This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LCA.2015.2402435, IEEE Computer Architecture Letters

NVMain 2.0: A User-friendly Memory Simulator to Model (Non-)Volatile Memory Systems

Matt Poremba, Tao Zhang, Yuan Xie
Department of Computer Science and Engineering
The Pennsylvania State University, PA, USA 16802
{poremba, zhangtao, yuanxie}@cse.psu.edu

Abstract—In this letter, a flexible memory simulator – NVMain 2.0, is introduced to help the community 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. Compared to existing memory simulators, NVMain 2.0 features a flexible user interface with compelling simulation speed and the capability of providing sub-array-level parallelism, fine-grained refresh, MLC and data encoder modeling, and distributed energy profiling.

1 Introduction

In the past decades, the evolution of DRAM technology has continued to provide a low-latency, high-bandwidth, and low-power main memory solution. Correspondingly, several DRAM simulators have been developed in recent years to accelerate research on the main memory subsystem [5], [11]. Unfortunately, these simulators are dedicated to DRAM modeling and are not built to provide accurate models of new types of memory such as emerging non-volatile memory (NVM) technologies (e.g., STT-RAM, PCRAM, and ReRAM). For example, these DRAM simulators are lack of the capability of endurance and fault modeling, support for multi-level cells (MLC), or hybrid memory systems. Moreover, major modification is required to simulate the hardware/software controlled memory page migration in hybrid systems and giga-scale DRAM caches.

As a result, a new memory simulator equipped with NVM support as well as high flexibility and friendly user interface is beneficial for researchers in memory subsystem so they can quickly start their studies with little modification. In this work, NVMain 2.0 is demonstrated as an improvement over the NVMain [9] memory simulator to fill the gap. Many new features have been designed in NVMain 2.0 to support both DRAM and NVM simulation, including A fine-grained memory bank model, MLC support, more flexible address translation, and "hooks", to encourage users to explore new memory system designs.

In this letter, we first give background on the new design and unique features in NVMain 2.0, then provide experimental results through multiple case studies, and finally discuss our verification process.

2 NVMain 2.0 Architecture

There are multiple research interests in non-volatile memory that currently have modeling difficulties. 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. In addition, a hybrid system that has both NVM and DRAM, each with their own timings, is also becoming increasingly popular but requires more flexible simulators to be modeled easily.

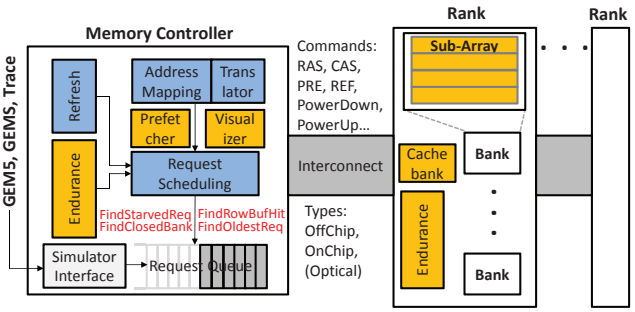

Fig. 1: Overview of NVMain Architecture. Only one memory controller with one channel is shown.

2.1 Energy Modeling

Existing DRAM memory simulators utilize published current numbers (IDD values) measured on an actual device. This approach is problematic when published data is not available, such is the common case for non-volatile memories and prototype memory system designs.

In order to work around this problem, we provide two device level energy models in NVMain 2.0: *Current-mode* and *Energy-mode*, and provide independent power calculations for remaining modules in the memory subsystem. The Current-mode model operates similar to standard DRAM simulators using IDD values for power calculation and is applicable to the simulated DRAM systems or DRAM portions of hybrid memory systems.

Alternatively, Energy-mode allows use of values readily obtained from circuit-level simulators such as NVSIM [4] or CACTI [8]. Each operation increments the system energy usage. Standby and powerdown energies are also calculated using the simulated leakage results.

2.2 Non-volatile Memory Support

Non-volatile memory requires concepts outside the realm of DRAM. 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

Poremba, Zhang, and Xie were supported in part by NSF 1218867, 1213052, 1409798. This material is based on work supported by the Department of Energy under Award Number DE - SC0005026.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LCA.2015.2402435, IEEE Computer Architecture Letters

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. We show in Section 3.1 how important data values can be and posit that simulating any data encoding for endurance or fault recovery mechanisms if very beneficial.

2.3 Fine-grained Memory 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 [1].

2.4 Memory System Flexibility

Figure 1 shows the high-level design of NVMain [9]. In the figure, each box represents a memory base object. 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.

We leverage this distributed approach to introduce two new concepts in NVMain 2.0: *Advanced Address Translators* and *Memory Object Hooks*. First, we change the flow of requests between memory objects to always invoke the address translator. Second, we allow for "plugin" memory objects called *hooks* that can snoop in-flight requests and potentially change request flow.

Address Translators Typically, the address translator takes a memory address and extracts specific bit values to ultimately determine the destination bank of a request. In this work, we further generalize this concept by simply defining the address translator as a function taking a memory address and returning the destination bank. An example system utilizing this change is presented in 3.2, where a hybrid memory system with hardware-based page migrator is implemented. In this particular case, the address is translated as normal and in parallel the address is checked against the migrated pages list. If found, the channel is changed to the "fast" memory and automatically rerouted to that channel's memory controller.

Memory Object Hooks Hooks are external memory objects which can snoop on requests arriving or returning from particular memory objects. For example, one may want to inspect all memory requests arriving at a specific rank or bank object. In this work, hooks are used to inspect memory commands arriving at each rank in order to provide output suitable for verification against Micron's model [7] or DRAMPower2 [2], which require input at the rank level. Inspection at the bank level is useful for visualization purposes, where we wish to instrument a specific bank and observe memory requests in action.

Other Memory Objects Due to the distributed nature of the simulator design, it is possible to implement many different types of memory objects that can be selected at simulation time. Several of these memory objects are provided with the NVMain 2.0 distribution, including but not limited to FCFS, FRFCFS, FRFCFS with write queue, and DRAM cache memory controllers; transmission-line and TSV-based interconnect models; and DDR and LPDDR2-N style banks. These objects exist solely for the purpose of a quick start and a reference material for implementing your own memory objects. More details are available at http://nvmain.org/.

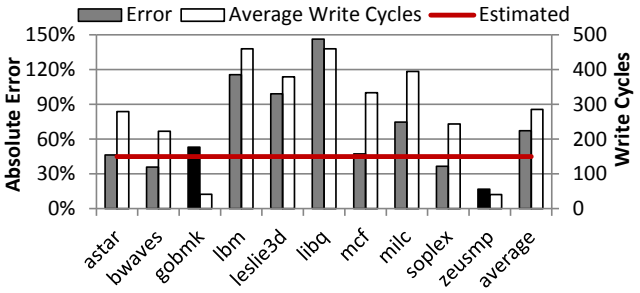

Fig. 2: Absolute error of an estimated write-pulse time compared to exact measurement based on data values.

3 Case Studies

3.1 MLC Simulation Accuracy

Simple models for measuring performance in non-volatile memory can be designed by encapsulating the non-volatile memory timings as DRAM timings —either through tRP or tWR. These timing modifications become more complicated with MLC memories. Worst case timings for MLC can be 7-8 times larger than SLC timings, due to a program and verify approach used.

We demonstrate the possible inaccuracy when simulating exact and estimated MLC timings for a 2-bit MLC memory. The estimated timing value assumes 3 SET pulses based on the average SETs across Table 1. At runtime NVMain inspects the data being written to memory and counts the number of each bit pattern[1]. Each combination of 2-bit values is given a programming time using a Gaussian distribution based on the mean in Table 1 and standard deviation of 1. Our write model assumes enough write drivers for the entire word. Thus, write time is the maximum programming time of any cell in the entire word.

TABLE 1: Simulated MLC Timing Parameters

Value	Mean SETs	Value	Mean SETs
00	0	01	7
10	5	11	1

The results in Figure 2 show the absolute error of IPC values between the estimated and exact approaches in the first column. `gobmk` and `zeusmp` performance is *underestimated*, indicated by black. The remaining benchmarks are *overestimated*, indicated by gray. We ran four simulations of the same approach with maximum set pulse deviation ranging from 0 to 3. The maximum error compared to 0 deviation was 7.5% at 3 standard deviations. This shows that even allowing the write pulse count to deviate by as much as 50%, the results are much more disparate than the estimated technique.

The second column shows the average number of writes cycles and the estimated value. Comparing with the first column, we can see a strong correlation between misestimate of write cycles and IPC error. Inaccuracy is directly related to the distribution of 2-bit data values being written to the MLC cells. The distribution is

1. The counting technique used is highly optimized and utilizes simple bitwise operations to perform each count.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LCA.2015.2402435, IEEE Computer Architecture Letters

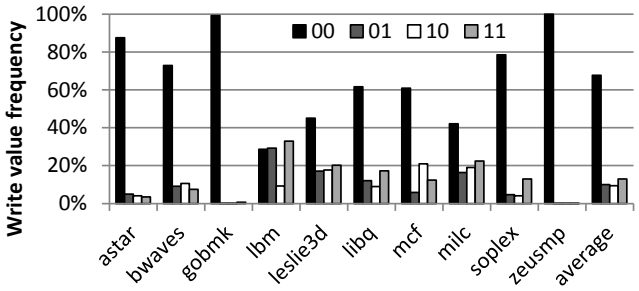

Fig. 3: Frequency of 2-bit data values written to MLC cells for various SPEC2006 benchmarks.

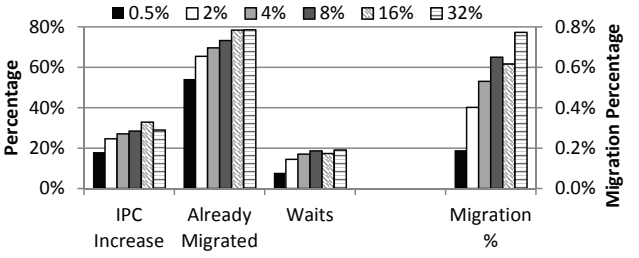

Fig. 4: IPC results and migration statistics for a hybrid memory.

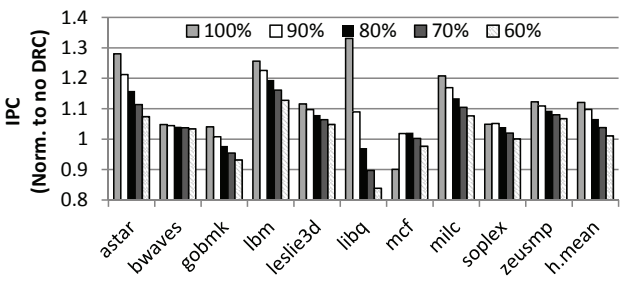

Fig. 5: IPC results of DRAM Cache with varying prediction accuracy.

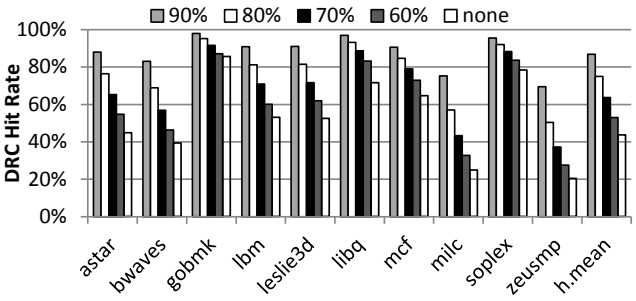

Fig. 6: Predictive DRAM Cache hit rate at various accuracies.

heavily skewed in most benchmarks, and there is no clear pattern as shown in Figure 3. Even benchmarks with as little as 10% combined 01/10 values begin to overestimate IPC. These results strongly motivate the need of data-aware simulation, particularly when utilizing MLC cells.

3.2 Hybrid Memory System

We implemented a hybrid memory system similar to [3] which uses a hardware-based page migrator to demonstrate the usefulness of advanced memory controllers and memory hooks. This simulated system uses a memory controller which employs a lookup table to find the destination of migrated pages and launch migration accordingly. Secondly, a memory hook is used to snoop on requests arriving at the memory controllers. This implementation uses a biased coin to decide whether to migrate a page or not for simplicity.

Each time a request is issued, there is a small chance it will be migrated to another memory channel designated as "fast" memory. In theory, if a page is truly a "hot page", it will eventually be migrated to fast memory. In this example, DRAM is used as fast memory and the remaining channels use NVM as slow memory. Migrations are done by using a single swap buffer. Our hook object injects *real* read requests to fill this buffer[2]. Once the buffer is filled up, *real* write requests are issued to the memory controllers. After writes complete, the buffer is now free for another swap to take place.

The results in Figure 4 show a sweep of migration probabilities ranging from 0.5% to 32%. Each result is the mean value of ten SPEC2006 benchmarks. Using this simple technique, the IPC can be improved between 19% and 33%. The last column shows the percentage of migrations using the right y-axis. Since there is only one swap buffer to exchange requests, the actual number of requests that result in a migration (denoted "Migration %") does not reach 1% and thus the increase in memory traffic due to migrations is minimized.

Address translators and hooks may also load new statistics and print out results if necessary. Using this we collect the number of requests which could not be migrated because they are already in fast memory. This number increases as the migration probability becomes more aggressive, since the data being used by the application becomes completely migrated to the fast memory. In Figure 4, "Waits" shows the percentage of migrations that could not start because another migration is in progress. As expected, it increases as the migration probability increases and saturates around 19%.

3.3 DRAM Cache

Another example of a more complex memory system is the one utilizing a giga-scale DRAM Cache. Our implementation of such system involves a memory controller which creates a new root system that acts as backing memory. Normally all requests go to DRAM Cache first and are routed to backing memory on a miss. The address translator is now an access predictor [10] which determines if data reside in the DRAM Cache. If the predictor guesses the data is not in the DRAM Cache, the request bypasses DRAM Cache and is sent directly to backing memory[3].

In this work we sweep over a range of predictor accuracy from 60%-100%, including the case with no predictor to evaluate the impact of prediction accuracy. The results are shown in Figure 5. The baseline "no DRC" stands for a standard off-chip memory system. Interestingly, `gobmk`, `mcf`, `soplex`, and most notably `libquantum` even drop below the performance of a predictionless scheme if accuracy is too low. Therefore, we observe that the prediction accuracy must be pretty high to ensure that the predictor can really help performance. Figure 6 also reveals that hit rate is not the major factor in this performance loss. Instead, the problem occurs because DRAM Cache does not see the requests that directly go to backing memory[4]. Furthermore, as prediction accuracy decreases, more mispredictions cause non-trivial memory traffics to both DRAM Cache and backing memory, which in turn increases average access times.

2. Requests may be served directly from the buffer during migration.

3. This bypass is not allowed for dirty data in the DRAM Cache.

4. This is specific to our implementation and is easily modified.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LCA.2015.2402435, IEEE Computer Architecture Letters

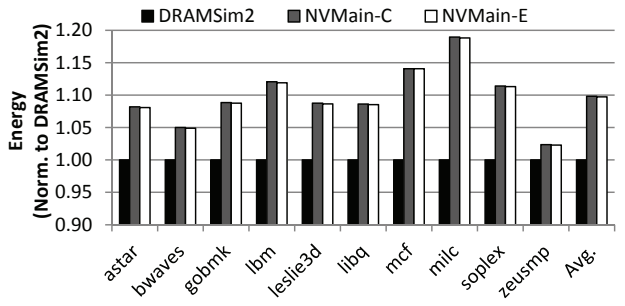

Fig. 7: Calculated memory subsystem power of NVMain normalized to DRAMSim2.

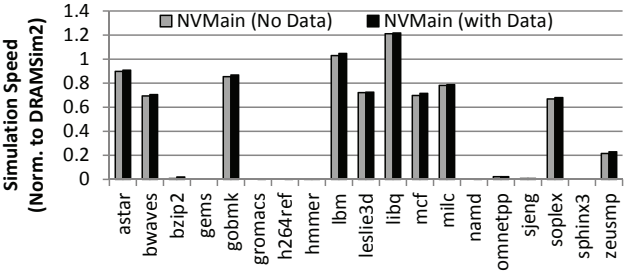

Fig. 8: Percentage of simulation time spent in memory subsystem.

4 Simulator Verification

In order to verify the correctness of NVMain 2.0, we show the validation results in this section. First, the verification for DDR3-like timings was performed against Verilog models available from Micron [7]. Then, we use the DRAMPower2 simulator [2] to obtain energy number and compare it with NVMain 2.0 in both Current-mode and Energy-mode. Next, the modeling of endurance data encoders and real MLC behavior requires proper data verification. Finally, we show our simulator's speed.

4.1 Timing Verification

We re-verified the timing model in NVMain 2.0 by using the Verilog model [7] as in NVMain 1.0. A simple DRAM-only system is simulated with an FR-FCFS memory controller. Hooks are used to output requests arriving at each rank in Verilog format. We use ModelSim to run the Verilog model and checked the transcript for violations. All memory behaviors matched and no timing violation was found. The instructions for re-running verification are included with the NVMain 2.0 distribution.

4.2 Energy Verification

We compare Current-mode against DRAMSim2 by running them in gem5. NVMain is configured as closely as possible to DRAMSim2 where DRAM timings and an FCFS memory controller are in use. Results of the comparison are shown in Figure 7 (NVMain-C series). Discrepancies in the comparison are mostly due to differences in the implementation of memory controller scheduling. NVMain schedules requests more frequently, resulting in a higher activity factor.

Energy-mode is verified against DRAMPower2 [2] with rank-level traces generated by hooks. We use DRAMPower2 to estimate energy values for each memory operation and compare Energy-mode and Current-mode results, respectively. Results of this verification are shown in the NVMain-E series.

4.3 Data Verification

To verify the correctness of data manipulation, we design a system with no caches and develop a small micro-benchmark that writes known values to large memory arrays repeatedly. We then check whether data is correctly read from simulator memory or trace file by comparing the observed data with the known values.

4.4 Simulation Speed

NVMain's simulation speed is measured by profiling gem5 for a fixed number of memory requests. The percentage of time spent in the memory subsystem is shown in Figure 8, normalized to DRAMSim2. We also evaluate an MLC memory with the exact approach depicted in Section 3.1. The results show that NVMain can outperform DRAMSim2 in most tests, even when data operation is strictly modeled. Note that the memory scheduling in NVMain has not been fully optimized for the simulation speed. We expect to further improve the speed in the near future.

5 Conclusion

We have introduced NVMain 2.0, a main memory simulator for DRAM and upcoming non-volatile memories. The support for NVM design, flexibility, and fast development are outlined as key values of NVMain 2.0. We have discussed some of the new features differentiated versus existing simulators. We also showed simple studies to demonstrate the capability and need for such a simulator. Finally, we described the verification process to ensure the correctness of NVMain 2.0. We believe that NVMain 2.0 is an extremely valuable contribution to the computer architecture community. It will help expedite the related researches in a multitude of directions in the main memory area.

References

[1] "JEDEC Standard: DDR4 SDRAM," http://www.jedec.org/sites/default/files/docs/JESD79-4.pdf.

[2] K. Chandrasekar, B. Akesson, and K. Goossens, "Improved power modeling of ddr sdrams," in *Digital System Design (DSD), 2011 14th Euromicro Conference on*, 2011, pp. 99–108.

[3] X. Dong, Y. Xie, N. Muralimanohar, and N. P. Jouppi, "Simple but effective heterogeneous main memory with on-chip memory controller support," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1–11. [Online]. Available: http://dx.doi.org/10.1109/SC.2010.50

[4] X. Dong, C. Xu, Y. Xie, and N. Jouppi, "Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 31, no. 7, pp. 994–1007, 2012.

[5] A. Hansson, N. Agarwal, A. Kolli, T. Wenisch, and A. Udipi, "Simulating dram controllers for future system architecture exploration," in *Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on*, March 2014, pp. 201–210.

[6] Y. Kim, V. Seshadri, D. Lee, J. Liu, and O. Mutlu, "A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM," in *ISCA'39*, Jun. 2012, pp. 368–379.

[7] Micron, "Ddr3 sdram verilog model," 2013.

[8] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "Cacti 6.0," HP Laboratories, Tech. Rep. HPL-2009-85, 2009.

[9] M. Poremba and Y. Xie, "Nvmain: An architectural-level main memory simulator for emerging non-volatile memories," in *VLSI (ISVLSI), 2012 IEEE Computer Society Annual Symposium on*, Aug 2012, pp. 392–397.

[10] M. Qureshi and G. Loh, "Fundamental latency trade-off in architecting dram caches: Outperforming impractical sram-tags with a simple and practical design," in *Microarchitecture (MICRO), 2012 45th Annual IEEE/ACM International Symposium on*, Dec 2012, pp. 235–246.

[11] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "DRAMSim2: A Cycle Accurate Memory System Simulator," *Computer Architecture Letters*, vol. 10, no. 1, pp. 16–19, Jan.–Jun. 2011.