Cleaning up the Mess

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

ETH Zürich

A MICRO 2024 best paper runner-up publication (the Mess paper [1]) with all three artifact badges awarded (including "Reproducible") 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.

In this paper, we demonstrate that the Ramulator 2.0 simulation results reported in [1] are incorrect and, at the time of the publication of [1], irreproducible. We find that the authors of [1] made multiple trivial human errors in both the configuration and usage of the simulators. We show that by correctly configuring Ramulator 2.0, Ramulator 2.0's simulated memory system performance actually resembles real system characteristics well, and thus a key claimed contribution of [1] is factually incorrect. We also identify that the DAMOV simulation results in [1] use wrong simulation statistics that are unrelated to the simulated DRAM performance. Moreover, the Mess paper's artifact repository [4, 5] lacks the necessary sources (simulator code, system configurations, memory traces, etc.) to fully reproduce all the Mess paper's results. We find that the experiment scripts in [4, 5] use simulator executables and other resources that are neither described in the Mess paper nor found in the artifact repository [4, 5].

Our work corrects [1]'s errors regarding Ramulator 2.0 and identifies important issues in [1]'s memory simulator evaluation methodology. We emphasize the importance of both carefully and rigorously validating simulation results and contacting simulator authors and developers, in true open source spirit, to ensure these simulators are used with correct configurations and as intended. Had we, as developers and maintainers of Ramulator 2.0, been properly contacted before the publication of [1], we could have easily identified the errors in the Mess paper and helped the authors of [1] to avoid publishing factually incorrect results and contributions. We encourage the authors of [1] and the computer architecture community to correct [1]'s errors. This is necessary to prevent the propagation of inaccurate and misleading results, and to maintain the reliability of the scientific record. Our investigation also opens up questions about the integrity of the review and artifact evaluation processes. To aid future work, our source code and scripts are openly available at https://github.com/CMU-SAFARI/ramulator2/tree/mess.

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 on Ramulator [7, 8]) is a highly modular and extensible cycle-accurate DRAM simulator that enables rapid exploration of new ideas in DRAM-based memory systems.

A MICRO 2024 best paper runner-up publication, *A Mess of Memory System Benchmarking, Simulation and Application Profiling* [1], which we refer to as "the Mess paper" throughout this paper, with all three artifact badges awarded (including "Reproducible") proposes a new benchmark to evaluate real and simulated memory system performance. While doing so, it makes some strong negative claims about Ramulator 2.0 and DAMOV [3, 9], which we demonstrate to be incorrect and are due to configuration errors made in the Mess paper.

Fig. 1-(a) and (b) (copied from Fig. 6 and Fig. 4 in [1]) show average memory latency as a function of throughput (used memory bandwidth) for the Mess benchmark using a Ramulator 2.0 simulation and a real system, respectively, as depicted in [1].[1] From these two figures, the authors of the Mess paper conclude that Ramulator 2.0 "poorly resembles the actual system performance" (Section I of [1]). They write "The simulated memory latency is unrealistically low and the maximum simulated memory bandwidth is only 126 GB/s which is less than a half of the 292 GB/s measured in the actual system" (Section IV of [1]). They attribute the discrepancy between Mess benchmark results of Ramulator 2.0 simulations and the real system to Ramulator 2.0-induced simulation errors: "This indicates that the main source of the large simulation error is indeed Ramulator 2" (Section 4 of [1]). The authors also show other odd and unexpected results. For example, the DRAM latency results from DAMOV (zsim+Ramulator) [3] (depicted in Section 4 of [1]) are unrealistically low and they seem to remain constant regardless of system load.

In this paper, we show that the Ramulator 2.0 simulation results in the Mess paper are *incorrect* and, at the time of the publication of [1], *irreproducible*. Fig. 1-(c) shows our reproduction of the memory latency-bandwidth curves for the Mess benchmark using the open source version of Ramulator 2.0 [2] after correcting the error we found in [1].[2] These results clearly show that Ramulator 2.0 does *not* poorly resemble real system performance contrary to what the Mess paper claims.

We carefully investigate the Mess paper's open source artifacts [4, 5] and have communicated with the Mess paper's

[1] The Mess benchmark consists of two workloads called *Stream* and *Pointer-Chase*. These workloads run concurrently. The interval between subsequent memory requests in Stream is controlled using NOPs to vary memory intensity. For example, 1K NOPs between subsequent memory requests lead to a relatively low memory intensity and 0 NOPs between subsequent memory requests lead to a very high memory intensity. Each latency-bandwidth curve in Fig. 1 is constructed by connecting Mess benchmark latency-bandwidth results in increasing memory intensity order such that the leftmost end of the curve corresponds to the smallest memory intensity and the other end of the curve corresponds to the largest memory intensity Mess benchmark configuration. Each curve depicts system performance for a unique mixture of read and write memory requests in Stream ranging from 50% reads and 50% writes to 100% reads and 0% writes.

[2] We freely and openly release all sources and data used in this work at https://github.com/CMU-SAFARI/ramulator2/tree/mess. One can reproduce *all* our graphs using the scripts we provide in that repository.

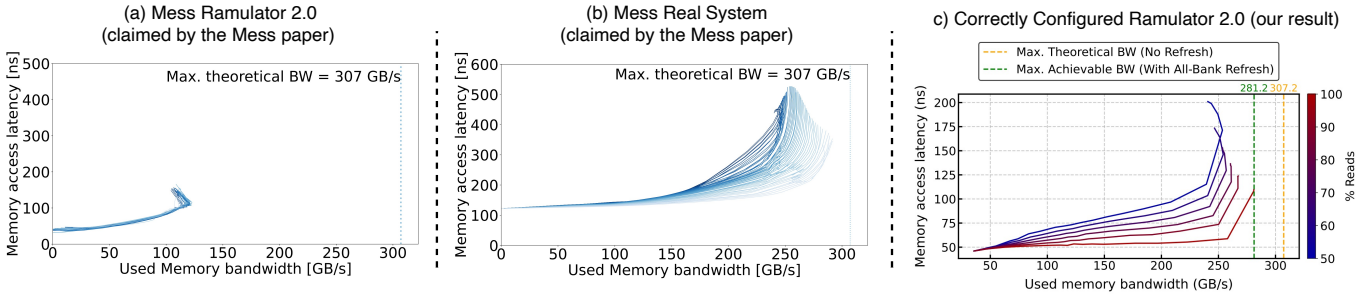


Fig. 1: Mess results for Ramulator 2.0 memory system simulation from Fig. 6 in [1] (a), Mess results for an ARM-based real system from Fig. 4 in [1] (b), our results for Ramulator 2.0 memory system simulation after correcting the errors we found in [1] (c), as explained in detail in Section 2 of this paper

authors to identify the root causes of the errors in [1]. We identify two major Ramulator 2.0 configuration errors and three other scholarship issues regarding Ramulator 2.0 in the Mess paper. We describe these issues in detail in Sections 2.3 and 2.4. In short, the Ramulator 2.0 results in the Mess paper are subject to gross configuration errors. Therefore, these results do *not* support the strong negative claims (asserted in the Mess paper) about Ramulator 2.0's modeling accuracy.[3] The configuration errors could have been avoided by careful and rigorous validation and/or by contacting Ramulator 2.0 authors and developers (before publishing such incorrect results) to ensure the simulator is correctly used with the correct configuration and as intended.

We also identify that the reason for unrealistically low and seemingly constant DRAM latency results for DAMOV from the Mess paper is that the Mess paper's authors report simulator statistics that are *not* updated during the DRAM simulation. Moreover, the Mess paper's artifact repository [4, 5] does *not* contain the relevant DRAM simulation statistics from Ramulator for DAMOV. We find that the experiment scripts of the Mess paper use simulator executables and other resources that are neither described in the Mess paper nor found in the Mess paper's artifact repository [4, 5].

Our work corrects [1]'s errors regarding Ramulator 2.0 and identifies important scholarship issues in [1]'s memory simulator evaluation methodology. We emphasize the importance of contacting simulator authors and developers (especially before publishing), in true open source spirit, to ensure these simulators are used with correct configurations and as intended. We encourage the authors of the Mess paper and the computer architecture community to correct [1]'s errors to prevent propagation of inaccurate and misleading results and to maintain the reliability of the scientific record. Our investigation also opens up questions about the integrity of the review and artifact evaluation processes.

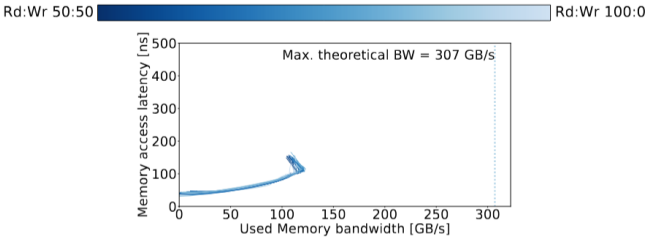
2. Analysis of the Evaluation of Ramulator 2.0 in the Mess Paper

2.1. The Mess Benchmark

The Mess benchmark aims to characterize a memory system based on a set of bandwidth-latency curves. To this end, the Mess paper introduces the Mess benchmark. The Mess benchmark is made up of two concurrently running workloads, *Stream* and *Pointer Chase* [10, 11]. Each curve is given by running these workloads with a pre-configured fraction of read and write requests for the Stream, ranging from 50% reads and 50% writes to 100% reads. Each point on a curve is given by a run of the benchmark with varying memory intensity. This is achieved by varying the interval between subsequent Stream requests by inserting NOPs. For example, 1K NOPs between subsequent memory requests lead to a relatively low memory intensity, and 0 NOPs between subsequent memory requests lead to a very high memory intensity.

2.2. Mess Paper's [1] Ramulator 2.0 Results

Fig. 2 (Fig. 1-a plotted again for the reader's convenience) shows the bandwidth-latency curves presented in [1] (Fig. 6 in [1]) for Ramulator 2.0 obtained with the Mess benchmark. This result unexpectedly and oddly shows that the simulated memory system 1) uses significantly smaller bandwidth than what is available (depicted as the "Max. theoretical BW" in the figure) and 2) yields a relatively small memory access latency.



(a) Ramulator2: 8×DDR5-4800

Fig. 2: Latency-bandwidth curves for Ramulator 2.0 (copied from Fig. 6 in [1])

We note that it was *impossible* to reproduce *any* Ramulator 2.0 result from the Mess paper artifacts available at the time of the publication and artifact evaluation of the Mess paper [4, 5]. Thus, we contacted the authors of [1] on 16 October 2024 to obtain 1) the Ramulator 2.0 source code and 2) the Mess benchmark memory traces used in Ramulator 2.0 simulations reported in [1]. The missing sources needed to replicate the Mess paper's Ramulator 2.0 results were eventually uploaded to the Mess Github repository [5] following our contact on 18 July 2025, *10 months after* the Mess paper was awarded (on 8 September 2024) all three artifact evaluation badges.

[3] Strong negative assertions made in the Mess paper such as "... the main source of the large simulation error is indeed Ramulator 2..." (Section 4 of [1]) and "... the most complex and trusted memory model shows the highest simulation error..." (Section 4 of [1]) are incorrect because the source of the error is the misconfiguration and misuse of Ramulator 2.0 in the Mess paper.

2.3. Fundamental Configuration Errors in [1]

We identify two fundamental configuration errors in the Mess paper's Ramulator 2.0 simulations. We attribute the discrepancy in bandwidth and memory access latency between the Mess paper's Ramulator 2.0 simulations and real system evaluation to these fundamental errors. We also identify three other scholarship issues.

First, the Mess paper's scripts (in the newly-uploaded source code on 18 July 2025) instantiate and use a single-channel DDR5 Ramulator 2.0 memory system. The authors extrapolate single-channel DDR5 results to 16-channel DDR5 results by multiplying the observed bandwidth by 8 instead of 16. This is because they incorrectly assume that a DDR5 channel in Ramulator 2.0 is 64 bits wide, consisting of two independent 32-bit channels. However, the source code of Ramulator 2.0 clearly shows that a DDR5 channel is 32 bits wide[4] in Ramulator 2.0. The Mess paper misleadingly presents (extrapolated) 8-channel (32-bit each) DDR5 results as 16-channel (32-bit each) DDR5 results, leading to incorrect claims like "...the maximum simulated memory bandwidth is only 126 GB/s which is less than a half of the 292 GB/s measured in the actual system" in Section 4 of [1]. The maximum bandwidth of an 8-channel DDR5-4800 [12] is 153.6 GB/s [12], *not* 307 GB/s. Thus, a bandwidth of more than 153.6 GB/s should *not* be expected of an 8-channel DDR5-4800 configuration. We show that a correctly configured 16-channel DDR5-4800 Ramulator 2.0 simulation achieves 281.1 GB/s bandwidth (as depicted in Fig. 1-c; see Section 4 for a detailed description).

Second, the Mess paper's scripts use the SimpleO3 frontend of Ramulator 2.0 [6] with unrealistically low latency configurations compared to a real system. For example, they configure 1) the cache latency to be *0* CPU clock cycles and 2) each CPU core to have 1024 miss-status holding registers (MSHRs). Compared to a real CPU, such a configuration has 1) significantly smaller round-trip latency between a core and DRAM, 2) significantly lower queuing delay for memory requests when memory intensity is high. Thus, the Mess paper's claim about Ramulator 2.0 that "...the simulated memory latency is unrealistically low..." (Section 4 of [1]) is unfair since the Mess paper's unrealistically low-latency configuration of Ramulator 2.0 is in part responsible for such low latency.

2.4. Scholarship Issues in [1]

First, the Mess paper's authors did *not* contact us regarding the results they obtained with their configuration of Ramulator 2.0 before the Mess paper was published. Therefore, we could *not* help them correct the issues before it was in scientific record. Their misconfiguration of the number of channels in DDR5 could have been easily resolved by emails and collegial discussions which are in the spirit of open-source development.

Second, the Mess paper does *not* disclose the unrealistically low latency configurations of the SimpleO3 frontend of Ramulator 2.0 in the paper. The Ramulator 2.0 configuration file available in the artifact repository also does *not* include any of these configurations. After the Mess paper's authors uploaded their Ramulator 2.0 source code (10 months after the Mess paper was awarded all three artifact evaluation badges), we found (via email exchanges with the Mess paper's authors) that they implement the unrealistically low latency configurations by *directly modifying the source code of Ramulator 2.0*, without any changes in the Ramulator 2.0 configuration file.

Third, the Mess paper uses *different* Mess benchmark methodologies for real systems and simulators *without* disclosing this difference in the paper.[5] For real systems (e.g., Fig. 5-a in [1]), the Mess benchmark is run as described in the Mess paper (i.e., by concurrently running *pointer-chasing* and *Stream* workloads and measuring the memory latency of *pointer-chasing* workload's memory accesses). For simulators (e.g., Figures 5-b, -c, -d, -e, and -f in [1]), the authors *do not run a pointer-chasing workload* and they measure the average memory latency for *only* the *Stream* workload. Thus, the Mess paper's authors compare apples to oranges by running and measuring different workloads in simulators and in real systems. And, this difference in simulation and real system evaluation methodologies is *not* disclosed in the Mess paper.

Through our comprehensive investigation of [1]'s Ramulator 2.0 artifacts, we conclude that the Ramulator 2.0 results in [1] are subject to multiple serious configuration and methodological errors that could have been easily avoided by careful and rigorous validation and/or by properly contacting the authors of Ramulator 2.0 in the true open-source spirit before the submission (and certainly before publication) of the Mess paper.

3. Our Evaluation Methodology

To show that Ramulator 2.0, when configured properly and used correctly, yields reasonable and realistic latency-bandwidth curves, we evaluate a modern DDR5 memory system in Ramulator 2.0 using the same access patterns as described in the Mess benchmark.

We introduce a new Ramulator 2.0 frontend module, the *Mess Request Generator*, which we open source, along with all our code and scripts in our repository [13] dedicated to this work. Our Mess Request Generator frontend is designed to emulate the memory access patterns of the Mess benchmark by generating a mixture of random access (*random*) and streaming (*stream*) read requests. The request generator issues *random* requests sequentially (to emulate a pointer-chasing workload), meaning that the request generator issues a new random request *only* when the previous *random* request returns from the memory controller. The request generator issues a configurable mix of *stream* read and *stream* write requests to the memory controller as often as every DRAM command interface clock cycle (e.g., $0.416\,\text{ns}$ for DDR5-4800 [12]). We modulate the frequency of *stream* requests using NOPs between two consecutive requests. The sequence of *stream* requests are designed to take advantage of bank-group- and bank-level parallelism and row hits to fully utilize DDR5 data transfer bandwidth. The request generator keeps issuing *stream* requests while waiting for a *random* read request to return from the memory controller.

Read/Write Mix. The Mess Request Generator frontend for Ramulator 2.0 is configurable with a read ratio between 0.5 and 1, which is applied to the stream memory requests. A read ratio of 1 corresponds to streaming accesses consisting of 100% reads, while a read ratio of 0.5 yields 50% read and 50% write requests for the streaming memory accesses. We use

[4] The `m_channel_width` parameter in DDR5.cpp in the Ramulator 2.0 repository [6].

[5] Identified via email exchanges with the Mess paper's authors.

read ratios of 0.5, 0.6, 0.7, 0.8, 0.9, and 1.0 in our evaluation.

***Stream* Bandwidth Use Modulation.** To simulate various system loads, we further extend the Mess Request generator with a configurable NOP frequency. In every Ramulator 2.0 frontend cycle, the Mess Request generator can either issue a memory request or not, depending on the value of the NOP frequency. For example, with a NOP frequency of 1, every cycle leads to a memory request being issued, while a NOP frequency of 100 means that every 100 simulated cycles, a memory request is issued.

We use the Mess Request Generator frontend for Ramulator 2.0 to construct the latency bandwidth curves. We use the Ramulator 2.0 memory system configuration in Table 1. We open source our Ramulator 2.0 configuration along with all source code and scripts to reproduce our results in [13].[6]

Frontend:	Mess Request Generator for Ramulator 2.0 [2]
DRAM:	DDR5-4800AN [12]; 16 channels, 1 rank, 8 bank groups, 4 banks
Timing Parameters:	t_{RCD}: 14.166 ns, t_{RAS}: 32.000 ns, t_{RP}: 14.166 ns, t_{RTP}: 7.5 ns, t_{CCD_S}: 8 nCK, t_{RFC}: 295 ns, t_{REFI}: 3.9 µs
Memory System:	FR-FCFS request scheduling policy [14, 15], All-bank refresh, Read queue size: 32, Write queue size: 32

Table 1: Simulated System Configuration

To construct each latency-bandwidth curve, we run the simulation for 20K *random* read requests and measure *only* the latency of the *random* read requests (i.e., same as in the Mess benchmark). We sweep NOP frequency from 1 to 10000 to vary system load.

4. Key Results

Fig. 3 shows the latency-bandwidth curves generated using Ramulator 2.0 with the Mess Request Generator frontend and the configuration given by Table 1. The maximum theoretical bandwidth for 16 channels of DDR5-4800 (indicated by the orange line in the figure) is 307.2 GB/s (as calculated by 16×19.2 GB/s [12]). The maximum bandwidth *cannot* be sustained during standard operation due to *all-bank periodic refresh operations* [12] that conflict with and delay data transfers. The memory controller can sustain the maximum bandwidth as long as subsequent $READ/WRITE$ commands are issued at the end of every short column-to-column delay ($tCCDS$). To serve a periodic refresh operation, the memory controller must precharge the open bank and issue a refresh command. Doing so introduces a minimum timing penalty of the sum of read-to-precharge ($tRTP$), precharge latency (tRP), refresh latency ($tRFC$), and activation latency ($tRCD$) between two subsequent $READ$ commands. Taking such refresh operations and their impact on data transfers into account, we compute the *maximum achievable bandwidth* from the maximum theoretical bandwidth as 281.2 GB/s (indicated by the green line in the figure) from the following equations:

$$BW_{achievable} = BW_{theoretical} \cdot \left(1 - \frac{REF_{penalty}}{tREFI}\right)$$

$$REF_{penalty} = tRTP + tRP + tRFC + tRCD$$

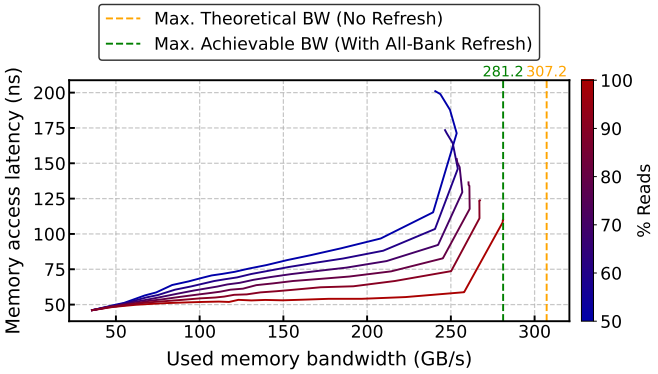


Fig. 3: Latency-bandwidth curves for DDR5-4800AN in Ramulator 2.0 using the Mess Request Generator frontend and configuration in Table 1

We make three major observations from Fig. 3. First, we observe that the maximum achieved memory bandwidth (281.1 GB/s for a read ratio of 1, i.e., 100% reads, and NOP frequency of 1, i.e., no NOP cycles) almost exactly matches the maximum achievable memory bandwidth. We thoroughly investigate the DRAM command sequence modeled by Ramulator 2.0 to understand the small difference between the maximum achieved bandwidth (281.1 GB/s) and the maximum achievable bandwidth (281.2 GB/s) values. We observe that the $REF_{penalty}$ slightly varies for each periodic refresh operation. For example, if the memory controller opens a DRAM row to serve a random access read request and schedules a periodic refresh operation immediately after this read request, the delay between two $READ$ commands interrupted by a refresh operation is slightly higher than $tRTP + tRP + tRC + tRCD$: $(tRAS - tRCD) + tRP + tRC + tRCD$.

Second, as the fraction of write memory requests increases, the latency-bandwidth curves are curved more aggressively towards the top left corner of the figure because a larger fraction of write memory requests induces a higher latency and bandwidth penalty due to more frequently incurred DRAM bus turnaround latency [16].

Third, we observe that the latency of the random access memory requests consistently increases with used memory bandwidth because interference from *stream* requests in the memory controller read/write queues increases, which in turn increases the queuing delay for the random read requests.[7]

[6] Our Ramulator 2.0 codebase [13] dedicated to this work introduces *only* minimal and necessary changes to the open source Ramulator 2.0 repository [6]. First, we implement the Mess Request Generator frontend we describe. Second, we add the timing parameters for the DDR5-4800AN standard [12] to remain consistent with what is evaluated in the Mess paper [1]. Third, we disable write-to-read forwarding in the memory controller to ensure all memory requests are served by DRAM. Fourth, we add utility functions to record various statistics about memory requests and to programmatically configure the Mess Request Generator.

[7] Note that the memory requests are *only* subject to delays incurred at the memory controller because our simulation injects read and write requests directly into the memory controller read and write queues.

We conclude that Ramulator 2.0, when configured and used correctly, yields reasonable and realistic latency-bandwidth curves that resemble the ones obtained from real systems by [1] (see Section 3, Fig. 3 in [1]). The results we present in this section thus show that the Ramulator 2.0 results presented in [1] are incorrect and are *not* representative of Ramulator 2.0's modeling accuracy and capability.

5. Evaluation of DAMOV in the Mess Paper

5.1. Mess Results for DAMOV [3] (zsim [17]+Ramulator [7])

Fig. 4 shows the latency-bandwidth curves of various zsim [17] memory models (sub-figures b, c, d, e, and f), compared to real system measurements (sub-figure a), from the Mess paper [1] (Fig. 5 in [1]). The Mess paper claims DAMOV (i.e., zsim+Ramulator) [3] "provides a fixed 25 ns latency in the whole bandwidth area and for all memory traffic configurations" (Section 4 in [1]).

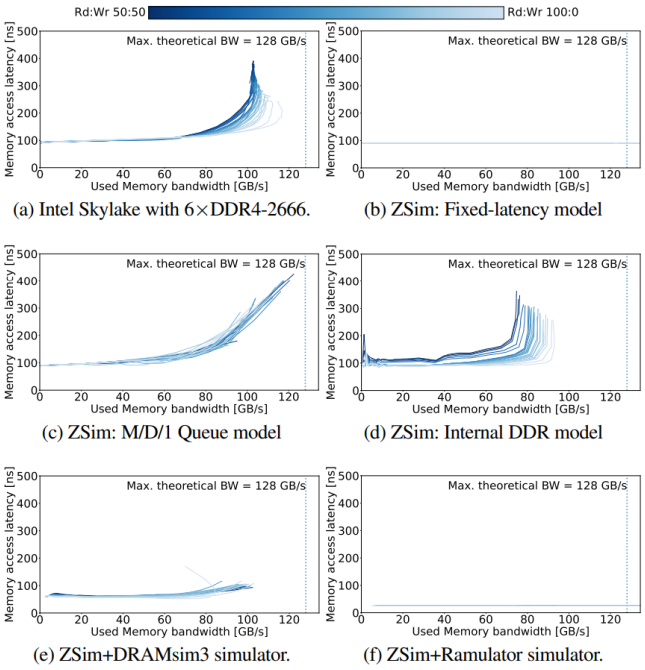


Fig. 4: Latency-bandwidth curves for zsim memory models from the Mess paper [1] (same as Fig. 5 in [1])

To understand why the Mess paper shows such an unrealistic (i.e., low and seemingly constant) memory latency for DAMOV, we carefully checked the code and scripts from the artifact repository of the Mess paper [4, 5, 18]. We identify two major issues: 1) irreproducible simulation, and 2) erroneous memory latency statistics usage, which we describe in Sections 5.2 and 5.3.

5.2. Irreproducible Simulation

We carefully checked the artifact repositories of the Mess paper [4,5,18] based on the description of the artifact in Appendix C of [1]. Unfortunately, we could *not* find the following three items used by the Mess paper to produce the results in Fig. 5 (f) in the Mess paper: 1) the source code of the zsim simulator, 2) the source code of Ramulator, and 3) the configuration of Ramulator used. We also checked the path to the simulator executable in the experiment scripts from the Mess paper artifact [4,5,18]. Unfortunately, the *hardcoded* path[8] in [4,5] points to a directory that does *not* exist in the artifacts [4, 5, 18].

We tried to independently reproduce the Mess paper's results (i.e., Fig. 5-f in [1]) by combining the zsim configuration available in the artifact repository of the Mess paper with the Ramulator configuration from DAMOV [3]. We modify the Ramulator configuration to model the 6×DDR4-2666 configuration from the Mess paper in a best-guess manner. We perform the simulation using 1) the zsim and Ramulator code from DAMOV [3], and 2) the Mess paper's `stream_mpi` workload. However, doing so triggers a rigid assertion in DAMOV because it does not support non-power-of-2 numbers of DRAM channels. Therefore, we cannot independently reproduce the DAMOV results in Fig. 5-f of the Mess paper using the configuration that the Mess paper describes that it used to produce Fig. 5-f in [1].

5.3. Erroneous Memory Latency Statistics Usage

We find that the Mess paper uses two statistics from zsim to calculate the average memory latency for DAMOV, `latGETnl` (i.e., the accumulative latency) and `mGETs` (i.e., the number of GET requests) from the L1-D cache.[9] These are *not* the correct statistics to compute the memory latency for DAMOV because the bound-weave simulation methodology of zsim [17] 1) only updates `latGETnl` with constant latencies (i.e., contention-free latency) throughout the memory hierarchy during the bound phase, and 2) only updates the *core clock cycle* (instead of `latGETnl`) with the actual simulated DRAM latency from Ramulator during the weave phase. In other words, the `latGETnl` statistics from zsim does *not* include any actual DRAM latency simulated by Ramulator, but only includes the *constant* contention-free latency.

We find that the average memory latency for DAMOV as the Mess paper measures (i.e., `latGETnl` / `mGETs`) is about 48 cycles. This matches closely with the accumulative contention-free latency from L1-D to the memory controller of 52 cycles (i.e., about 25ns for 2.1GHz CPU frequency) based on the zsim configuration[10] of the Mess paper [1].

The DRAM performance metrics should come from Ramulator statistics output itself. Unfortunately, *all* the Ramulator statistics output[11] in the Mess artifact are *empty*. We therefore believe that the results in the Mess paper for DAMOV are reflecting incorrect statistics due to simulator usage error. We suggest that the researchers carefully use, understand, and configure simulators before claiming strongly that the simulators they use are inaccurate.

6. Conclusion

We demonstrated that the claims made by the Mess paper about Ramulator 2.0 [2] and DAMOV [3] are incorrect and are due to simulator configuration and simulator usage errors

[8] `/home/bsc18/bsc18278/zsimdramsim3/acmSimulation/zsim.git_mt/build/debug/zsim` from line 33 of `Mess-benchmark/CPU/Simulators/Execution-driven/ZSim/ramulator/submit.batch`

[9] From lines 162-163 in `Mess-benchmark/CPU/Simulators/Execution-driven/ZSim/ramulator/processing/calculator.py`

[10] From `Mess-benchmark/CPU/Simulators/Execution-driven/ZSim/ramulator/sb.cfg`

[11] E.g., `Mess-benchmark/CPU/Simulators/Execution-driven/ZSim/ramulator/measuring/bw-lat/measurment_100_0/bwLatCurves.ramulator.stats`

made in the Mess paper [1] We open source all our code and results at https://github.com/CMU-SAFARI/ramulator2/tree/mess. We encourage the authors of the Mess paper and the computer architecture research community to correct these issues. More broadly, simulators should be configured, understood, and used very carefully and rigorously, especially before making claims about them being wrong. The open source spirit also suggests that the authors of freely available simulators are properly contacted (especially before publishing) to ensure these simulators are used with correct configurations and as intended. Our results also open up questions about the integrity of the review and artifact evaluation processes.

References

[1] Pouya Esmaili-Dokht, Francesco Sgherzi, Valéria Soldera Girelli, Isaac Boixaderas, Mariana Carmin, Alireza Monemi, Adrià Armejach, Estanislao Mercadal, Germán Llort, Petar Radojković, Miquel Moreto, Judit Giménez, Xavier Martorell, Eduard Ayguadé, Jesus Labarta, Emanuele Confalonieri, Rishabh Dubey, and Jason Adlard. A Mess of Memory System Benchmarking, Simulation and Application Profiling. In *MICRO*, 2024.

[2] 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. *IEEE CAL*, 2024.

[3] SAFARI Research Group. DAMOV: A New Methodology and Benchmark Suite for Evaluating Data Movement Bottlenecks. https://github.com/CMU-SAFARI/DAMOV.

[4] Esmaili-Dokht, Pouya. A Mess of Memory System Benchmarking, Simulation and Application Profiling. https://zenodo.org/records/13748674, 2024.

[5] Memory systems for HPC and AI @BSC. Mess benchmark. https://github.com/bsc-mem/Mess-benchmark, 2024.

[6] SAFARI Research Group. Ramulator 2.0. https://github.com/CMU-SAFARI/ramulator2.

[7] Yoongu Kim, Weikun Yang, and Onur Mutlu. Ramulator: A Fast and Extensible DRAM Simulator. *IEEE CAL*, 2015.

[8] SAFARI Research Group. Ramulator. https://github.com/CMU-SAFARI/ramulator.

[9] Geraldo F Oliveira, Juan Gómez-Luna, Lois Orosa, Saugata Ghose, Nandita Vijaykumar, Ivan Fernandez, Mohammad Sadrosadati, and Onur Mutlu. DAMOV: A New Methodology and Benchmark Suite for Evaluating Data Movement Bottlenecks. *IEEE Access*, 2021.

[10] Rommel Sánchez Verdejo and Petar Radojkovic. Microbenchmarks for Detailed Validation and Tuning of Hardware Simulators. In *HPCS*, 2017.

[11] Rommel Sánchez Verdejo, Kazi Asifuzzaman, Milan Radulovic, Petar Radojković, Eduard Ayguadé, and Bruce Jacob. Main Memory Latency Simulation: The Missing Link. In *MEMSYS*, 2018.

[12] JEDEC. *JESD79-5C: DDR5 SDRAM Standard*, 2024.

[13] SAFARI Research Group. Ramulator 2.0 – Mess benchmark. https://github.com/CMU-SAFARI/ramulator2/tree/mess.

[14] Scott Rixner, William J. Dally, Ujval J. Kapasi, Peter Mattson, and John D. Owens. Memory access scheduling. In *ISCA*, 2000.

[15] William K Zuravleff and Timothy Robinson. Controller for a synchronous DRAM that maximizes throughput by allowing memory requests and commands to be issued out of order. US Patent 5,630,096, 1997.

[16] Chang Joo Lee, Veynu Narasiman, Eiman Ebrahimi, Onur Mutlu, and Yale N Patt. DRAM-Aware Last-Level Cache Writeback: Reducing Write-Caused Interference in Memory Systems. Technical Report HPS-2010-002, 2010.

[17] Daniel Sanchez and Christos Kozyrakis. ZSim: Fast and Accurate Microarchitectural Simulation of Thousand-core Systems. In *ISCA*, 2013.

[18] Memory systems for HPC and AI @BSC. Mess simulator. https://github.com/bsc-mem/Mess-simulator, 2025.