# NMTSim: Transaction-Command based Simulator for New Memory Technology Devices

Peng Gu*, Benjamin S. Lim†, Wenqin Huangfu*, Krishan T. Malladi†, Andrew Chang†, and Yuan Xie*
*University of California, Santa Barbara, USA
†Samsung Semiconductor, San Jose, USA

**Abstract**—To mitigate the impact of non-deterministic media access latencies in new memory technology devices, a recently proposed NVDIMM (Non-Volatile Dual In-line Memory Module) standard, NVDIMM-P [1], uses novel out-of-order transaction commands. The previous DRAM simulators [2], [3], [4] are unable to support this transaction protocol due to deterministic DDR timing. Also, existing NVDIMM [5] simulators are customized for NAND flash memory, which are not generally applicable to emerging NVM (Non-Volatile Memory). In this paper, we present NMTSim, a transaction-command based and cycle accurate simulator for new memory technology devices. Strictly conforming to NVDIMM-P standard, NMTSim introduces a new memory controller with transaction handling and command issuing capabilities. To enable simulation for emerging NVM using DDR4 standard, we propose some new NVM timing parameters and incorporated them into DRAMSim2 [2]. Furthermore, DRAMSim2 is augmented with transaction handling and command scheduling logic to be the backend for the media controller. In addition, NMTSim incorporates an optimized transaction command issuing policy and an early notification mode to optimize access latency. We verify NMTSim using Intel Optane [6], and characterize its performance using synthetic benchmarks with different read / write ratios.

**Index Terms**—Emerging technologies, Primary memory, Simulation

## 1 Introduction

NVDIMM (Non-Volatile Dual In-line Memory Module) is a viable approach to realize persistent memory, which can bridge the performance and capacity gap between memory and storage hierarchies. Since NVDIMM supports various new memory technology devices in the same channel, it is necessary to tolerate the non-deterministic access latency of different media types. To enable this, a recently proposed NVDIMM-P [1] protocol utilizes novel out-of-order transaction commands. Unlike DDR timing which assumes synchronous media responses, NVDIMM-P allows asynchronous media activities through a transaction handshaking mechanism. For example, after receiving an $XREAD$ (transaction READ) command, the media controller can respond to the host ($RD\_RDY$) after a non-deterministic time. The host can acknowledge the $RD\_RDY$ signal by issuing a $SEND$ command also after a non-deterministic time.

However, existing memory simulators are either unable to support the novel transaction features in NVDIMM-P, or confined to a limited media scope. Previous DRAM simulators, including DRAMSim2 [2], NVMain2 [3], and Ramulator [4], only employ deterministic DDR timing protocols. Significant modification efforts are required to add handshaking and transaction handling logic in the complex scheduling unit of memory controller. Also, the passive memory module needs to add extensive new functionalities to become a media controller that can independently process host-issued commands. Previous NVDIMM simulators (e.g., FlashDIMMSim [5]) are customized for traditional flash media with block-granularity. While emerging NVMs have demonstrated better performance and byte-addressability, it is more promising to explore them as memory media for NVDIMM.

This paper proposes NMTSim, a transaction-based (NVDIMM-P compatible) and cycle accurate memory simulator for new memory technology devices. To support transaction semantics, NMTSim includes a new memory controller with queuing structures, transaction handling logic and a command issuing unit. For the media controller, NMTSim uses a modified DRAMSim2 [2] simulator as the DRAM/NVM back-end, adapts DDR4 timing parameters to simulate emerging NVM devices, and adds a command scheduling unit and corresponding transaction functionalities. We validate NMTSim's simulation accuracy using real hardware measurements from Intel Optane memory [6], which uses a proprietary transaction command protocol for 3D Xpoint memory.

NMTSim also incorporates two architectural level optimizations to reduce latency overhead introduced by transaction commands. The first optimization grants $SEND$ command higher priority than $XREAD$ command, and can significantly reduce access latency under high host request bandwidth. The second optimization enables early host notification from media controller, and can save $tRL$ latency for all host request bandwidth. After applying these optimizations, we thoroughly compare the performance of NVDIMM-P and DDR4 using DRAM and NVM with synthetic benchmarks under different read/write ratios. We summarize the main contributions of NMTSim as follows:

1) We propose NMTSim, a transaction-command based and cycle accurate simulator for new memory technologies. We show the simulation framework of NMTSim and verify it using Intel Optane memory [6].
2) We incorporate a command issue optimization and an early notification functionality for transaction-command in NMTSim, and demonstrate the latency improvements of these two schemes.
3) We evaluate NMTSim using synthetic benchmarks. Evaluation results on both DRAM and NVM devices show slight latency overhead of NVDIMM-P compared with DDR4.

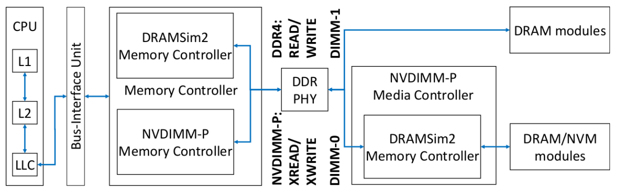

Figure 1. **NMTSim's high-level overview.**

## 2 NMTSim Design

We first introduce the high-level overview of NMTSim in Sec. 2.1. Then, in Sec. 2.2 we explain the detailed simulator com-

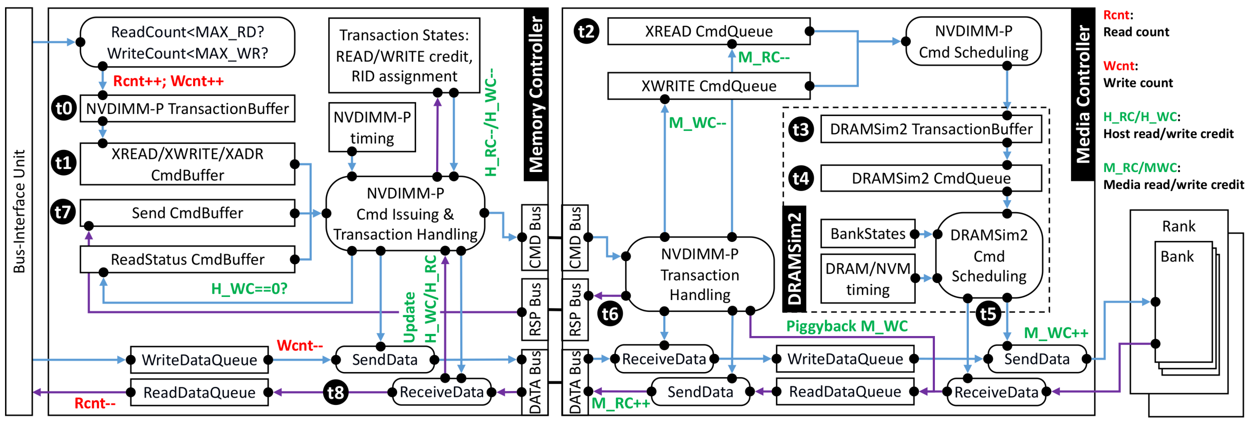

Figure 2. **NMTSim's block diagram and data/control paths. The circled numbers represent time stamps in a XREAD command in Fig. 3**

ponents, data paths, and control paths to support transaction commands. Last, Sec. 2.3 discusses the new timing parameters to support emerging NVM.

## 2.1 High-level Overview

NMTSim consists of a front-end processor interface, a memory controller, an interface bus, and a media controller with DRAM/NVM modules as shown in Fig. 1. In trace-based simulation mode, the front-end processor interface is compatible with DRAMSim2, and can be easily modified to support any new processors. The memory controller supports two memory protocols: DDR4 protocol which is implemented by a modified DRAMSim2 memory controller, and NVDIMM-P protocol which is realized by the proposed NVDIMM-P memory controller. The two protocols coexist on the same memory channel, on which DDR4 uses normal READ/WRITE commands for DIMM-0, and NVDIMM-P uses XREAD/XWRITE commands for DIMM-1. For NVDIMM-P protocol, an additional media controller is included to support transaction semantics. Other NVDIMM-P commands such as PWRITE and FLUSH are also supported, but are out of the scope of this paper and will be studied in future work. In full-system simulation mode, NMTSim can be integrated into existing architecture simulators (e.g., GEM5) to receive and respond host memory requests.

## 2.2 Support Transaction Commands Simulation

To support transaction commands, a new memory controller with NVDIMM-P queuing structures, command issuing logic, and transaction handling logic are proposed as shown in Fig. 2. The NVDIMM-P transaction buffer will store accepted host memory requests, and the command buffer will store translated NVDIMM-P commands. The NVDIMM-P command issuing unit will decide which command can be sent to the media controller according to transaction states and command priority, and the transaction handling logic will arbitrate command and data traffic to avoid conflicts on the bus according to NVDIMM-P timing. An XREAD (XWRITE) command can only be issued if there is available read (write) credit, and the corresponding credit will decrement by one after a command is sent. The read credit will increment by one if memory controller receives a returned read data packet. Released write credit will be piggybacked through returned read data packet. However, if there is no enough write credit, memory controller can compose a READ_STATUS command and explicitly ask the media controller for more write credit. After receiving a RD_RDY signal from RSP bus, a SEND command will be prepared and issued by the command issuing logic.

From the media controller side, the received commands will first be interpreted and added to command queues for further scheduling. At the same time, the corresponding credit will decrement by one. Since NVDIMM-P supports out-of-order transactions, the NVDIMM-P command scheduling logic will select a command without data hazard for execution. To optimize read latency, an XREAD command has higher scheduling priorities than an XWRITE command. The write credit will increment after the XWRITE command is consumed by DDRAMSim2. After an XREAD command finishes execution by DRAMSim2, the NVDIMM-P transaction handling logic will inform the memory controller by signaling the RSP bus. Then, the media controller will return read data packets and increment read credit after receiving SEND command.

By default, the NVDIMM-P protocol enables $256$ maximum read/write transaction commands. However, we find that for memory setting with small bank number, this large command count will cause unnecessary command queuing delay. To reduce this delay, we add maximum read/write count to constrain host issued requests. The maximum read/write count is set to match the total bank number per rank, assuming we use a per-rank queuing structure in DRAMSim2.

Table 1
**Latency terminologies for an XREAD command.**

| Parameter | Definition |
|---|---|
| $T1$ | Memory Controller Transaction Buffer Latency |
| $T2$ | Memory Controller Command Buffer Latency |
| $T3$ | Media Controller Command Queue Latency |
| $T4$ | DRAMSim2 Transaction Buffer Latency |
| $T5$ | DRAMSim2 Command Queue Latency |
| $T6$ | Media Bank Access Latency |
| $T7$ | Memory Controller Response Latency |
| $T8$ | Memory Controller SEND Latency |
| $T_{add}$ | Cache Access Handling and Bus-interface Unit Latency |

## 2.3 Support Emerging NVM Timing Simulation

In order to support emerging NVM timing simulation using DRAM compatible timing parameters, we propose the following changes. First, since NVM is non-volatile in nature, for NVM mode DRAM REFRESH command is disabled and all corresponding timing parameters are set to zero. Second, we change the $t_{RCD}$ to $t_{RCD\_R}$ for read and $t_{RCD\_W}$ for write considering the asymmetric read/write behavior for NVM. Since NVM write does not require row activation, we set $t_{RCD\_W} = 0$ for a full page write. Similarly, we change the $t_{RP}$ to $t_{RP\_R}$ for read and $t_{RP\_W}$ for write. Since NVM read does not require row precharge, we set $t_{RP\_R} = 0$. Third, to account for partial page write overhead, we add a new timing parameter $t_{RMW}$ (Read-Modify-Write). If write request number to the same row is smaller than the page size, then an additional $t_{RMW}$ latency overhead is induced to read the unmodified portion of the page.

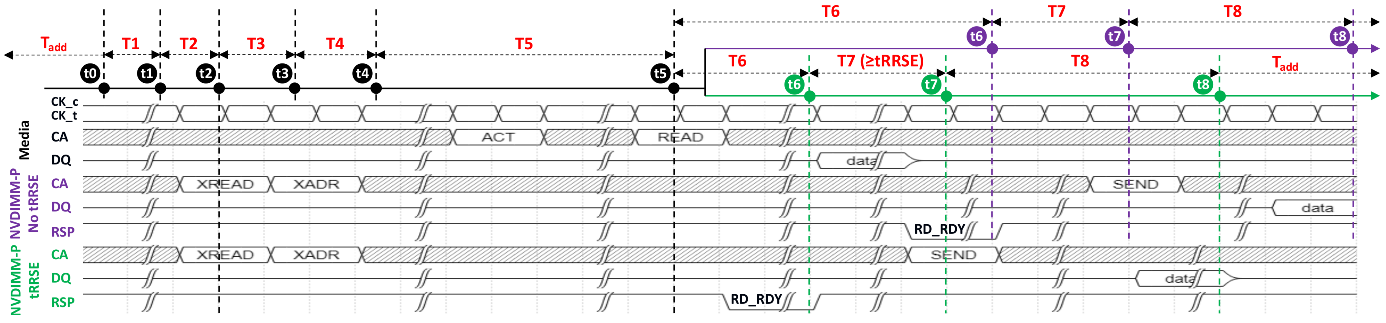

Figure 3. **XREAD latency breakdown and illustration of early notification mode. Timing terminologies are detailed in Table. 1.**

Table 2
**NMTSim Architecture and Timing Configuration.**

| Parameter | NVM | DRAM | Parameter | NVM | DRAM |
|---|---|---|---|---|---|
| Channel | 1 | 1 | $t_{CK}$ | 0.75ns (2666MHz) | |
| Rank | 1 | 2 | $t_{RCD\_R}$ | 192ns | 14.3ns |
| BankGroup | 2 | 4 | $t_{RCD\_W}$ | 0ns | 14.3ns |
| Bank | 4 | 4 | $t_{RP\_R}$ | 0ns | 13.5ns |
| Row | $2^{26}$ | $2^{16}$ | $t_{RP\_W}$ | 489ns | 13.5ns |
| Column | $2^{5}$ | $2^{10}$ | $t_{RMW}$ | $t_{RCD\_R}$+$t_{RP\_W}$ | - |
| DeviceWidth | 8 | 8 | $t_{REF}$ | - | 7800ns |
| Capacity | 128GB | 16GB | $t_{RFC}$ | - | 260.3ns |
| CmdQueueStruct | Per Rank | | CmdQueueDepth | 16 | 32 |
| RowBufferPolicy | Close-Page | | Scheduling | Rank-Bank Round Robin | |

The $t_{RCD\_R}$ and $t_{RP\_W}$ parameters should be derived based on actual NVM measurement results. First, a sequential benchmark should be used to stress the actual NVM hardware, and the maximum achievable read and write bandwidth can be discovered. Then, using the same benchmark and assuming the same configuration as the baseline NVM hardware, we can sweep $t_{RCD\_R}/t_{RP\_W}$ parameter space to acquire the maximum achievable read/write bandwidth curve. In the end, we can select the $t_{RCD\_R}/t_{RP\_W}$ value if the corresponding read/write bandwidth matches hardware measurements.

## 3 Transaction Command Optimization

Before introducing the optimizations, we illustrate the latency breakdown for an XREAD command in Fig. 3 and related timing terminologies in Table. 1. Since the load-to-use latency measures the request-to-service interval from a host CPU, we also include a constant additional latency [7] to count for host cache access handling and bus-interface unit delay, which are extracted from the CPU baseline in Sec. 4.1. Specifically, the $T_{add}$ represents the latency between the memory controller and the host load-store interface. All analysis in this section uses synthetic random benchmarks ($64Byte$ access granularity) based on the hardware configuration of Table. 2.

### 3.1 Command Issue Optimization

The memory controller command issuing logic needs to decide the priority between SEND and XREAD for optimal performance when the command bus is congested. The first policy ($p1$) grants XREAD with higher priority to reduce memory controller side queuing latency and delays SEND. In contrast, the second policy ($p2$) gives SEND the higher priority which shortens the latency of already-issued XREAD but blocks new XREAD. We plot the latency-bandwidth graph of the two policies for DRAM and NVM in Fig. 4 (a-1)/(b-1), respectively. We can observe that for DRAM, $p2$ can significantly reduce access latency compared with $p1$ for high memory bandwidth cases. Further latency breakdown in Fig. 4 (a-2) finds that the latency increase in $p1$ is mainly incurred by $T7$, which is the memory controller response queuing latency for SEND. This validates the benefits of $p2$. However, we also observe that $p2$ has larger $T5$ than $p1$, which corresponds to DRAMSim2 command queuing latency. Since we add maximum read count constraints in the simulator, earlier completion of XREAD ($p2$) will encourage the memory controller to accept and send more XREAD to media controller, which in turn increases the queuing delay ($T5$). For NVM, these two policies show little difference in terms of latency. This is because NVM has much lower sustained bandwidth compared with DRAM, so the command bus will not be congested to issue SEND and XREAD commands, as shown in Fig. 4 (b-2).

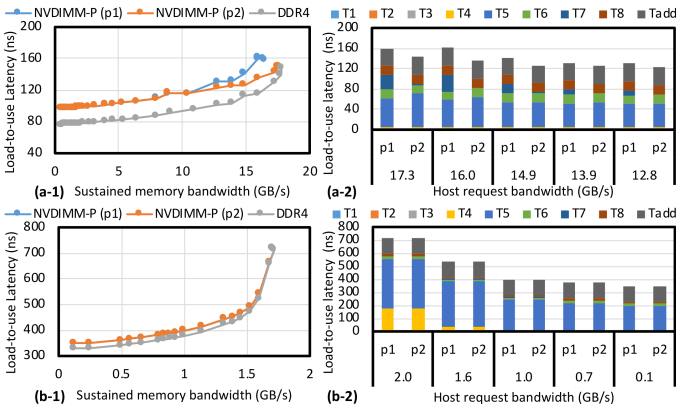

Figure 4. **The comparison of two command issuing policies ($p1$,$p2$). (a) DRAM media. (b) NVM media. Left: bandwidth-latency graph. Right: latency breakdown.**

### 3.2 Early Notification

NVDIMM-P supports early host notification to reduce response latency. Fig. 3 illustrates the timing differences of disabling (No $t_{RRSE}$) and enabling ($t_{RRSE}$) early notification. When early notification is not supported, media controller needs to wait until the media returns read data to inform host ($RD\_RDY$). When early notification is enabled, media controller can inform the host ($RD\_RDY$) in advance, and memory controller needs to wait for a certain time interval ($t_{RRSE}$) after receiving $RD\_RDY$ to issue $SEND$. The choice of early notification timing depends on implementation of memory controller and the design of media controller. Here, for optimistic estimation, we assume host response immediately after receiving early notification and media controller signals $RD\_RDY$ at the same time it issues $READ$ to the media. After including the optimization in Sec. 3.1, we plot the bandwidth-latency graph of disabling and enabling early notification for DRAM and NVM in Fig. 5 (a-1)/(b-1). We find that for low to medium memory bandwidth, enabling early notification can greatly reduce latency ($\sim 15ns$) for both NVM and DRAM. However, for high memory bandwidth, the latency gap closes between

$e1$ and $e2$. We further investigate the latency breakdown in Fig. 5 (a-2)/(b-2). First, we observe the latency reduction of $e2$ is mainly contributed by reducing $T6$, since the media bank access latency is overlapped by $t_{SEND}$ in early notification case. Second, we observe that as request bandwidth increases, $T5$ increases rapidly for $e2$ until the latency saturates. Under the constraints of maximum read count, $e1$ require less XREAD to saturates the latency, so the extra XREAD for $e2$ will stack in DRAMSim2's command queue and add to the queuing latency.

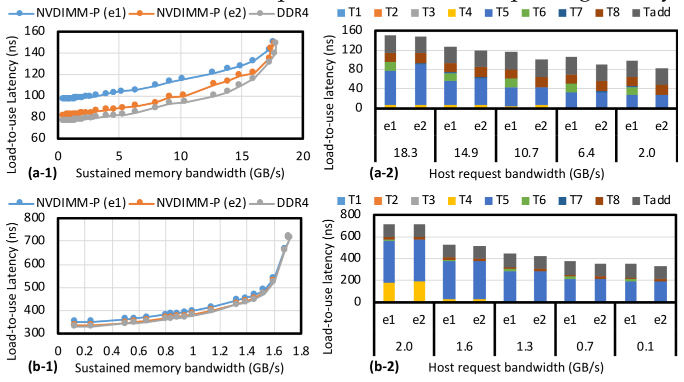

Figure 5. **The comparison between disabling($e1$)/enabling($e2$) early notification. (a) DRAM media. (b) NVM media. Left: bandwidth-latency graph. Right: latency breakdown.**

## 4 Validation and Evaluation

### 4.1 Validation

For the baseline hardware configuration [6], we use an Intel Xeon Platinum 8280L processor with a single memory channel, which contains a 128GB DCPMM Intel Optane memory (App Direct mode) and a 16GB RDIMM Samsung (M393A2K43BB1) DRAM. The baseline uses DDR-T, which is a proprietary transaction protocol of Intel. In comparison, NMTSim assumes NVDIMM-P transaction protocol for both DRAM and NVM media and uses the configuration as detailed in Table. 2. The optimizations from Sec. 3.1 and Sec. 3.2 are also incorporated in NMTSim. We extract the additional latency ($T_{add}$) of $\sim 35ns$ for RDIMM+DRAM device and $120ns$ for DCPMM+Optane device. For NVM simulation, to match the maximum read bandwidth ($8.3GB/s$) and maximum write bandwidth ($2.2GB/s$), $t_{RP\_W}$ is set to 192ns and $t_{RP\_W}$ is set to 489ns. We plot the bandwidth-latency graph and use all_read and 2reads_1write random access benchmarks (64Byte granularity) to validate the results of NMTSim with baseline hardware measurements in Fig. 6. The validation results show that on average NMTSim has 2.8% and 3.4% latency error compared with the baseline.

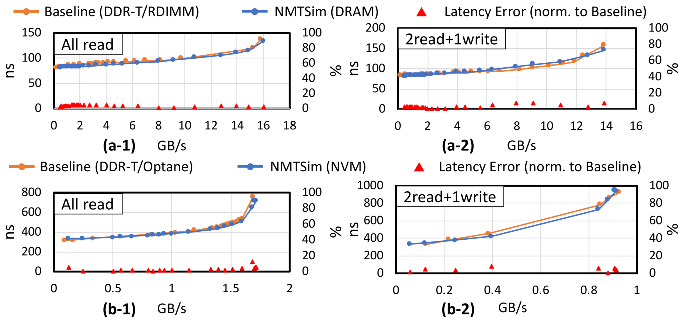

Figure 6. **Validation of NMTSim with Intel Optane.**

### 4.2 Evaluation on Synthetic Benchmarks

After applying the optimizations from Sec. 3, we use random access benchmarks (64Byte granularity) with various read/write ratios ($R = 0.8/0.5/0.1$) to characterize the performance of NMTSim using combinations of NVDIMM-P and DDR4 protocols with different media types, as shown in Fig. 7. For both DRAM and NVM media, NVDIMM-P adds additional transaction latency with respect to DDR4. For DRAM, the additional latency varies little as memory bandwidth changes. However, for NVM, the additional latency tends to increase as read/write ratio is low. This is because NVM's write performance is much worse than its read performance. As read/write ratio goes low, NVM writes will interfere read request more significantly compared with DRAM. In addition, we observe that NVM's bandwidth-latency curve bends more obvious than DRAM's as read/write ratio becomes lower, which is also explained by NVM's asymmetric read/write performance.

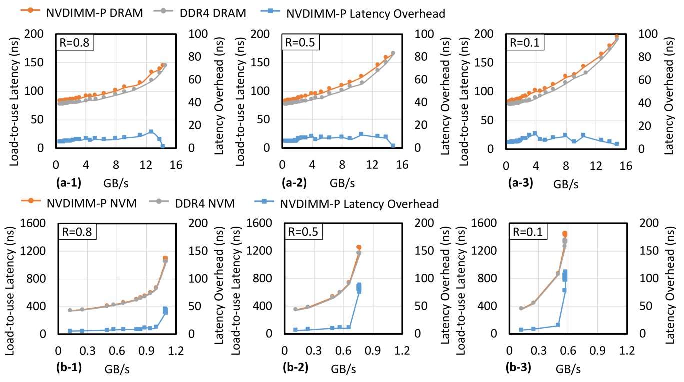

Figure 7. **Characterize NMTSim using different Read/Write ratio ($R$). (a) DRAM media. (b) NVM media.**

## 5 Conclusion

In this paper, we present NMTSim, a transaction-command based and cycle accurate simulator for new memory technology. NMTSim introduces a new memory controller with transaction handling and command issuing logic. To enable simulation for emerging NVM using DDR4 standard, we propose some new NVM timing parameters and incorporated them into DRAMSim2 [2]. Furthermore, DRAMSim2 is augmented with transaction handling and command scheduling logic to be the backend for the media controller. In addition, NMTSim incorporates an optimized transaction command issuing policy and an early notification mode to optimize access latency. We verify NMTSim using Intel Optane memory [6], and characterize its performance using synthetic benchmarks with different read/write ratio.

## References

[1] B. Gervasi, D. Designs, and J. Hinkle, "Overcoming system memory challenges with persistent memory and nvdimm-p," in *JEDEC Server Forum*, vol. 2017, 2017.

[2] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "Dramsim2: A cycle accurate memory system simulator," *IEEE computer architecture letters*, vol. 10, no. 1, pp. 16–19, 2011.

[3] M. Poremba, T. Zhang, and Y. Xie, "Nvmain 2.0: A user-friendly memory simulator to model (non-) volatile memory systems," *IEEE Computer Architecture Letters*, vol. 14, no. 2, pp. 140–143, 2015.

[4] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A fast and extensible dram simulator," *IEEE Computer architecture letters*, vol. 15, no. 1, pp. 45–49, 2015.

[5] P. Enns, *FlashDIMMSim: A reasonably accurate flash DIMM simulator*. [Online]. Available: https://github.com/jimstevens2001/NVDIMMSim

[6] B. Tristian, L. Travis, and C. Jamie, "Analyzing the performance of intel optane dc persistent memory in app direct mode in lenovo thinksystem servers," *Lenovo Press*, 2019.

[7] R. S. Verdejo, K. Asifuzzaman, M. Radulovic, P. Radojković, E. Ayguadé, and B. Jacob, "Main memory latency simulation: the missing link," in *Proceedings of the International Symposium on Memory Systems*, 2018, pp. 107–116.