4] N. Binkert, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. Hill, D. Wood, B. Beckmann, G. Black, S. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. Hower, T. Krishna, *The Gem5 Simulator*, 2011.

[5] K. Chandrasekar, C. Weis, Y. Li, S. Goossens, M. Jung, O. Naji, B. Akesson, W. Norbert,G. Kees *DRAMPower: Open-source DRAM power and energy estimation tool*, 2011.