

Introduction to Ramulator 2.1 and Memory Systems Research

Haocong Luo

ETH Zurich

3rd R&DB Workshop and Tutorial @ ICS'26

06.07.2026

Ramulator

A Fast and Extensible DRAM Simulator
[IEEE Computer Architecture Letters'15]

Motivation

- DRAM and Memory Controller landscape is changing
- Many new and upcoming standards
- Many new controller designs
- A **fast** and **easy-to-extend** simulator is very much needed

<i>Segment</i>	<i>DRAM Standards & Architectures</i>
Commodity	DDR3 (2007) [14]; DDR4 (2012) [18]
Low-Power	LPDDR3 (2012) [17]; LPDDR4 (2014) [20]
Graphics	GDDR5 (2009) [15]
Performance	eDRAM [28], [32]; RLD RAM3 (2011) [29]
3D-Stacked	WIO (2011) [16]; WIO2 (2014) [21]; MCDRAM (2015) [13]; HBM (2013) [19]; HMC1.0 (2013) [10]; HMC1.1 (2014) [11]
Academic	SBA/SSA (2010) [38]; Staged Reads (2012) [8]; RAIDR (2012) [27]; SALP (2012) [24]; TL-DRAM (2013) [26]; RowClone (2013) [37]; Half-DRAM (2014) [39]; Row-Buffer Decoupling (2014) [33]; SARP (2014) [6]; AL-DRAM (2015) [25]

Table 1. Landscape of DRAM-based memory

Ramulator: A Fast and Extensible DRAM Simulator

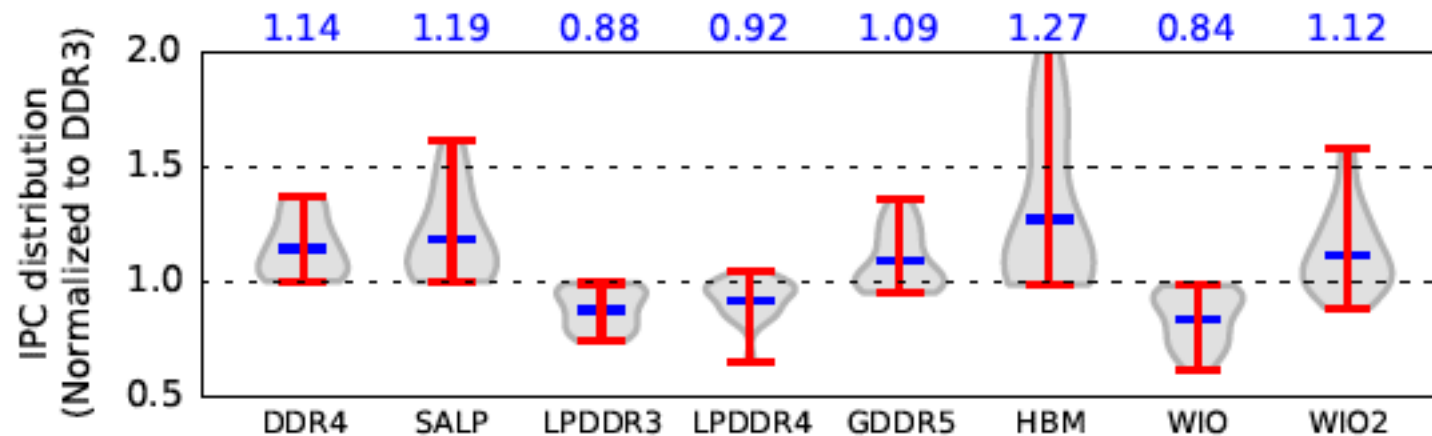
- Modular and extensible to different standards
- Provides out-of-the box support for many DRAM standards:
 - DDR3/4, LPDDR3/4, GDDR5, WIO1/2, HBM, PCM, STT-RAM
 - new proposals (SALP, AL-DRAM, TLD RAM, RowClone, and SARP)
- ~2.5X faster than fastest open-source simulator

<i>Simulator</i> (clang -O3)	<i>Cycles</i> (10^6)		<i>Runtime</i> (sec.)		<i>Req/sec</i> (10^3)		<i>Memory</i> (MB)
	<i>Random</i>	<i>Stream</i>	<i>Random</i>	<i>Stream</i>	<i>Random</i>	<i>Stream</i>	
Ramulator	652	411	752	249	133	402	2.1
DRAMSim2	645	413	2,030	876	49	114	1.2
USIMM	661	409	1,880	750	53	133	4.5
DrSim	647	406	18,109	12,984	6	8	1.6
NVMain	666	413	6,881	5,023	15	20	4,230.0

Table 3. Comparison of five simulators using two traces

Case Study: Comparison of DRAM Standards

<i>Standard</i>	<i>Rate (MT/s)</i>	<i>Timing (CL-RCD-RP)</i>	<i>Data-Bus (Width×Chan.)</i>	<i>Rank-per-Chan</i>	<i>BW (GB/s)</i>
DDR3	1,600	11-11-11	64-bit × 1	1	11.9
DDR4	2,400	16-16-16	64-bit × 1	1	17.9
SALP [†]	1,600	11-11-11	64-bit × 1	1	11.9
LPDDR3	1,600	12-15-15	64-bit × 1	1	11.9
LPDDR4	2,400	22-22-22	32-bit × 2*	1	17.9
GDDR5 [12]	6,000	18-18-18	64-bit × 1	1	44.7
HBM	1,000	7-7-7	128-bit × 8*	1	119.2
WIO	266	7-7-7	128-bit × 4*	1	15.9
WIO2	1,066	9-10-10	128-bit × 8*	1	127.2



Across 22 workloads, simple CPU model

Figure 2. Performance comparison of DRAM standards

Ramulator Paper and Source Code

- Yoongu Kim, Weikun Yang, and Onur Mutlu,
"Ramulator: A Fast and Extensible DRAM Simulator"
IEEE Computer Architecture Letters (**CAL**), March 2015.
[[Source Code](#)]
- Source code is released under the liberal MIT License
 - <https://github.com/CMU-SAFARI/ramulator>

Ramulator: A Fast and Extensible DRAM Simulator

Yoongu Kim¹ Weikun Yang^{1,2} Onur Mutlu¹
¹Carnegie Mellon University ²Peking University

Ramulator: Free & Open Source

☰ README.md

Ramulator: A DRAM Simulator

Ramulator is a fast and cycle-accurate DRAM simulator [1, 2] that supports a wide array of commercial, as well as academic, DRAM standards:

- DDR3 (2007), DDR4 (2012)
- LPDDR3 (2012), LPDDR4 (2014)
- GDDR5 (2009)
- WIO (2011), WIO2 (2014)
- HBM (2013)
- SALP [3]
- TL-DRAM [4]
- RowClone [5]
- DSARP [6]

The initial release of Ramulator is described in the following paper:

Y. Kim, W. Yang, O. Mutlu. "[Ramulator: A Fast and Extensible DRAM Simulator](#)". In *IEEE Computer Architecture Letters*, March 2015.

For information on new features, along with an extensive memory characterization using Ramulator, please read:

S. Ghose, T. Li, N. Hajinazar, D. Senol Cali, O. Mutlu. "[Demystifying Complex Workload-DRAM Interactions: An Experimental Study](#)". In *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, June 2019 ([slides](#)). In *Proceedings of the ACM on Measurement and Analysis of Computing Systems (POMACS)*, 2019.

[1] Kim et al. *Ramulator: A Fast and Extensible DRAM Simulator*. IEEE CAL 2015.

[2] Ghose et al. *Demystifying Complex Workload-DRAM Interactions: An Experimental Study*. SIGMETRICS 2019.

[3] Kim et al. *A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM*. ISCA 2012.

[4] Lee et al. *Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture*. HPCA 2013.

[5] Seshadri et al. *RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization*. MICRO 2013.

[6] Chang et al. *Improving DRAM Performance by Parallelizing Refreshes with Accesses*. HPCA 2014.

Usage

Ramulator supports three different usage modes.

1. **Memory Trace Driven:** Ramulator directly reads memory traces from a file, and simulates only the DRAM subsystem. Each line in the trace file represents a memory request, with the hexadecimal address followed by 'R' or 'W' for read or write.

- 0x12345680 R
- 0x4cbd56c0 W

• ...

Ramulator: Reproducibility

Reproducing Results from Paper (Kim et al. [1])

Debugging & Verification (Section 4.1)

For debugging and verification purposes, Ramulator can print the trace of every DRAM command it issues along with their address and timing information. To do so, please turn on the `print_cmd_trace` variable in the configuration file.

Comparison Against Other Simulators (Section 4.2)

For comparing Ramulator against other DRAM simulators, we provide a script that automates the process: `test_ddr3.py`. Before you run this script, however, you must specify the location of their executables and configuration files at designated lines in the script's source code:

- Ramulator
- DRAMSim2 (<https://wiki.umd.edu/DRAMSim2>): `test_ddr3.py` lines 39-40
- USIMM (<http://www.cs.utah.edu/~rajeew/jwac12>): `test_ddr3.py` lines 54-55
- DrSim (<http://ph.ece.utexas.edu/public/Main/DrSim>): `test_ddr3.py` lines 66-67
- NVMain (<http://wiki.nvmain.org>): `test_ddr3.py` lines 78-79

Please refer to their respective websites to download, build, and set-up the other simulators. The simulators must be executed in saturation mode (always filling up the request queues when possible).

All five simulators were configured using the same parameters:

- DDR3-1600K (11-11-11), 1 Channel, 1 Rank, 2Gb x8 chips
- FR-FCFS Scheduling
- Open-Row Policy
- 32/32 Entry Read/Write Queues
- High/Low Watermarks for Write Queue: 28/16

Finally, execute `test_ddr3.py <num-requests>` to start off the simulation. Please make sure that there are no other active processes during simulation to yield accurate measurements of memory usage and CPU time.

Cross-Sectional Study of DRAM Standards (Section 4.3)

Please use the CPU traces (SPEC 2006) provided in the `cputraces` folder to run CPU trace driven simulations.

Other Tips

Power Estimation

For estimating power consumption, Ramulator can record the trace of every DRAM command it issues to a file in DRAMPower [8] format. To do so, please turn on the `record_cmd_trace` variable in the configuration file. The resulting DRAM command trace (e.g., `cmd-trace-chan-N-rank-M.cmdtrace`) should be fed into a compatible DRAM energy simulator such as VAMPIRE [8] or DRAMPower [9] with the correct configuration (standard/speed/organization) to estimate energy/power usage for a single rank (a current limitation of both VAMPIRE and DRAMPower).

Ramulator: Integration with Other Simulators

☰ README.md

Usage

Ramulator supports three different usage modes.

1. **Memory Trace Driven:** Ramulator directly reads memory traces from a file, and simulates only the DRAM subsystem. Each line in the trace file represents a memory request, with the hexadecimal address followed by 'R' or 'W' for read or write.

- 0x12345680 R
- 0x4cbd56c0 W
- ...

2. **CPU Trace Driven:** Ramulator directly reads instruction traces from a file, and simulates a simplified model of a "core" that generates memory requests to the DRAM subsystem. Each line in the trace file represents a memory request, and can have one of the following two formats.

- `<num-cpuinst> <addr-read>` : For a line with two tokens, the first token represents the number of CPU (i.e., non-memory) instructions before the memory request, and the second token is the decimal address of a *read*.
- `<num-cpuinst> <addr-read> <addr-writeback>` : For a line with three tokens, the third token is the decimal address of the *writeback* request, which is the dirty cache-line eviction caused by the read request before it.

3. **gem5 Driven:** Ramulator runs as part of a full-system simulator (gem5 [7]), from which it receives memory request as they are generated.

For some of the DRAM standards, Ramulator is also capable of reporting power consumption by relying on either VAMPIRE [8] or DRAMPower [9] as the backend.

[7] The gem5 Simulator System.

[8] Ghose et al. *What Your DRAM Power Models Are Not Telling You: Lessons from a Detailed Experimental Study*. SIGMETRICS 2018.

[9] Chandrasekar et al. *DRAMPower: Open-Source DRAM Power & Energy Estimation Tool*. IEEE CAL 2015.

Getting Started

Ramulator requires a C++11 compiler (e.g., `clang++`, `g++-5`).

1. **Memory Trace Driven**

```
$ cd ramulator
$ make -j
$ ./ramulator configs/DDR3-config.cfg --mode=dram dram.trace
Simulation done. Statistics written to DDR3.stats
# NOTE: dram.trace is a very short trace file provided only as an example.
$ ./ramulator configs/DDR3-config.cfg --mode=dram --stats my_output.txt dram.trace
Simulation done. Statistics written to my_output.txt
# NOTE: optional --stats flag changes the statistics output filename
```

2. **CPU Trace Driven**

Ramulator Enabled Many Works

Journals & Magazines > IEEE Computer Architecture Le... > Volume: 15 Issue: 1 ?

Ramulator: A Fast and Extensible DRAM Simulator

Publisher: IEEE

[Cite This](#)



[Yoongu Kim](#) ; [Weikun Yang](#) ; [Onur Mutlu](#) [All Authors](#)

563

Cites in
Papers

2

Cites in
Patents

6741

Full
Text Views



Abstract

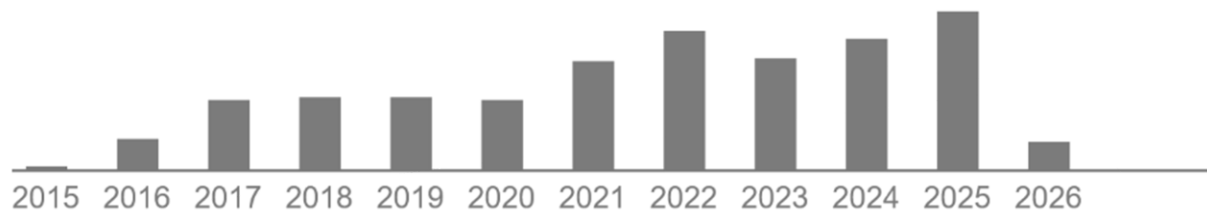
Document Sections

- 1 Introduction
- 2 Ramulator: High-Level Design
- 3 Extensibility of Ramulator
- 4 Validation & Evaluation
- 5 Conclusion

Abstract:

Recently, both industry and academia have proposed many different roadmaps for the future of DRAM. Consequently, there is a growing need for an extensible DRAM simulator, which can be easily modified to judge the merits of today's DRAM standards as well as those of tomorrow. In this paper, we present Ramulator, a fast and cycle-accurate DRAM simulator that is built from the ground up for extensibility. Unlike existing simulators, Ramulator is based on a generalized template for modeling a DRAM system, which is only later infused with the specific details of a DRAM standard. Thanks to such a decoupled and modular design, Ramulator is able to provide out-of-the-box support for a wide array of DRAM standards: DDR3/4, LPDDR3/4, GDDR5,

Total citations Cited by 1019



Ramulator 2.0

A Modern, Modular, and Extensible
DRAM Simulator

[IEEE Computer Architecture Letters'24]

Motivation

- DRAM and Memory Controller landscape is changing
- **Robustness issues** in DRAM
 - Data retention
 - Read disturbance (RowHammer, RowPress, ColumnDisturb etc.)
 - ...
- **Performance issues** in main memory system
 - Processing-in-Memory architectures
 - Emerging memory technologies
 - Performance overhead analysis of read disturbance mitigation techniques

DRAM simulation infrastructures are needed
to understand, characterize, and evaluate
the robustness and performance of DRAM

Executive Summary

Ramulator 2.0: Modern, modular, and extensible
DRAM & memory system simulator

- Models a wide range of **DRAM standards** and **memory controller functionalities**
- **Modular and extensible** software architecture
- **Fine-grained modeling** of DRAM operation (cycle-level)
- Used in a wide range of research works

Implementing Many RowHammer Mitigations in Ramulator 2.0

- Haocong Luo, Yahya Can Tugrul, F. Nisa Bostanci, Ataberk Olgun, A. Giray Yaglikci, and Onur Mutlu,

"Ramulator 2.0: A Modern, Modular, and Extensible DRAM Simulator"

*Preprint on **arxiv**, August 2023.*

[[arXiv version](#)]

[[Ramulator 2.0 Source Code](#)]

Ramulator 2.0: A Modern, Modular, and Extensible DRAM Simulator

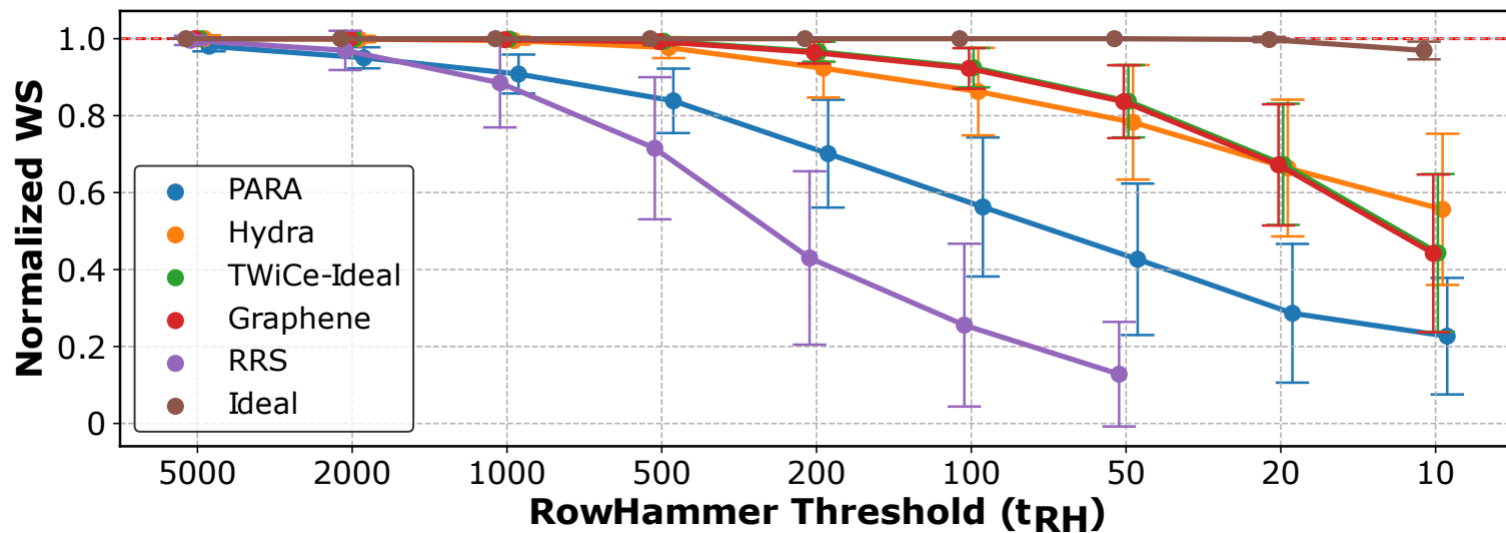
Haocong Luo, Yahya Can Tuğrul, F. Nisa Bostancı, Ataberk Olgun, A. Giray Yağlıkçı, and Onur Mutlu

<https://arxiv.org/pdf/2308.11030.pdf>

<https://github.com/CMU-SAFARI/ramulator2>

Implementing Many RowHammer Mitigations in Ramulator 2.0

- **Cross-section performance overhead evaluation of different RowHammer mitigation techniques**
- Six different RowHammer mitigation techniques all implemented as plugins to the same memory controller implementation



As DRAM becomes more vulnerable to RowHammer, evaluated RowHammer mitigations incur **higher performance overheads**

Evaluating Per-Row Activation Counting (as introduced in DDR5 standard)

- Oğuzhan Canpolat, A. Giray Yağlıkçı, Geraldo F. Oliveira, Ataberk Olgun, Oğuz Ergin, and Onur Mutlu,
"Understanding the Security Benefits and Overheads of Emerging Industry Solutions to DRAM Read Disturbance"
4th Workshop on DRAM Security (DRAMsec), held with **ISCA**,
Buenos Aires, Argentina, July 2024.

Understanding the Security Benefits and Overheads of Emerging Industry Solutions to DRAM Read Disturbance

Oğuzhan Canpolat^{§†}

A. Giray Yağlıkçı[§]

Geraldo F. Oliveira[§]

Ataberk Olgun[§]

Oğuz Ergin[†]

Onur Mutlu[§]

[§]ETH Zürich

[†]TOBB University of Economics and Technology

<https://arxiv.org/pdf/2406.19094>

<https://github.com/CMU-SAFARI/ramulator2>

Addressing Drawbacks of PRAC

- Oguzhan Canpolat, Abdullah Giray Yaglikci, Geraldo Francisco de Oliveira, Ataberk Olgun, Nisa Bostanci, Ismail Emir Yuksel, Haocong Luo, Oguz Ergin, and Onur Mutlu,
"Chronus: Understanding and Securing the Cutting-Edge Industry Solutions to DRAM Read Disturbance"
HPCA, Las Vegas, NV, USA, March 2025.
Officially artifact evaluated as available, functional, and reproduced.

Chronus: Understanding and Securing the Cutting-Edge Industry Solutions to DRAM Read Disturbance

Oğuzhan Canpolat^{§†} A. Giray Yağlıkçı[§] Geraldo F. Oliveira[§] Ataberk Olgun[§]
Nisa Bostancı[§] Ismail Emir Yuksel[§] Haocong Luo[§] Oğuz Ergin^{‡†} Onur Mutlu[§]
[§]ETH Zürich [†]TOBB University of Economics and Technology [‡]University of Sharjah

<https://arxiv.org/pdf/2502.12650>

A Recent Paper using gem5+Ramulator 2.0

MICRO 2025



Understanding and Mitigating Covert Channel and Side Channel Vulnerabilities Introduced by RowHammer Defenses

F. Nisa Bostancı¹ Oğuzhan Canpolat^{1,2} Ataberk Olgun¹ İsmail Emir Yüksel¹
Konstantinos Kanellopoulos¹ Mohammad Sadrosadati¹ A. Giray Yağlıkçı^{1,3} Onur Mutlu¹
¹ETH Zürich ²TOBB ETÜ ³CISPA

DRAM chips are increasingly vulnerable to read disturbance phenomena (e.g., RowHammer and RowPress), where repeatedly accessing or keeping open a DRAM row causes bitflips in nearby rows, due to DRAM density scaling. Attackers can exploit RowHammer bitflips in real systems to compromise security, which has motivated many prior works on RowHammer defenses. To enable such defenses, recent DDR specifications

rows) [1–14]. RowHammer [1, 2, 4, 14] is a type of read disturbance phenomenon, where a victim row can experience bitflips when at least one nearby row is hammered more times than a threshold, called the RowHammer threshold (N_{RH}). Modern DRAM chips become more vulnerable to RowHammer as DRAM technology node size becomes smaller [1, 2, 4, 14–19]. RowPress [13] is another example DRAM read disturbance

<https://arxiv.org/pdf/2503.17891>

<https://github.com/CMU-SAFARI/LeakyHammer>

2025



Cleaning up the Mess

Re-Evaluating the Real-System Modeling Accuracy of Ramulator 2.0

F. Nisa Bostancı*

Haocong Luo*

Ataberk Olgun*

Maria Makeenkova

Geraldo F. de Oliveira

A. Giray Yağlıkçı

Onur Mutlu

* Equal Contribution

SAFARI

ETH zürich

Mess: A MICRO'24 Best Paper Runner-Up

- ***"A Mess of Memory System Benchmarking, Simulation and Application Profiling"***, MICRO'24
 - **A best paper runner-up** with **all 3 artifact badges** (including *"Reproducible"*)
- Proposes **a new benchmark** to evaluate **the real** and **simulated memory system performance**
- Makes **strong negative claims** about two of our simulation infrastructures, Ramulator 2.0 and DAMOV, and shows **unexpected results**
 - Contending that they *"poorly resemble the actual system performance"*

Cleaning up the Mess

We demonstrate that **these claims and results are incorrect** and are due to **configuration** and **simulator usage errors** made in the Mess paper

We describe **four best practices** to aid users and developers of simulation tools

To facilitate transparency and reproducibility, we **open source all our code** and **results**:
<https://github.com/CMU-SAFARI/Cleaning-up-the-Mess>

Cleaning up the Mess

**Analysis of the Evaluation of Ramulator 2.0
in the Mess Paper**

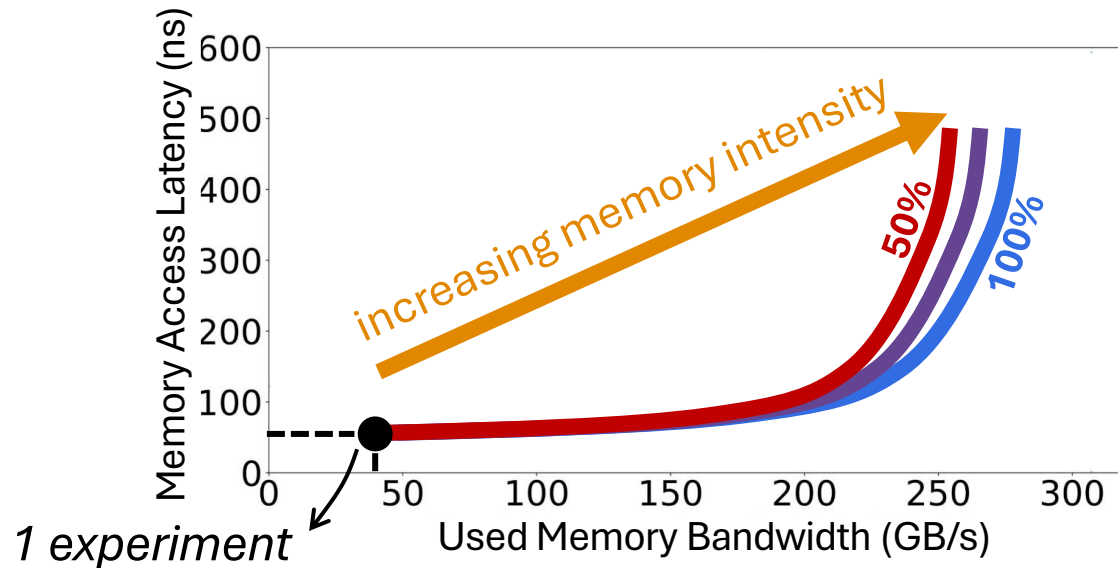
Analysis of the Evaluation of DAMOV
in the Mess Paper

Ramulator 2.0 and DAMOV Best Practices

High-Level Memory System

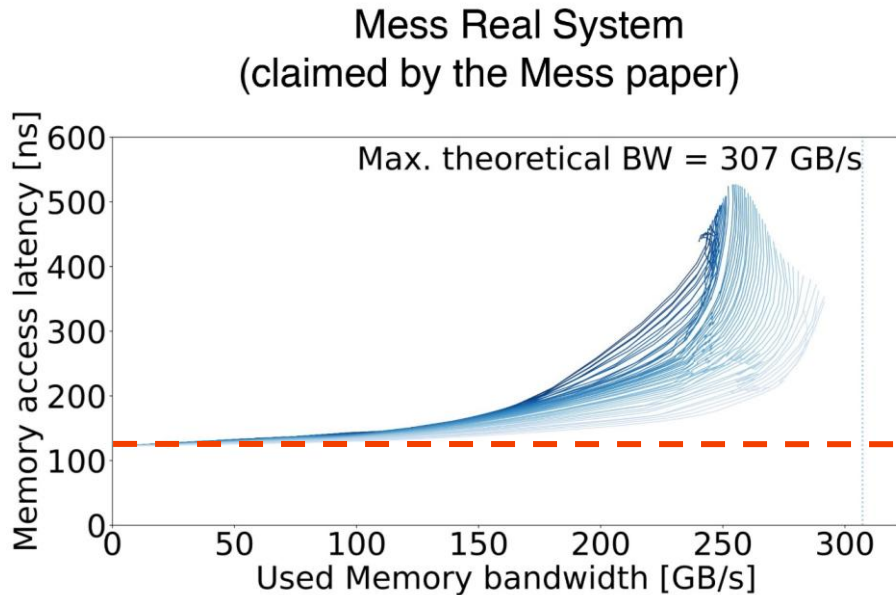
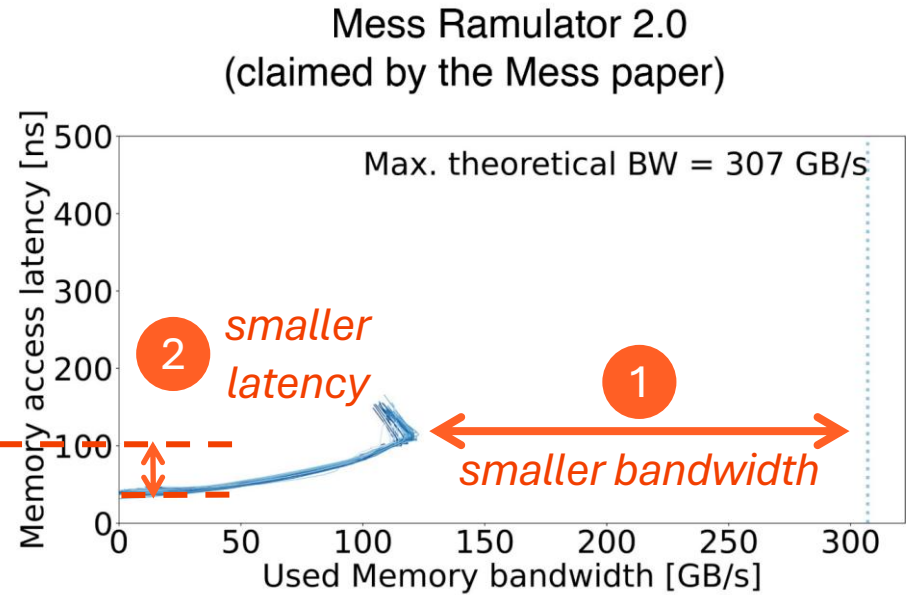
Performance Evaluation & Reasoning

- **The Mess benchmark** aims to characterize a memory system based on a set of *latency-throughput* ("used bandwidth") curves



- Consists of two concurrently running workloads: *Streaming* and *Pointer-Chasing* patterns
 - **Varies the memory intensity** using NOPs between requests
 - Evaluates **a sweep of read/write intensities**
 - (from **50%** to **100%** reads)

The Mess Paper's Evaluation of Ramulator 2.0



- Contending that Ramulator 2.0 *"poorly resembles the actual system performance"*

Ramulator 2.0 simulation results in the Mess paper are **incorrect** and, at the time of the publication of the Mess paper, **irreproducible**

Analysis of the Evaluation of Ramulator 2.0 in the Mess Paper

Reproducibility Issues:

1

It was *not possible* to reproduce *any* Ramulator 2.0 result from the Mess paper artifacts available at the time of publication and artifact evaluation of the paper

The missing source codes were uploaded to the Mess GitHub repository 10 months after the artifact evaluation badges

Fundamental Configuration Errors:

2

- 1) The Mess paper effectively compares an **8-channel** DDR5 memory system in Ramulator 2.0 against a **16-channel** real DDR5 memory system
- 2) The Mess paper uses **unrealistically low cache latency configurations** that lead to **low memory access latency**

Scholarship Issues:

3

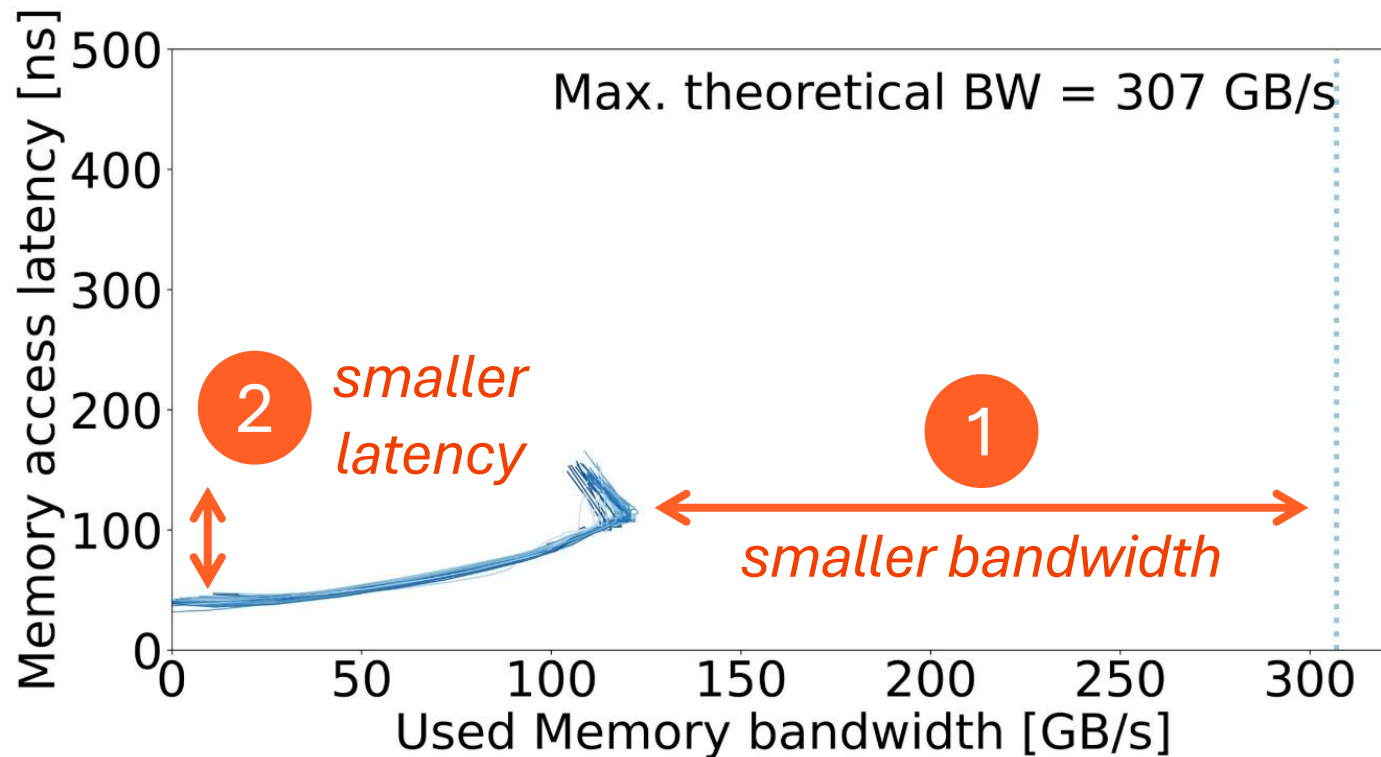
The Mess paper **does not disclose**

- 1) **modifications** to the cache latency configurations, and
- 2) **differences in evaluation methodology** between simulation and real system results (identified via email exchange with Mess paper's authors)

Fundamental Configuration Errors

- **The Mess paper's configuration errors** result in the claimed Ramulator 2.0 inaccuracies

Mess Ramulator 2.0
(claimed by the Mess paper)

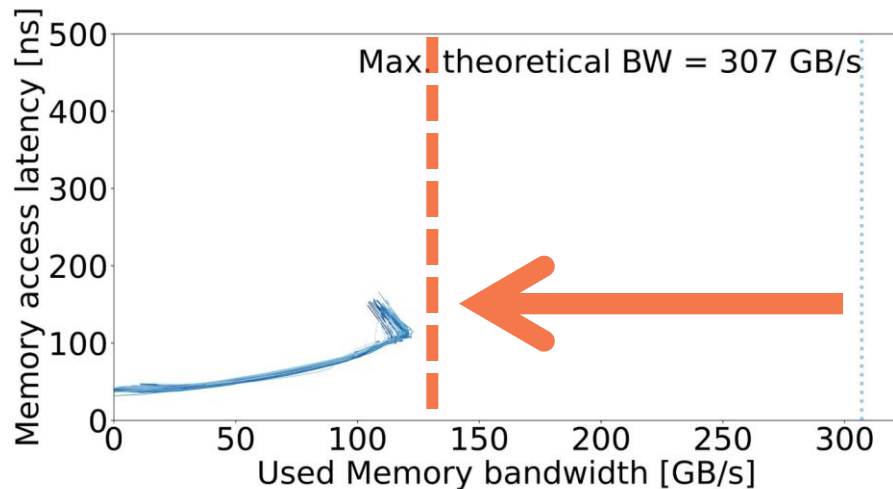


Fundamental Configuration Errors: Smaller Used Bandwidth

Mess Paper's Ramulator 2.0 Configuration

8 DDR5 channels

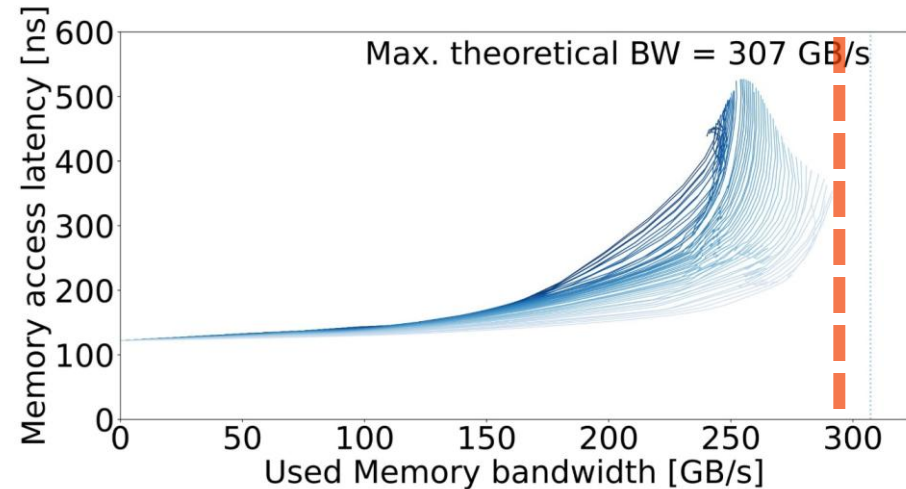
Mess Ramulator 2.0
(claimed by the Mess paper)



Mess Paper's Real System Configuration

16 DDR5 channels

Mess Real System
(claimed by the Mess paper)



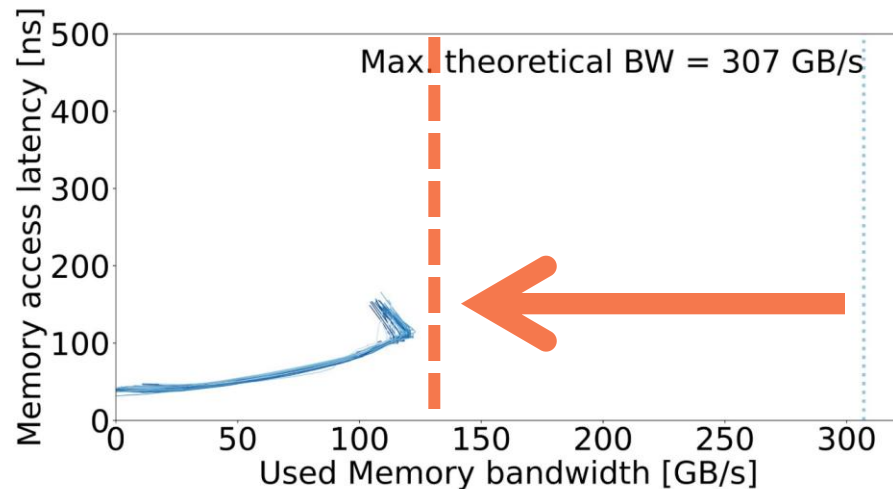
The Mess paper **incorrectly assumes** a DDR5 channel in Ramulator 2.0 is **64-bit**, instead of **the actual width of 32-bit** that is **consistent with JEDEC DDR5 standards** (JESD79-5)

Fundamental Configuration Errors: Smaller Used Bandwidth

Mess Paper's Ramulator 2.0 Configuration

8 DDR5 channels

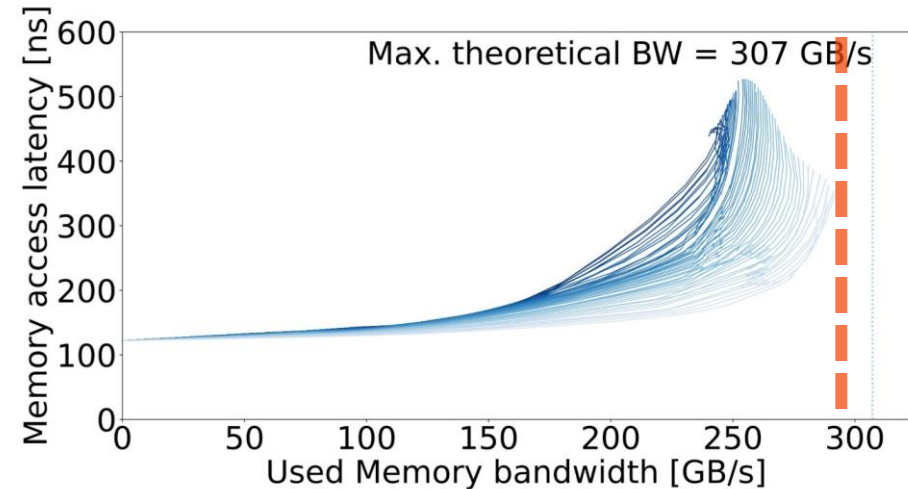
Mess Ramulator 2.0
(claimed by the Mess paper)



Mess Paper's Real System Configuration

16 DDR5 channels

Mess Real System
(claimed by the Mess paper)



The Mess paper configures Ramulator 2.0 with **half the channels** of a real system, and thus, results in **smaller used bandwidth**

Fundamental Configuration Errors: Smaller Memory Access Latency

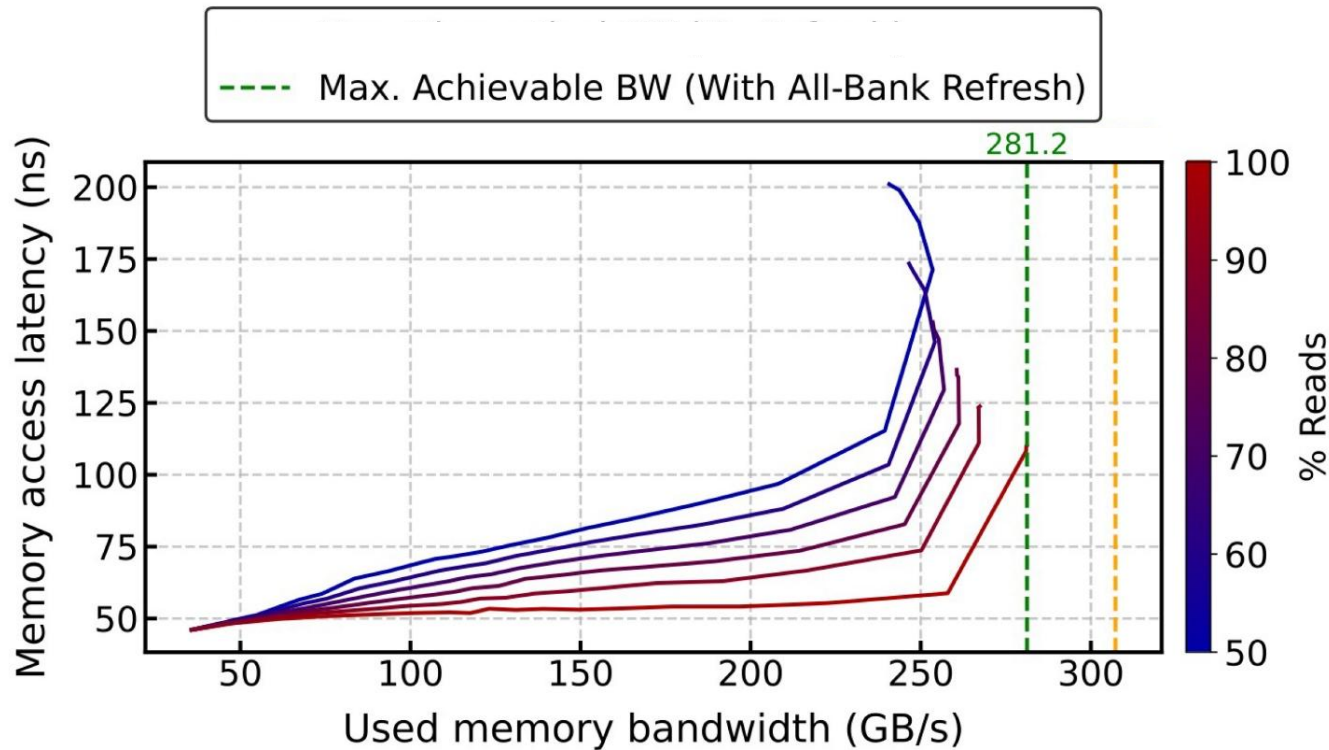
- The Mess paper's artifact hardcodes
 - the cache latency as **0 cycles**
 - the number of miss-status holding registers (MSHRs) per CPU as **1024**
- This leads to significantly smaller DRAM round-trip latency and queueing delays, thus **smaller memory access latency**

The Mess paper uses **unrealistically low cache latency configurations** resulting in the **low memory access latency** that it attributes to Ramulator 2.0

Our Evaluation Methodology

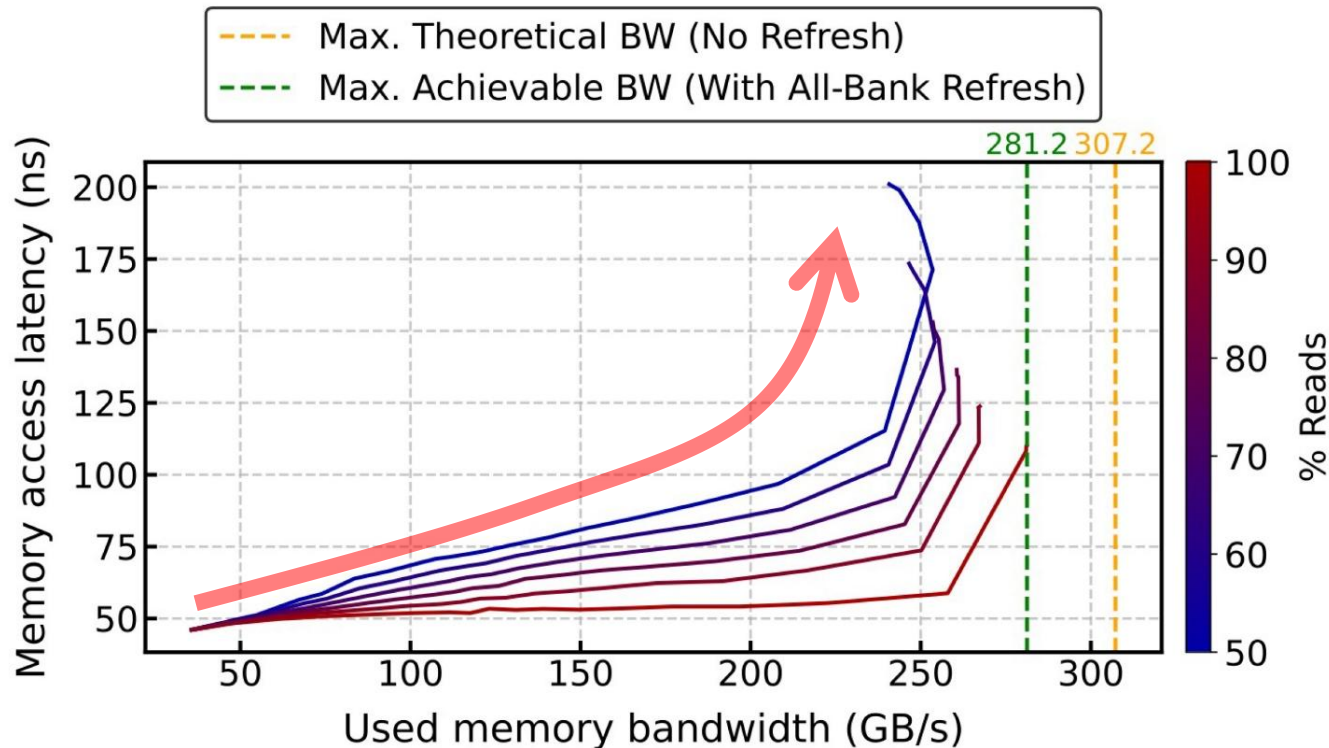
- We **independently evaluate the Mess benchmark** by executing the same access pattern
 - *Streaming* and *Pointer-Chasing* access patterns
 - Sweeping memory intensity
- We **correctly configure Ramulator 2.0**
 - Provide the same number of channels as the real system
 - Evaluate the memory access latency from the memory controller to isolate the DDR5 model's simulation accuracy

Our Key Results



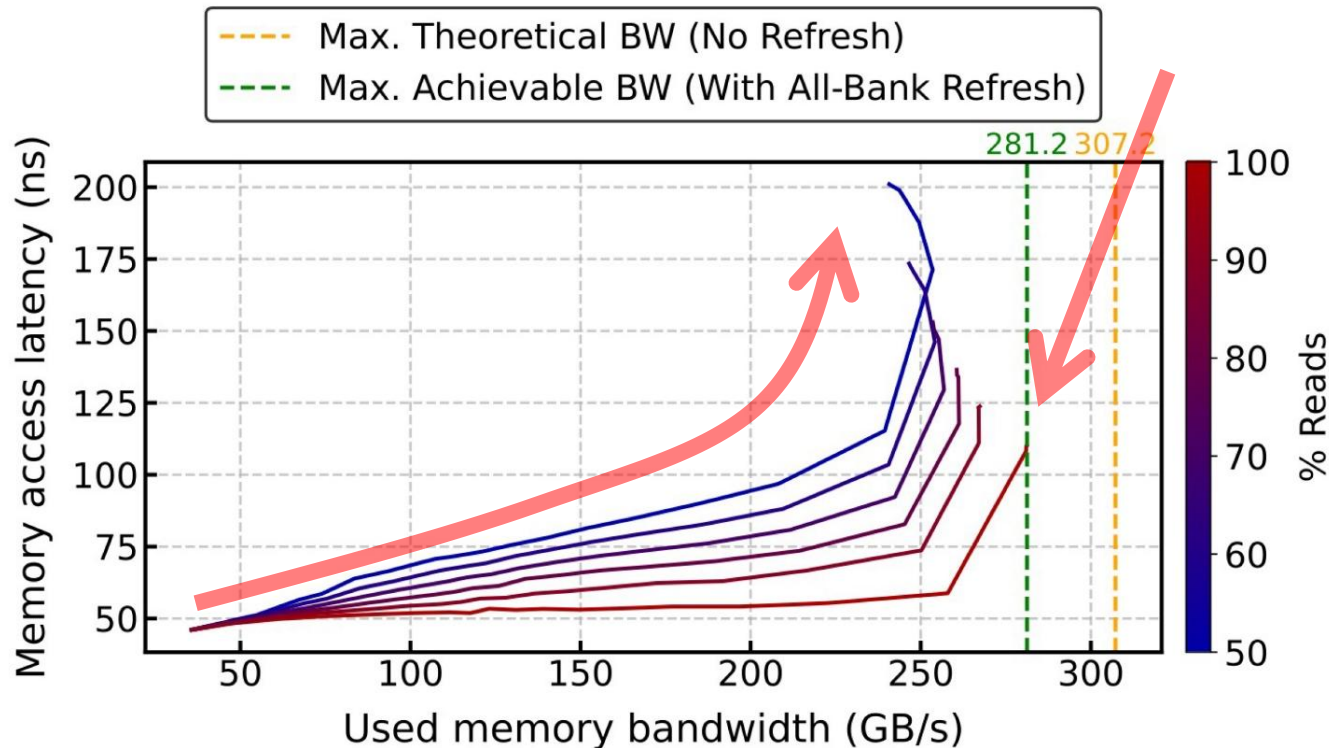
Ramulator 2.0's memory model achieves **high used memory bandwidth**, close to the **achievable bandwidth value**, as opposed to the claims of the Mess paper

Our Key Results



The **memory access latency** of Ramulator 2.0's model **increases** as the **used memory bandwidth** and the **write ratio** increases

Our Key Results



When **correctly configured**, Ramulator 2.0's simulated memory system performance **resembles real system characteristics**

Cleaning up the Mess

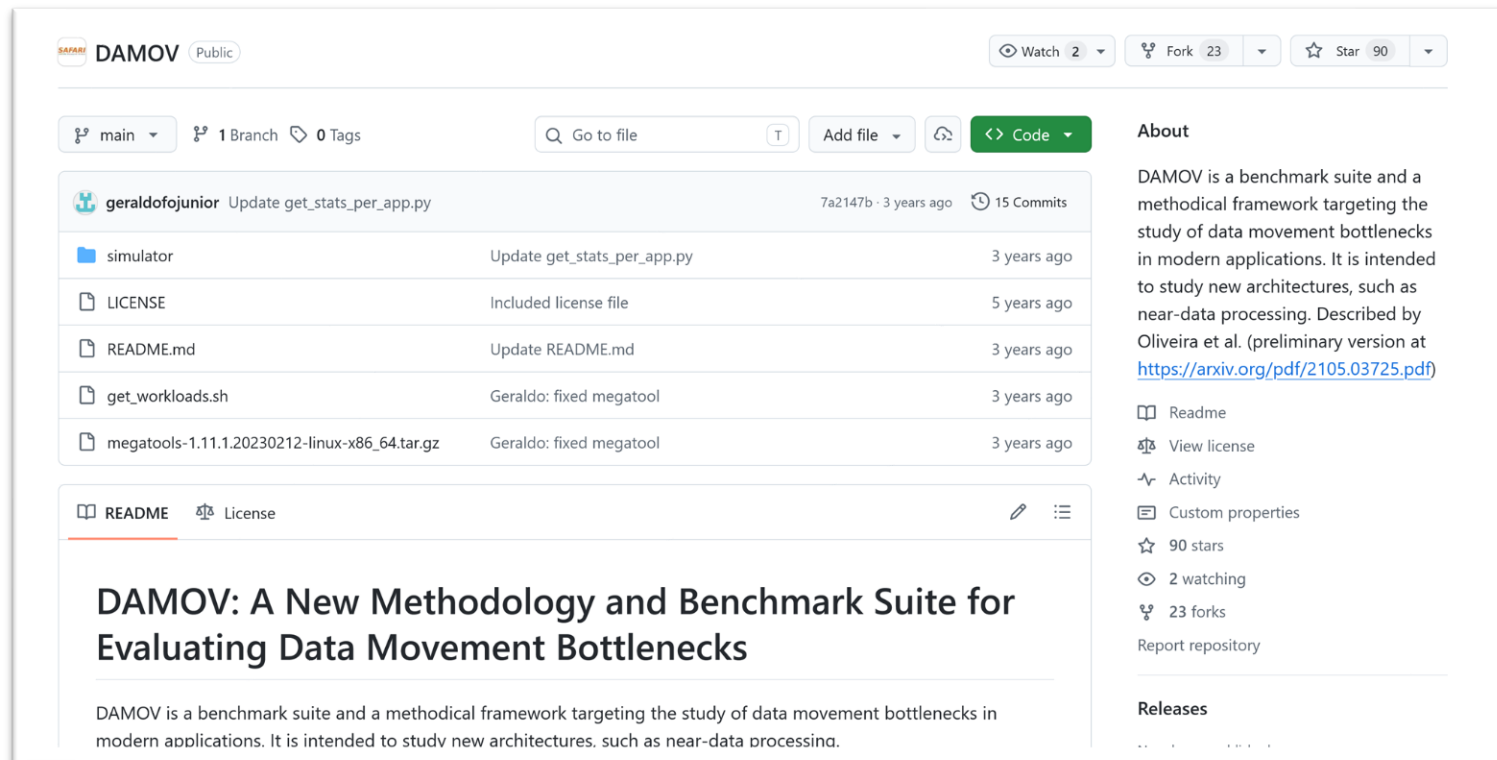
Analysis of the Evaluation of Ramulator 2.0
in the Mess Paper

Analysis of the Evaluation of DAMOV
in the Mess Paper

Ramulator 2.0 and DAMOV Best Practices

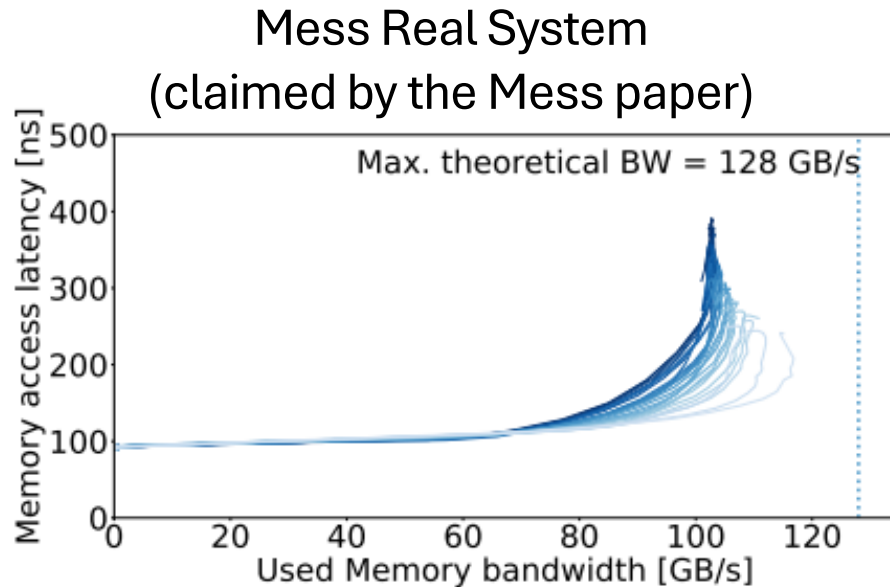
DAMOV: A Simulation Framework for PiM

- DAMOV integrates ZSim [Sanchez+, ISCA'13] with Ramulator [Kim+, CAL'15]
 - Enables evaluating emerging processing-in-memory architectures

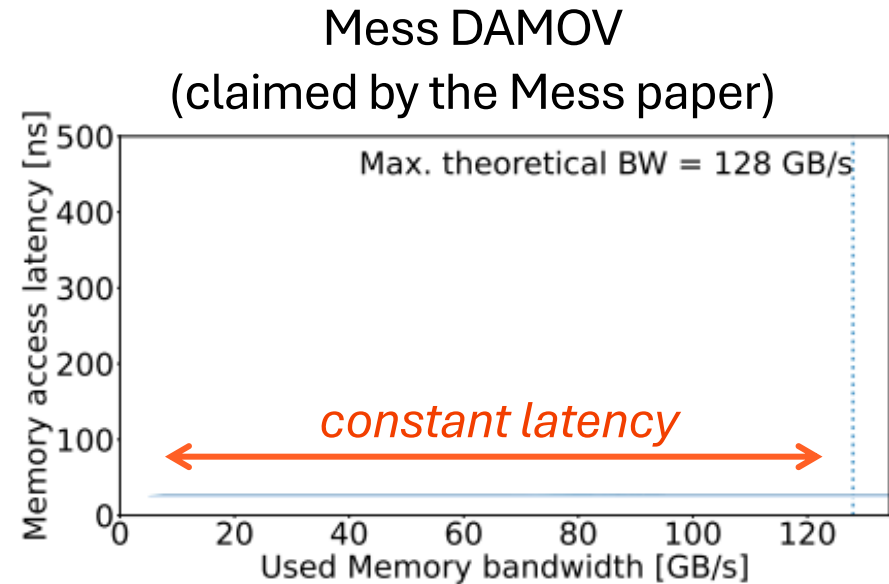


<https://github.com/CMU-SAFARI/DAMOV>

The Mess Paper's Evaluation of DAMOV



(a) Intel Skylake with $6 \times$ DDR4-2666.



(f) ZSim+Ramulator simulator.

DAMOV results in the Mess paper **use incorrect simulation statistics** that are unrelated to the simulated DRAM performance

Analysis of the Evaluation of DAMOV in the Mess Paper

Reproducibility Issues:

1

We could not reproduce any DAMOV results in the Mess paper, using the Mess paper's artifact due to:

- a) **Missing source code and configurations** of Ramulator
- b) Rigid **assertion failure** according to the ZSim configurations

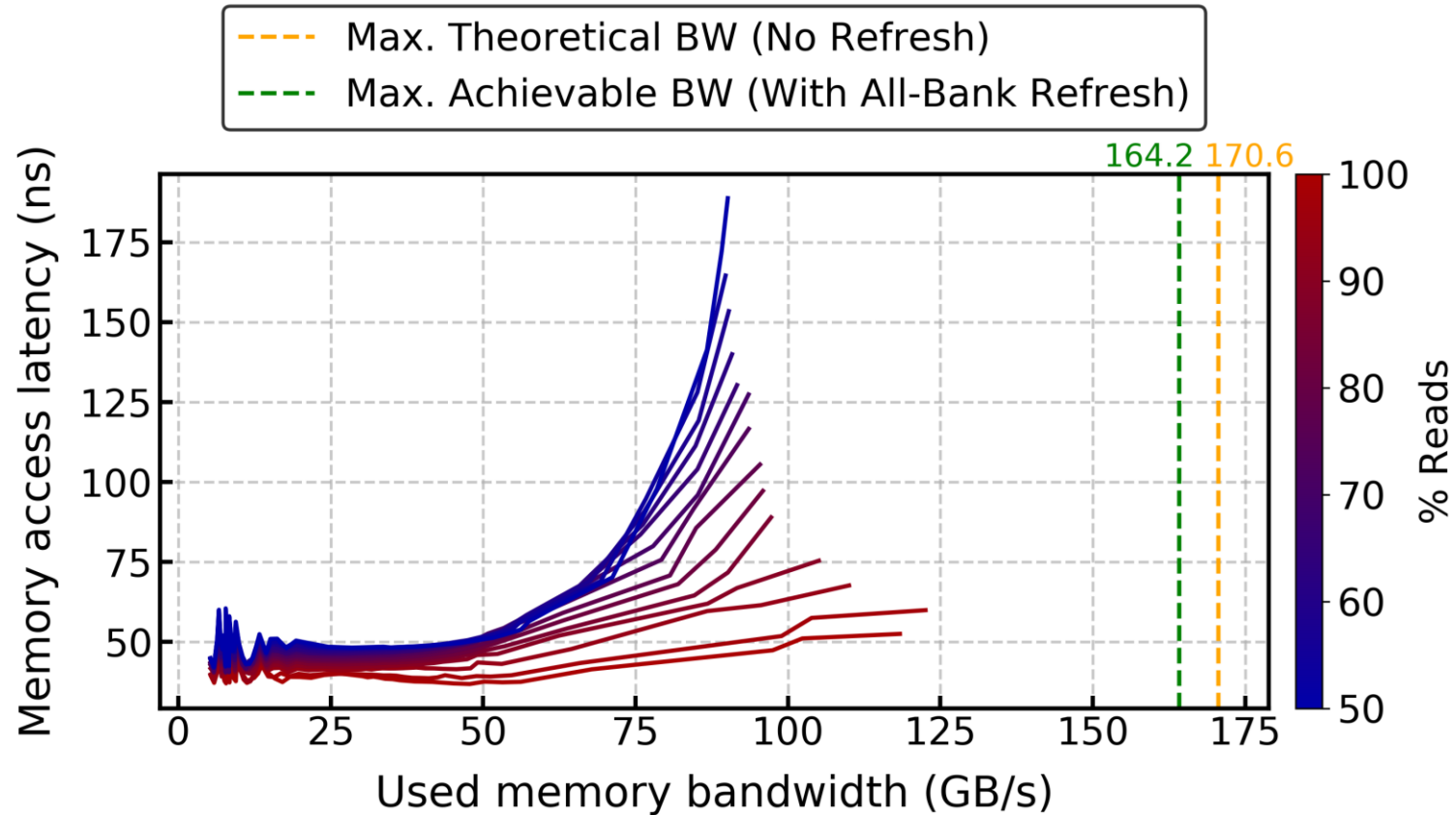
Technical Simulator Usage Errors:

2

- a) Uses two ZSim statistics for DRAM latency: **latGETnl** and **mGETs** from the L1-D cache
- b) These statistics are **not Ramulator statistics** and **does not reflect simulated DRAM latency** from Ramulator

We independently evaluate DAMOV by executing the Mess benchmark and adding missing pieces to the artifact in a best-guess manner and measure the used bandwidth statistics reported by Ramulator

Our Key DAMOV Results



When correct statistics are used, DAMOV's memory access latency is **not constant**, in contrast to the claims of the Mess paper

More in the Paper

Cleaning up the Mess: Re-Evaluating the Real-System Modeling Accuracy of Ramulator 2.0

F. Nisa Bostancı* Haocong Luo* Ataberk Olgun* Maria Makeenkova
Geraldo F. de Oliveira A. Giray Yağlıkçı Onur Mutlu

*SAFARI Research Group
ETH Zürich*

A MICRO 2024 best paper runner-up publication (the Mess paper [1]) with all three artifact badges awarded (including “Re-producible”) proposes a new benchmark to evaluate real and simulated memory system performance. The publication contends that Ramulator 2.0 [2] and DAMOV [3] (ZSim+Ramulator) (along with other existing memory system simulators) “poorly resemble the actual system performance” and asserts that their simulator is better.

add sanity checks to aid users and developers of simulation tools.

1. Introduction

Cycle-level DRAM simulators enable research focused on improving the performance, efficiency, and robustness (including security, safety, reliability) of DRAM-based memory systems by providing accurate and flexible models for DRAM and memory controller operations. Ramulator 2.0 [2, 6] (which builds

** Equal contribution*

<https://arxiv.org/pdf/2510.15744>

Cleaning up the Mess is Open Source



Cleaning-up-the-Mess

master 1 Branch 0 Tags

Go to file T Add file <> Code

nisabostanci update README files to be consistent 5d71d72 · 3 weeks ago 3 Commits

DAMOV	initial commit	3 weeks ago
ramulator2	initial commit	3 weeks ago
standalone-ramulator	update README files to be consistent	3 weeks ago
LICENSE	initial commit	3 weeks ago
README.md	update README files to be consistent	3 weeks ago

README MIT license

Cleaning up the Mess - Artifacts

This repository contains the source code and data to reproduce the Ramulator2, DAMOV and standalone Ramulator results of our ISPASS 2026 paper titled "Cleaning up the Mess".

<https://github.com/CMU-SAFARI/Cleaning-up-the-Mess>

Cleaning up the Mess

Analysis of the Evaluation of Ramulator 2.0
in the Mess Paper

Analysis of the Evaluation of DAMOV
in the Mess Paper

Ramulator 2.0 and DAMOV Best Practices

Ramulator 2.0 and DAMOV Best Practices

1

Be familiarized with JEDEC standard documents

instead of relying on information other than the official standard or the simulator documentation

2

Understand Ramulator 2.0's

memory organization and configurations,

which are documented in the code and in the repository

3

**Understand simulation methodology and
simulator implementations**

sufficiently to use the correct statistics

4

Communicate with the simulator developers

or user community when unexpected results are observed
to ensure correct simulator usage

Conclusion

A recent paper, "**A Mess of Memory System Benchmarking, Simulation and Application Profiling**", MICRO'24 makes **strong negative claims** about Ramulator 2.0 and DAMOV and shows **unexpected results**

We demonstrate that these claims and results are **incorrect** and are due to **configuration and simulator usage errors** made in the Mess paper

We strongly encourage the computer architecture community to consider our corrections to the Ramulator 2.0 and DAMOV results of the Mess paper to **prevent the propagation of inaccurate and misleading results**

To facilitate **transparency and reproducibility**, we open source all our code and results: <https://github.com/CMU-SAFARI/Cleaning-up-the-Mess>



Paper



GitHub



Cleaning up the Mess

Re-Evaluating the Real-System Modeling Accuracy of Ramulator 2.0

F. Nisa Bostancı*

Haocong Luo*

Ataberk Olgun*

Maria Makeenkova

Geraldo F. de Oliveira

A. Giray Yağlıkçı

Onur Mutlu

* Equal Contribution

SAFARI

ETH zürich

Ramulator 2.1 - Introduction

- **Ramulator 2.1 is a major overhaul of Ramulator 2.0**
 - ❑ **Fixes bugs** identified by Github issues and internal testing
 - ❑ More comprehensive support for **new DRAM & controller features**
 - ❑ More comprehensive set of **tests and validations**
 - ❑ Significantly improved **ease of use, configuration, and extension**
 - ❑ Overall code quality and documentation improvements
- Github main branch updated to v2.1.0
 - ❑ <https://github.com/CMU-SAFARI/ramulator2/>
 - ❑ codespace and devcontainer ready!
 - ❑ V2.0a is archived as legacy/v2.0a
- **Welcomes your use, contribution, and feedback!**
 - ❑ Issues & Discussions on Github
 - ❑ Open PRs for bug fixes, improvements, and new features

Key Bug Fixes & New Features (I)

■ Currently modeled DRAM Standards

❑ **DDR**

- ✓ DDR3
- ✓ DDR4
- ✓ DDR5

❑ **LPDDR**

- ✓ LPDDR5
- ✓ LPDDR6

❑ **GDDR**

- ✓ GDDR6
- ✓ GDDR7

❑ **HBM**

- ✓ HBM1 (Legacy mode)
- ✓ HBM2 (Pseudochannel mode)
- ✓ HBM3
- ✓ HBM4

■ Modeling scope & priority

- ❑ Timing and functional simulation of DRAM commands
- ❑ Load/Store transactions, refresh, read disturbance mitigation
- ❑ High frequency operation and organization

Key Bug Fixes & New Features (II)

- **Major bugs from Github Issues all fixed**
 - HBM/HBM2/HBM3 organization, timing, throughput, etc.
 - LPDDR5 timing, WCK2CK sync commands, etc.
- **Better support for new DRAM & controller features**
 - **HBMController for HBM/HBM2/HBM3/HBM4**
 - ✓ Pseudo Channels (HBM2+)
 - ✓ Row and Column commands dual-issue in the same cycle
 - **LPDDR5/6 Controller**
 - ✓ WCK2CK sync (CAS_RD/WR) commands with sync state expiry tracking
 - ✓ ACT1-ACT2 separation with tAAD deadline-aware scheduling
 - **GDDR7 Controller**
- **Improved scheduler implementation**
 - Add custom filtering logic to FR-FCFS via lambdas

Key Bug Fixes & New Features (III)

■ Enhanced memory controller plugins

- Standardized and modular interface to flexibly extend the functionality of existing memory controllers

```
class IControllerPlugin {  
    RAMULATOR_REGISTER_INTERFACE(IControllerPlugin, "controller_plugin")  
public:  
    virtual void pre_schedule() {  
    }  
    virtual void on_issue(const Request& req) {  
    }  
    virtual void post_schedule() {  
    }  
};
```

- Enables plug&play implementation of
 - Utilities like DRAM command trace recorder
 - RH mitigations

Tests & Validations (I)

■ Three automated test & validation workflows

- **Smoke tests:** Quick checks to see if simulation crashes
- **DRAM command and request traces:** Low-level verification of DRAM timings and controller scheduling
- **Latency-Throughput Plots:** High-level validation of memory system performance

Tests & Validations (II)

■ DRAM Commands and Timings Test

```
def test_device_under_test_example_flow():
    # Build a normal DRAM object first, then wrap it in DeviceUnderTest.
    dram = ramulator.dram.DDR4(org_preset="DDR4_8Gb_x8", timing_preset="DDR4_2400R", rank=1)
    dut = device_timings.DeviceUnderTest(dram)

    # Named address construction keeps short protocol tests readable.
    a = dut.addr_vec(Rank=0, BankGroup=0, Bank=0, Row=12, Column=0)

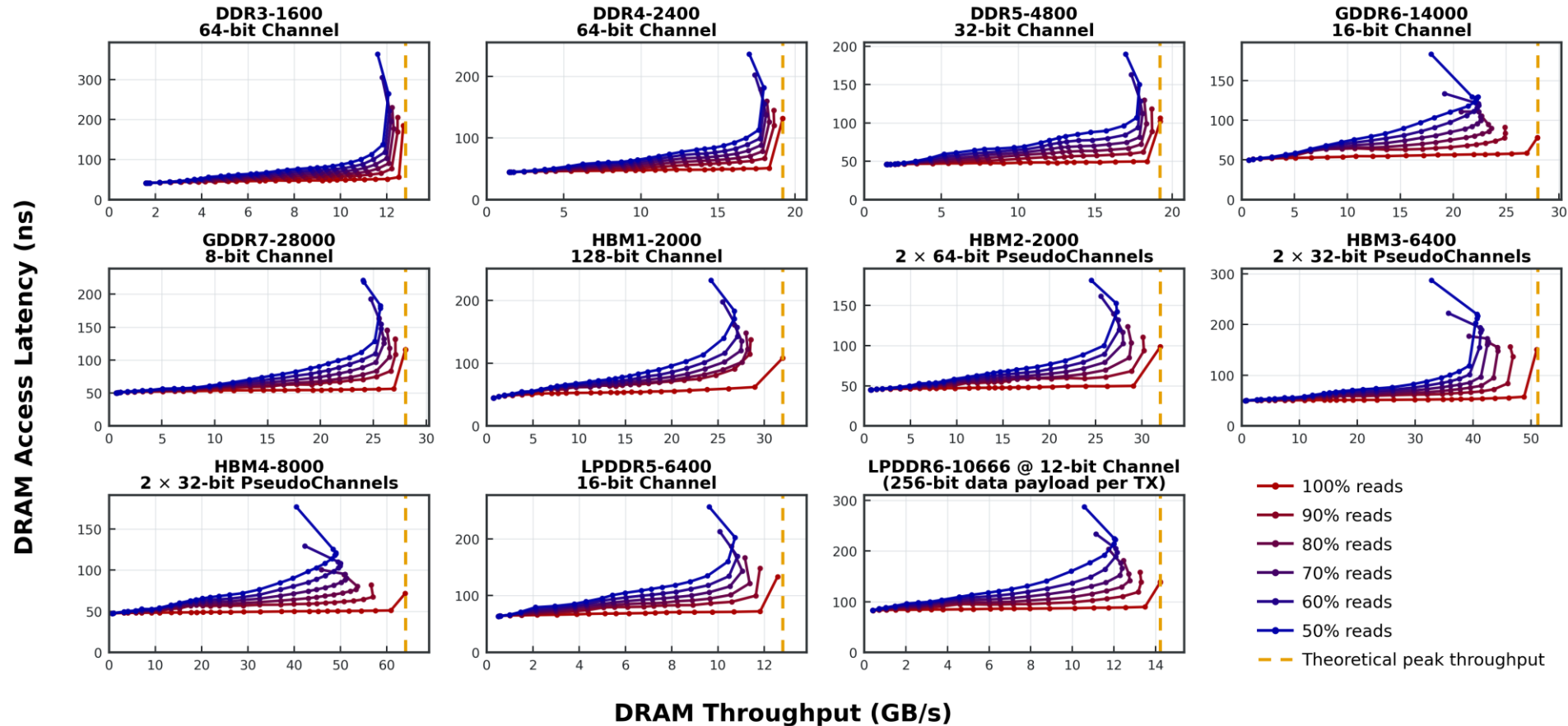
    # A closed-bank read is functionally blocked until the row is opened.
    closed = dut.probe("RD", a, clk=0)
    # Without ACT issued, the prerequisite is ACT.
    assert closed.preq == "ACT"
    # Here we only check the timing constraints. The timing is OK here since no ACT has been issued yet!
    assert closed.timing_OK is True
    # ready means the command is fully issuable now: correct prerequisite and timing_OK.
    assert closed.ready is False

    # Open the row at cycle 0.
    dut.issue("ACT", a, clk=0)

    # Before nRCD, the row state is correct for RD but timing still blocks it.
    early = dut.probe("RD", a, clk=dut.timings["nRCD"] - 1)
    assert early.preq == "RD"
    assert early.timing_OK is False
    assert early.ready is False
    assert early.row_hit is True
    assert early.row_open is True
```

Tests & Validations (III)

■ Latency-Throughput Plots



QoL Improvements (I)

■ **Ramulator is now a python package**

- C++ code with python bindings

■ **Python driven configuration**

- Configuration, simulation, results **parsing all in one script**
- Easy parameter sweeps with python flexibility
- gem5 friendly
 - Both SE and FS works out-of-box (tested with gem5 v25.1)

■ **No extra burden for end-user**

- Single-source of truth implementation
- Python wrappers are **automatically generated** from C++
 - **No need of manual maintenance of boilerplate code**
- Equivalent YAML config can be **automatically generated** for other pure C++ simulators

QoL Improvements (II)

- Write code in C++
- Use macros to register your interface, implementation, parameters

```
RAMULATOR_REGISTER_INTERFACE(IfceClassName, "ifce_name")  
RAMULATOR_REGISTER_IMPLEMENTATION(IfceClassName, ImplClassName, "ImplName")  
  
RAMULATOR_PARSE_PARAM(parsed_variable, type_t, "param_name")  
RAMULATOR_CREATE_CHILD(IfceClassName, "ifce_name")
```

- Ramulator 2.1's software framework and build system automatically picks up these information to:
 - ❑ Understand, register, and build the object hierarchy of your interfaces and implementations
 - ❑ Create Python wrappers (proxy objects) with the specified parameters

QoL Improvements (III)

■ DRAM Standard Authoring in python

- ❑ Replaced verbose and hacky C++ DRAM specs in Ramulator 2.0 with much **cleaner python scripts**
- ❑ Flexible authoring of DRAM organization, commands, and timings, etc. thanks to the scriptibility of Python
- ❑ Code generation utility **automatically generates** C++ DRAM specs that Ramulator 2's internal modeling and simulation engine recognizes

QoL In

■ DRAM

- ❑ Replace with m
- ❑ Flexible etc. th
- ❑ Code g
DRAM
engine

```
class DDR5(DRAMStandard):
    name = "DDR5"
    internal_prefetch_size = 16
    read_latency = "nCL + nBL"

    # ---- Hierarchy (level name → init state) ----
    levels = {
        "Channel": "N_A",
        "Rank": "N_A",
        "BankGroup": "N_A",
        "Bank": "Closed",
        "Row": "Closed",
        "Column": "N_A",
    }

    # ---- Commands ----
    commands = ["ACT", "PREpb", "PREab", "RD", "WR", "RDA", "WRA", "REFab"]

    # ---- CA bus cycle count per command (JEDEC Table 30) ----
    # 2-cycle commands carry address bits that don't fit in 1 CA cycle.
    # The DRAM only begins internal operation after the 2nd cycle.
    # Multi-cycle adjustment applied automatically in to_config().
    command_cycles = {"ACT": 2, "RD": 2, "RDA": 2, "WR": 2, "WRA": 2}

    # ---- States ----
    states = ["Opened", "Closed", "N_A"]

    # ---- Timing parameters (C++ Timing enum order) ----
    timing_params = [
        "rate", "nBL", "nCL", "nRCD", "nRP", "nRAS", "nRC",
        "nWR", "nRTP", "nCWL", "nPPD",
        "nCCDS", "nCCDL", "nCCDS_WR", "nCCDL_WR",
        "nRRDS", "nRRDL", "nWTRS", "nWTRL",
        "nFAW", "nRFC", "nREFI", "nCS", "tCK_ps",
    ]
```

2.0

nings,

ation

class DDR5(DRAMStandard):

---- Timing constraints ----

timing_constraints = [

Channel – data bus occupancy

TimingConstraint(level="Channel", preceding=["RD", "RDA"], following=["RD", "RDA"], latency="nBL"),

TimingConstraint(level="Channel", preceding=["WR", "WRA"], following=["WR", "WRA"], latency="nBL"),

Rank – CAS read timing (different bank group)

TimingConstraint(level="Rank", preceding=["RD", "RDA"], following=["RD", "RDA"], latency="nCCDS"),

Rank – CAS write timing (different bank group, DDR5 separates read/write)

TimingConstraint(level="Rank", preceding=["WR", "WRA"], following=["WR", "WRA"], latency="nCCDS_WR"),

Rank – read-to-write turnaround (tRPST=0.5→2 tCK, tWPRE=2 tCK)

TimingConstraint(level="Rank", preceding=["RD", "RDA"], following=["WR", "WRA"], latency="nCL + nBL + 2 - nCWL +

Rank – write-to-read turnaround

TimingConstraint(level="Rank", preceding=["WR", "WRA"], following=["RD", "RDA"], latency="nCWL + nBL + nWTRS"),

Rank – sibling (rank switching)

TimingConstraint(level="Rank", preceding=["RD", "RDA"], following=["RD", "RDA", "WR", "WRA"], latency="nBL + nCS

TimingConstraint(level="Rank", preceding=["WR", "WRA"], following=["RD", "RDA"], latency="nCWL + nBL + nCS - nCL

Rank – CAS to PREab

TimingConstraint(level="Rank", preceding=["RD"], following=["PREab"], latency="nRTP"),

TimingConstraint(level="Rank", preceding=["WR"], following=["PREab"], latency="nCWL + nBL + nWR"),

Rank – RAS timing

TimingConstraint(level="Rank", preceding=["ACT"], following=["ACT"], latency="nRRDS"),

TimingConstraint(level="Rank", preceding=["ACT"], following=["ACT"], latency="nFAW", window=4),

TimingConstraint(level="Rank", preceding=["ACT"], following=["PREab"], latency="nRAS"),

TimingConstraint(level="Rank", preceding=["PREab"], following=["ACT"], latency="nRP"),

Rank – precharge-to-precharge delay (DDR5-specific nPPD)

TimingConstraint(level="Rank", preceding=["PREpb", "PREab"], following=["PREpb", "PREab"], latency="nPPD"),

Rank – RAS to REF

TimingConstraint(level="Rank", preceding=["ACT"], following=["REFab"], latency="nRC"),

TimingConstraint(level="Rank", preceding=["PREpb", "PREab"], following=["REFab"], latency="nRP"),

TimingConstraint(level="Rank", preceding=["RDA"], following=["REFab"], latency="nRP + nRTP"),

TimingConstraint(level="Rank", preceding=["WRA"], following=["REFab"], latency="nCWL + nBL + nWR + nRP"),

TimingConstraint(level="Rank", preceding=["REFab"], following=["ACT", "PREab"], latency="nRFC"),

QoL Improvements (IV)

■ DRAM Standard Authoring in python

- Significant reduction in lines of source code

Table 1: Reduction in lines of source code of DRAM standards in Ramulator 2.1 (Python) compared to Ramulator 2.0 (C++).

DRAM Standard	Lines of Source Code		Reduction
	v2.0 (C++)	v2.1 (Python)	
DDR3	325	129	60.3%
DDR4	354	161	54.5%
DDR5	402	132	67.2%
GDDR6	327	199	39.1%
HBM1	287	133	53.7%
HBM2	289	146	49.5%
LPDDR5	395	143	63.8%
DDR4-VRR	375	18	95.2%
DDR5-VRR	445	18	96.0%
Total	3,199	1,079	66.3%

QoL Improvements (V)

■ DRAM Standard Authoring in python

- Python scripting enables **easy extension of DRAM standards** by just adding what is new (i.e., no duplication of base DRAM code)
 1. Inherit the base DRAM standard
 2. Append just the new commands, timings, etc. to those of the base DRAM standard

```
from ramulator.dram.ddr3 import DDR3
from ramulator.dram.spec import TimingConstraint

class DDR3Foo(DDR3):
    name = "DDR3Foo"
    commands = DDR3.commands + ["FOO"]
    timing_params = DDR3.timing_params + ["nFOO"]
    timing_constraints = DDR3.timing_constraints + [
        TimingConstraint(level="Bank", preceding=["FOO"], following=["ACT"], latency="nFOO"),
        TimingConstraint(level="Bank", preceding=["ACT"], following=["FOO"], latency="nRC"),
    ]
```

QoL Improvements (III)

■ Web-Based Interactive DRAM Trace Visualizer

- ❑ Supports both **offline** (read dumped trace file) and **online** (from a running simulation) visualization
- ❑ **Inspect memory requests and DRAM commands** for performance analysis & debugging (e.g., timing violation)
 - Many convenient and QoL features WIP
- ❑ Real-time interactive rendering (tested w/ a 300MB+ trace)



Contribute to Ramulator 2.1 (I)

■ Released on Github (current tag v2.1.0)

- ❑ <https://github.com/CMU-SAFARI/ramulator2/>
- ❑ Please checkout v2.1 and let us know if you encounter any issues!
 - We will regularly address issues and check PRs
- ❑ Earlier v2.0a version is archived at legacy/v2.0a

■ How to contribute to Ramulator 2.1

- ❑ Use it, report any issue/feedback you have on Github
- ❑ Add new features and components
- ❑ Add more testcases
- ❑ ...

■ Check our detailed README and rich in-source documentation on using, extending, and integrating Ramulator 2.1

Contribute to Ramulator 2.1 (II)

■ Example directions of contributions

□ Better timing presets of DRAM standards

- Some of the DRAM timings are our best effort “guesstimates”
- We welcome your more accurate set of timings w/o breaking NDA

□ More DRAM request scheduling techniques

□ More DRAM read disturbance mitigation techniques

- Make use of controller plugins and scheduler filters

□ More intelligent scheduling of refresh

- Refresh postponing & pull-in
- Dynamic choice of REFab/sb/pb

□ PUD/PNM modeling

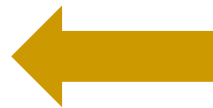
□ More integration with other simulators

■ Please open an Issue/Discussion first to coordinate with potential internal development!

Acknowledgements

■ Contributors (growing)

- ❑ Haocong Luo
- ❑ Nisa Bostanci
- ❑ Ataberk Olgun
- ❑ Maria Makeenkova
- ❑ Ziad Malik
- ❑ Ipek Akdeniz
- ❑ Kislay Arya
- ❑ Roman Hoffmann
- ❑ Wonhyuk Yang
- ❑ @jiegec
- ❑ Julien Eudine
- ❑ Sangjae Park
- ❑ ...



Join the list!