Characterizing and Modeling Non-Volatile Memory Systems

Zixuan Wang Xiao Liu Jian Yang[*] Theodore Michailidis Steven Swanson Jishen Zhao
University of California, San Diego

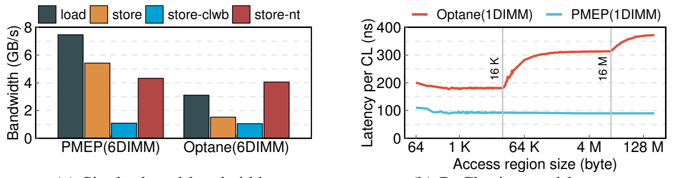
***Abstract*—Scalable server-grade non-volatile RAM (NVRAM) DIMMs became commercially available with the release of Intel's Optane DIMM. Recent studies on Optane DIMM systems unveil discrepant performance characteristics, compared to what many researchers assumed before the product release. Most of these studies focus on system software design and performance analysis. To thoroughly analyze the source of this discrepancy and facilitate real-NVRAM-aware architecture design, we propose a framework that characterizes and models Optane DIMM's microarchitecture. Our framework consists of a Low-level profilEr for Non-volatile memory Systems (LENS) and a Validated cycle-Accurate NVRAM Simulator (VANS). LENS allows us to comprehensively analyze the performance attributes and reverse engineer NVRAM microarchitectures. Based on LENS characterization, we develop VANS, which models the sophisticated microarchitecture design of Optane DIMM, and is validated by comparing with the detailed performance characteristics of Optane-DIMM-attached Intel servers. VANS adopts a modular design that can be easily modified to extend to other NVRAM architecture designs; it can also be attached to full-system simulators, such as gem5[1]. By using LENS and VANS, we develop two architectural optimizations on top of Optane DIMM, Lazy Cache and Pre-translation, which significantly improve cloud workload performance.**

***Index Terms*—nonvolatile memory, memory systems, simulation**

I. Introduction

Non-volatile RAMs (NVRAMs) [6], [34], [61] have been envisioned as a new tier of memory in server systems, offering comparable performance to DRAM with the durability property of storage devices [28], [33], [53], [54], [70]. Seeing the great value, a large body of prior studies investigated how to exploit NVRAM to benefit future computer systems. Yet only recently has the first server-grade NVRAM DIMM product come to market, namely Intel Optane DC Persistent Memory (aka. Optane DIMM) [23].

The latest characterization studies [28], [41], [42], [58], [59], [66] exposed substantial discrepant performance characteristics compared to what we thought before the real products were released. For instance, Figure 1 compares results of the widely-used Persistent Memory Emulation Platform (PMEP) [11] and our Optane DIMM server measurement (denoted as *Optane*) on single-thread read and write bandwidth (Figure 1a) and latency per cache line (CL) access (Figure 1b) with a pointer chasing microbenchmark. The microbenchmark randomly accesses a contiguous data region with fixed 64B



(a) Single thread bandwidth. (b) PtrChasing read latency.

Fig. 1. Comparison between PMEP [11] emulation and Optane DIMM-based real machine measurement. *Store-clwb* are writes followed by `clwb` cache write-back instructions [20]; *store-nt* are non-temporal writes [20].

objects. By varying the data region size from 64B to 256MB, we observe a clear inconsistency between the emulation and real-system results (discussed in detail in Section II-B). Other previous NVRAM emulators and simulators also generate different performance characteristics compared to real-machine results, as discussed in Section II and Section III. As such, previous NVRAM emulation and simulation tools are insufficient to model modern real NVRAM systems.

Moreover, Intel's announcement [23] mentions that the Optane DIMM adopts an on-DIMM buffer structure, although without further details. The *Optane* curve in Figure 1b also shows buffer effects at the sizes of 16KB and 16MB, respectively. Both indicate that Optane DIMM adopts a much more complex microarchitecture design than conventional DRAM DIMMs. However, most of the existing Optane DIMM studies focused on systems-level performance profiling and system software design [8], [29], [42], [58], [59], without detailed investigation on microarchitecture. Furthermore, the existing architecture- and system-level performance profiling tools are designed for DRAM-based memories, inefficient in investigating the architectural structure details of real NVRAM systems as compared in Table I.

Our goal in this paper is to facilitate research on architecture and systems design of NVRAM-based memory systems by (1) investigating detailed performance and microarchitecture characteristics of Optane DIMM, (2) designing a generic architecture-level performance profiling and reverse engineering framework, which identifies the key architectural and performance characteristics of NVRAM systems, and (3) developing a memory simulator that models the microarchitecture designs of modern NVRAM DIMMs. To achieve our goal, we make the following technical contributions:

- **Profiling and reverse engineering framework.** We design a Low-level profilEr for Non-volatile memory Systems (LENS), which consists of a set of profiling tools and microbenchmarks to analyze the architectural and performance characteristics of NVRAM-based memory systems. LENS

[*]Now at Google.

[1]LENS and VANS are open-sourced at GitHub [1].

is designed as a Linux kernel module. It exposes detailed architectural characteristics of memory systems, such as on-DIMM buffer size and hierarchy, queuing scheme, and the access granularity of each component.

- **Real-system characterization.** We employ LENS to perform a comprehensive architecture and performance analysis on Optane DIMM-based servers. Using LENS, our profiling reveals a detailed picture of the Optane DIMM microarchitecture.
- **NVRAM simulator.** We develop a Validated cycle-Accurate NVRAM Simulator (VANS), which models NVRAM-based memory system architecture designs based on our characterization. We validate VANS against extensive Optane DIMM-based real-machine profiling. VANS adopts a flexible modular design, which allows users to explore a design space with various architecture designs and extend the simulator to model other NVRAM microarchitecture designs.
- **Case studies.** We employ VANS to evaluate two architecture optimizations on Optane DIMM, Lazy cache and Pre-translation, which address Optane DIMM-based memory system performance inefficiencies in cloud workloads identified by LENS.

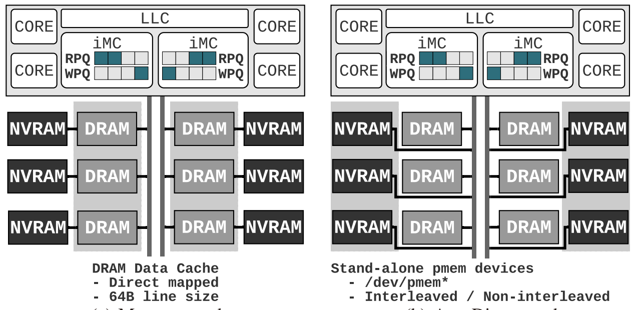
II. Background and Motivation

Overview of Our Motivation. Recent Optane DIMM profiling [28], [41], [42], [58], [59], [66] shows different performance characteristics compared to many previous NVRAM studies [45], [62]–[64], [67], [68]. However, most of the Optane DIMM studies focus on system-level performance analysis and system software design. Industrial documents indicate that the memory controllers and Optane DIMMs adopt more complex microarchitecture designs than conventional DRAM DIMM-based memory systems, but without many of the details. These challenges motivate us to explore microarchitecture design and memory system modeling based on real NVRAM products.

A. Optane DIMM-based Server Systems

Server System Organization. Figure 2 illustrates an example of Optane DIMM-based server systems. Optane DIMMs – denoted as NVRAMs – sit on the memory bus along with DRAM DIMMs. They are controlled by the processor's integrated memory controllers (iMCs). Intel's Cascade Lake is the first microarchitecture to support Optane DIMM [19]. Specifically, our server systems adopt one or two processor dies in each CPU socket. Each processor die has two iMCs, and each iMC supports three memory channels. Each channel is configurable to have zero or one Optane DIMM.

Operation Modes. Optane DIMMs operate in one of two modes: Memory Mode or App Direct Mode. The Memory Mode does not support data persistence. Each memory channel incorporates an Optane DIMM and a DRAM DIMM; the DRAM serves as a data cache for the Optane DIMM. In App Direct Mode, the Optane DIMMs are used as persistent memory, which unifies the fast access interface of memory



(a) Memory mode. (b) App Direct mode.

Fig. 2. Optane DIMM-based system configurations.

with the persistence property of storage [40], [68]. For example, programmers can directly issue load/store instructions to access in-memory data structures while NVRAM system software and hardware ensure that the data structures are recoverable in the face of system crashes and power loss [57], [63].

Known Architecture Components. Industrial documents disclose several architectural design components related to Optane DIMM [23], [37], [48]; but most of the descriptions do not provide the details as identified by our study (Figure 4). To ensure data persistence, each iMC maintains a write pending queue (WPQ) and a read pending queue (RPQ) for Optane DIMMs; the WPQs belong to the asynchronous DRAM refresh (ADR) domain [48]. CPU ensures that the data reaches the ADR domain is persisted during power outage. Optane DIMMs support CPU cache line granularity access. The current implementation adopts 3D-Xpoint chips as NVRAM media with a 256-byte access granularity [37]. In addition, each Optane DIMM maintains buffers inside the DIMM controller; sub-256-byte write accesses will trigger a read-modify-write (RMW) procedure in the buffers [23].

B. Performance Discrepancy

Many previous NVRAM studies perform experiments using NVRAM emulators and simulators, which assume that NVRAM performs like a slower DRAM. For instance, widely-used NVRAM emulators PMEP [11] and Quartz [56] model NVRAM systems by stalling the CPU for additional cycles (estimated as the cycles that the CPU would have to wait if DRAM is replaced by a slower NVRAM) and throttling bandwidth. Other studies simply inject delays in software codes to emulate NVRAM's slower writes relative to DRAM [9], [57]. However, Optane DIMM characterization [28], [41], [42], [58], [59], [66] demonstrates that real NVRAM systems have much more complex performance behaviors than conventional DRAM systems. As an example, Figure 1 shows our experiments that illustrate two observations on the performance discrepancy. First, Figure 1a shows that PMEP models both load and store bandwidth higher than the bandwidth of non-temporal stores (*store-nt*). However, with Optane DIMM, using non-temporal stores lead to a higher bandwidth than the others. Second, PMEP maintains a stable latency per cache line read, as shown in Figure 1b. If Optane DIMM were a slower DRAM, the curve would appear stable with a similar shape

TABLE I
COMPARISON OF PROFILING TOOLS.

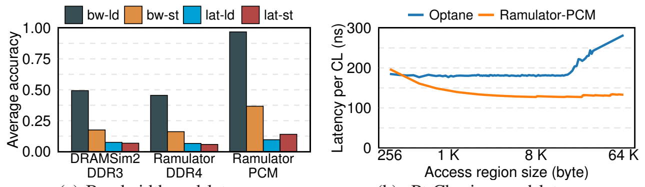
Tools	Basic			On-DIMM buffer			Long tail-latency*	
	Latency	Bandwidth	Addr mapping	Size	Granularity	Hierarchy	Frequency	Granularity
MLC [21]	✓	✓	✗	✗	✗	✗	✗	✗
perf [15]	✓	✓	✗	✗	✗	✗	✗	✗
DRAMA [43]	✓	✗	Partial	✗	✗	✗	✗	✗
LENS (our design)	✓	✓	Partial	✓	✓	✓	✓	✓

* Long write latency during repeated writes to the same memory area.

as the PMEP curve. However, Optane DIMM read latency increases with the growth of the pointer chasing region size, showing three clear segments in the curve. Both observations show that the real NVRAM system is not simply a slower DRAM DIMM-based system.

C. Inefficient NVRAM Simulation Models

Previous memory simulators, such as DRAMSim2 [46] and Ramulator [32], model memory architecture based on conventional DRAM systems. We find that these simulation models fail to accurately match the performance behaviors of Optane DIMM, as demonstrated in Figure 3. In Figure 3a, we compare the load/store bandwidth and latency of various memory simulators with Optane DIMM. The average accuracy is evaluated as the arithmetic mean of accuracies under experiments with different access sizes. Figure 3a shows a large mismatch of bandwidth and latency characteristics between simulation and our real-system profiling. Figure 3b shows the same experiment as in Figure 1b, performed by Ramulator simulation. Similar to PMEP results, the simulated read latency appears to be stable, whereas Optane DIMM system profiling shows an increasing latency with larger pointer chasing regions.



(a) Bandwidth and latency. (b) PtrChasing read latency.

Fig. 3. Comparison between memory simulators and Optane DIMM system profiling. (a) Simulator average accuracy wrt. Optane DIMM load/store bandwidth (*bw-ld* and *bw-st*) and latency (*lat-ld* and *lat-st*). (b) Comparison between Ramulator and Optane DIMM on read latency per cache line with the pointer chasing test.

D. Insufficient Profiling Tools

In order to develop a memory simulator that models the sophisticated performance behavior and microarchitecture of real NVRAM systems, we need to collect sufficient information about detailed performance characteristics. This requires a comprehensive architecture-level memory system profiling. Most previous performance profiling tools focus on investigating basic memory system performance characteristics, such as latency, bandwidth, and access counts. Beyond these characteristics, DRAMA [43] examines certain address mapping schemes. However, as illustrated in Table I, none of the widely-used performance profiling tools allows us to analyze detailed on-DIMM buffering and management schemes of Optane DIMM.

III. LENS: LOW-LEVEL NVRAM PROFILER

To address the challenge of insufficient profiling tools, we propose a *Low-level profilEr for Non-volatile memory System* (LENS). LENS consists of a set of NVRAM profiling tools and low-level microbenchmarks. With detailed performance profiling, LENS also allows us to reverse engineer the microarchitectural design of NVRAM systems with on-DIMM buffers and various control schemes. In this paper, we use LENS to profile the detailed architecture design of Optane DIMM, discussed in Section III-B.

A. LENS Framework

LENS adopts three key components – *probers* – to examine the following three aspects of NVRAM architecture design, respectively.

- **Buffer prober** – Analyzes on-DIMM buffers' architecture and their properties, including buffer (or queue) hierarchy, capacity, and access granularity.
- **Policy prober** – Analyzes NVRAM control policies on data migration and multi-DIMM interleaving.
- **Performance prober** – Facilitates the above two probers to analyze performance characteristics, including memory bandwidth and the access latency of various buffers and queues in NVRAM DIMMs and iMCs.

Each prober employs customized microbenchmarks to trigger specific hardware behaviors, which leads to different latency and bandwidth patterns with different memory architecture properties (Table II). By analyzing the performance patterns, we identify the corresponding microarchitecture properties and parameters.

TABLE II
LENS OVERVIEW.

Prober	Microbenchmark	Hardware Behavior	Microarchitecture
Buffer	PtrChasing (64B block)	Buffer overflow	Buffer size
	PtrChasing (various block)	R/W amplification	Buffer entry size
	Read-after-write	Data fast-forwarding	Buffer hierarchy
Policy	Sequential/Strided write	Interleaving speedup	Interleaving scheme
	Overwrite (256B region)	Data migration	Migration latency
	Overwrite (various region)	Data migration	Migration block size
Perf.	Strided write	Stable amplification	Internal bandwidth
	N/A	N/A	Internal latency

Microbenchmarks. LENS provides three microbenchmarks: *pointer chasing*, *overwrite*, and *stride*.

Pointer chasing is a random memory access benchmark: it divides a contiguous memory region – referred to as a pointer chasing region (PC-Region) – into equal-sized blocks (PC-Blocks); it reads/writes all PC-Blocks in a PC-Region in random order, and sequentially accesses data within each PC-Block. In order to bypass CPU caches, while still generating cacheline-sized memory accesses, all pointer chasing tests are implemented using non-temporal AVX512 load/store instructions. Pointer chasing has three variants to detect various buffer architecture characteristics: (1) collecting average latency per cache line with a fixed PC-Block size across various PC-Region sizes, (2) quantifying read and write amplification by using a fixed PC-Region size, while varying the size of PC-Block, (3) issuing read-after-write requests, which issue writes in a pointer chasing order, followed by reads in the same order.

Overwrite repeatedly generates sequential writes to a fixed memory region, and then measures the execution time of each iteration. It has two variants: (1) collecting the execution time of each write iteration with a fixed memory region size, (2) measuring the frequency of long tail-latency by changing the memory region size.

Stride sequentially reads or writes to a set of stride cache lines with a fixed striding distance. It has two variants: (1) measuring bandwidth by using a fixed striding distance and increasing the access size, (2) characterizing multi-DIMM interleaving by a fixed total access size and a variable striding distance.

Buffer Prober detects the on-DIMM buffer capacity, entry size, and organization, by measuring the latency change caused by buffer overflow and read/write amplification.

To identify the buffer capacity, the buffer prober runs the pointer chasing microbenchmark with fix-sized objects (PC-Block). The prober measures the average read/write latency by scanning through various pointer chasing region (PC-Region) sizes. Once a PC-region is sufficiently large to overflow a buffer, the prober will detect a rapid increase in read/write latency, e.g., the inflection point of the Optane latency curve at the 16KB region size in Figure 3b. As such, we estimate that the buffer capacity equals to the PC-Region size corresponding to the inflection point in the latency curve.

The prober also examines buffer organization by answering two questions: (i) How many levels in the buffers? (ii) If there are multiple levels of buffers, are they organized as a multi-level inclusive hierarchy or multiple independent buffers? We answer question (i) by analyzing the aforementioned buffer overflow results: the number of the inflection points in a latency curve indicates the number of buffers with different capacities. In fact, we identify that Optane DIMMs adopt multiple levels of on-DIMM buffers by inducing multiple overflows under different PC-Region sizes (Section III-C). To answer question (ii), the prober measures the data fast-forward effect – the effect of reading the existing dirty data in a buffer before it is written back to the underlying memory or the next level of buffer – by a pointer chasing read-after-write (RaW) test. Independent buffers can fast-forward data from each buffer in parallel. Therefore, the RaW latency is shorter than the sum of independently collected read latency and write latency. A multi-level inclusive buffer hierarchy does not have such a parallel data fast-forward behavior.

To identify the buffer entry size, the prober evaluates the read/write amplification – measured as a ratio of actually read/written data size to the requested data size. For example, let's consider a buffer with 256B entries, where a read request of 64B will read 256B into the buffer, leading to a read amplification of 4. As one memory bus transfer has a fixed size with a fixed latency, a read/write amplification will result in more bus transfers detected as a latency increase. In practice, the buffer prober identifies the entry size of various buffers by a series of pointer chasing tests with different PC-Block sizes. Once the read/write amplification drops to one, that PC-Block size is considered as the buffer entry size. We observe that Optane DIMMs adopt different entry sizes in different levels of buffers (Section III-C). Due to the lack of hardware counters we can access on Optane DIMM machines, we indirectly evaluate the read/write amplification by deriving an "amplification score". It is calculated as the ratio of latency of the buffer overflow case to the non-overflow case. As such, the amplification score drops to one if and only if the actual amplification drops to one.

Policy Prober investigates NVRAM control policies, including data migration (e.g., for wear-leveling) and multi-DIMM interleaving policy. Typical NVRAM wear-leveling schemes migrate data from one NVRAM media location to another to maintain evenly distributed wear out. The prober detects the frequency, granularity, and latency overhead of such data migration procedures.

To detect the migration frequency and latency, the prober employs the overwrite microbenchmark to constantly write 256B regions and measures the latency of each 256B write. Once a migration occurs in this region, e.g., triggered by a wear-leveling algorithm, the subsequent writes cannot be issued until the migration completes. As a result, the latency of a write delayed by a migration is over a magnitude higher than a normal write, showing as a tail latency. The prober estimates the migration latency to be equal to the elevated tail latency. In addition, the prober also collects the time intervals between two consecutive migrations to calculate the migration frequency.

To detect the migration granularity, the prober performs overwrite tests on regions of various sizes and collects the frequency of migration. Once the region size is sufficiently large to cross over two migration blocks, the frequency will drop. Therefore, we identify the migration granularity as the overwrite region size that leads to a frequency drop.

The policy prober also examines the multi-DIMM interleaving behavior, which is an address mapping mechanism to achieve higher bandwidth by spreading data among different DIMMs. To detect such an interleaving scheme in an NVRAM design, the prober measures the speedup of sequential and strided writes on NVRAM DIMMs. Compared to non-interleaved DIMMs, interleaved DIMMs result in lower execution time with same-sized sequential writes (Figure 7a). The prober also identifies the interleaving granularity (the size of the maximum continuous memory block assigned to each

TABLE III
SERVER HARDWARE CONFIGURATION.

CPU	Intel Cascade Lake engineering sample 24 Cores per socket, 2.2 GHz 2 sockets, HyperThreading off
L1 Cache	32KB 8-way I$, 32KB 8-way D$, private
L2 Cache	1MB, 16-way, private
L3 Cache	33MB, 11-way, shared
TLB	L1D 4-way 64 entries, L1I 8-way 128 entries STLB 12-way 1536 entries
DRAM	DDR4, 32GB, 2666MHz, 2 sockets, 6 channels per socket
NVRAM	Intel Optane DIMM, 256 GB, 2666 MHz 2 sockets, 6 channels per socket

DIMM) by detecting the repeated pattern in sequential and strided writes execution time on interleaved NVRAMs.

Performance Prober facilitates the analysis of the other two probers to measure the device bandwidth and latency for on-DIMM architecture components. The performance prober measures the read bandwidth of a buffer by a stride read experiment, with a stride size equals to the buffer entry size. Each buffer entry is read exactly once to prevent the impact of an upper-level buffer on the bandwidth. To measure the latency of accessing a buffer, the prober takes pointer chasing latency results from the buffer prober and then estimates the buffer miss rate based on the buffer size and access granularity. Then, the prober calculates the buffer latency as a function of pointer chasing latency and buffer miss rate.

LENS Implementation. We build LENS as a Linux kernel module to avoid the overhead of switching between user and kernel spaces. We implement all our microbenchmarks in assembly language to enforce a well-controlled memory access behavior. We also disable process preemption and hardware prefetchers to avoid noises on probing results.

B. Optane DIMM System Characterization

We employ LENS to characterize the architecture details of a real server memory system with Optane DIMMs.

Real System Configuration. Table III lists our server hardware configuration. We set the Optane DIMMs to App Direct mode and use a customized Linux v4.13 kernel that supports App Direct mode. We use the `ipmctl` tool [26] to configure these Optane DIMMs in non-interleaved mode; we also employ the `ndctl` tool [27] to create a Linux `pmem` device on a single Optane DIMM from the local NUMA domain. LENS creates a dummy filesystem on the `pmem` device and performs experiments in the kernel space. We set the Model Specific Register (MSR) 0x1a4 to 0xf to disable the following CPU cache prefetchers to avoid noises in profiling [55]: L2 hardware prefetcher, L2 adjacent cache line prefetcher, DCU prefetcher, and DCU IP prefetcher. We run our experiments for over 500 times and show average profiling results with error envelopes/bars (some error envelopes may be too narrow to be visible in the figures).

Overview of Our Characterization and Observations. Figure 4 shows an overview of our real-machine characterization.

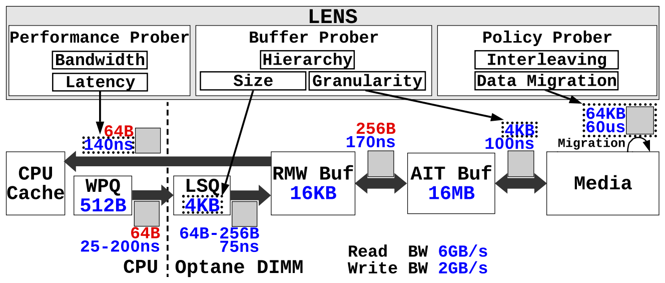


Fig. 4. LENS probers and Optane DIMM parameters. Red numbers are obtained from Intel documents; blue numbers are characterized by LENS.

Each prober identifies certain memory system architecture and performance characteristics, as illustrated by arrows in the graph. Figure 8 shows details of the Optane DIMM microarchitecture identified by our characterization. We identified the write-pending-queue (WPQ) size and multi-DIMM interleaving scheme in iMC. We also identified two on-DIMM buffers, a 16KB SRAM-based read-modify-write (RMW) buffer and a 16MB DRAM-based address indirection translation (AIT) [3] buffer, with 256B and 4KB access granularity, respectively. We find that these two buffers are organized as a two-level inclusive buffer hierarchy rather than independent buffers. We also identified the size of a load-store-queue (LSQ) [49], which reorders the incoming requests to perform write combining. Finally, we identified a long tail-latency effect, which may be caused by wear-leveling data migration. We present details of our characterization results and observations in Section III-C and Section III-D.

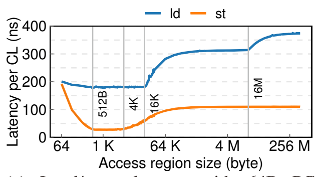
The Existence of the Architecture Components. Industrial articles revealed the existence of several aforementioned components, as described in Section II-A. In addition, Intel's announcement [23], [49] declared that transactions could be re-ordered in Optane DIMM, indicating an on-DIMM reorder queue, which we refer to as LSQ. Optane DIMM is also believed to have on-DIMM SRAM and DDR4 DRAM modules [31]. However, these articles do not provide detailed information about these components, such as size, access granularity, management policy, and latency. We employ LENS to profile these parameters and identify other architecture components.

Profiling Results Confirmation. We confirmed as much of our profiling results as possible with the vendor, including our bandwidth profiling, idle latency of sequential and random accesses, and load latency across various access granularities.

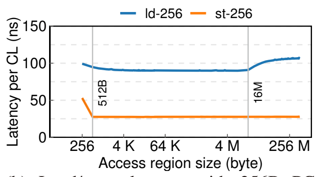
C. On-DIMM Buffer Analysis

We run the LENS buffer and performance probers to profile the on-DIMM buffer architecture design.

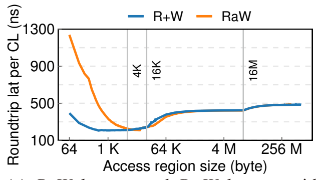
Buffer Capacity. Figure 5 shows LENS buffer prober results. Figure 5a and Figure 5b show a pointer chasing read/write test with various PC-Block and PC-Region sizes. The experiments measure latency per cache line (CL) access. The shape of the curves exposes the read requests overflow points at 16KB and 16MB, where the read latency drastically changes. This indicates that the Optane DIMM has two levels of read buffers,



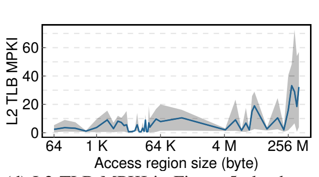
(a) Load/store latency with 64B PC-Block.



(b) Load/store latency with 256B PC-Block.

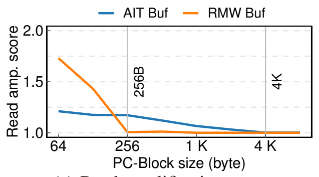


(c) *RaW* latency and *R+W* latency with 64B PC-Block.

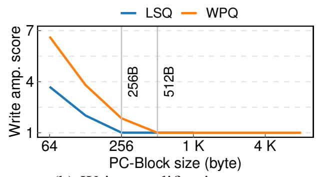


(d) L2 TLB MPKI in Figure 5a load test.

Fig. 5. LENS buffer prober tests on Optane DIMM. The curves show average results of 500 runs; shaded areas represent error envelopes. In (c) *RaW* is read-after-write, *R+W* is the sum of load and store latency in (a).



(a) Read amplification score.



(b) Write amplification score.

Fig. 6. Read(a) and write(b) amplification scores in our buffer prober test.

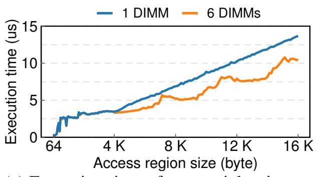
one 16KB and the other 16MB. Figure 5a also shows that the write curve (denoted by *st*) has two overflow points at 512B and 4KB, respectively. This indicates that the memory has two write buffers, namely WPQ and LSQ, of two distinct capacities. Note that the L2 TLB of our server machine has 1536 entries, which can store the address translation of up to 4KB$\times$1536, i.e., 6MB, of data. However, Figure 5a shows that the read latency significantly increases at 16MB instead of 6MB, ruling out that this happens due to L2 TLB misses. Furthermore, we track L2 TLB misses for the load test. As shown in Figure 5d, the L2 TLB miss rate remains stable without significant change at 16KB and 16MB access regions. Therefore, we conclude that L2 TLB misses are not the main reason for the sudden latency increases in Figure 5a.

We refer to the 16MB buffer as AIT Buffer. Because its read latency is approximately 100ns as shown in Figure 5a, it is likely to locate inside the on-DIMM DRAM, where AIT resides [3], [23]. We believe that the 512B buffer is WPQ [48], given its small size; the 4KB buffer is an on-DIMM LSQ that reorders the read/write requests [49].

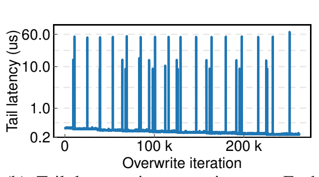
Access Granularity. Figure 6a shows our read amplification tests. Our results indicate that the RMW Buffer and AIT Buffer adopt 256B and 4KB access granularities, respectively. Figure 6b shows our buffer write amplification tests. It demonstrates that the two write queues, WPQ and LSQ, adopt 512B and 256B granularities, respectively. As a result, a *mfence* will cause the WPQ to flush in total 512B data; the LSQ combines 64B writes into 256B in order to reduce RMW operations.

Buffer Hierarchy Organization. Figure 5c shows our read-after-write (RaW) experiment to characterize the buffer hierarchy organization. Figure 5c indicates that RMW Buffer and AIT Buffer form a two-level inclusive hierarchy – if they were independent from each other, they would have demonstrated a RaW latency speedup at 16MB by fast-forwarding data in parallel. They do not appear to be exclusive either, because they adopt different access granularities and entry sizes.

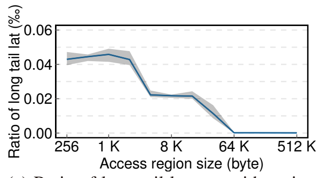
Figure 5c also demonstrates that the read-after-write latency (denoted as *RaW*) is significantly higher than the total latency



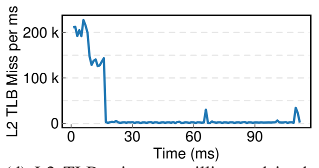
(a) Execution time of sequential write test.



(b) Tail latency in overwrite test. Each iteration is one 256B write.



(c) Ratio of long tail latency with various overwrite granularities.



(d) L2 TLB miss per millisecond in the overwrite test of (b).

Fig. 7. LENS policy prober tests on Optane DIMM.

of performing individual reads and writes (denoted as *R+W*) for small PC-Regions. The key reasons are small-sized RaW requests (i) trigger frequent memory bus redirection [69] and (ii) under-utilize the WPQ and LSQ capacity. Therefore, due to the identified architectural design, small-sized requests in Optane DIMMs tend to perform better with pure reads and writes than with mixed read and write access patterns. In addition, the high RaW latency with small PC-Region sizes also indicates that *mfence* flushes the LSQ, because: (i) RaW latency reduces when the PC-Region size increases; (ii) RaW latency equals the R+W latency when the PC-Region size reaches 4KB, the size of the LSQ.

D. Policy Analysis

We run LENS policy and performance probers to profile Optane DIMM management schemes.

Multi-DIMM Interleaving Analysis. Figure 7a shows a multi-DIMM interleaving policy analysis. LENS measures the execution time of various sized sequential writes on interleaved and non-interleaved DIMMs, respectively. In the interleaved experiment, the first 4KB has a similar execution time as in the non-interleaved case, indicating that the first 4KB is written to a single DIMM. We also observe a repeated execution time pattern every 4KB, indicating that different 4KB data writes are directed to different DIMMs. These observations indicate a 4KB granularity for multi-DIMM interleaving. We consider the reason for the interleaving granularity is to fully utilize the 4KB sized LSQ and 4KB entries in AIT Buffer.

Tail Latency Analysis. Figure 7b shows the overwrite experiment for tail latency analysis. The policy prober constantly writes data to the same 256B memory region, and measures

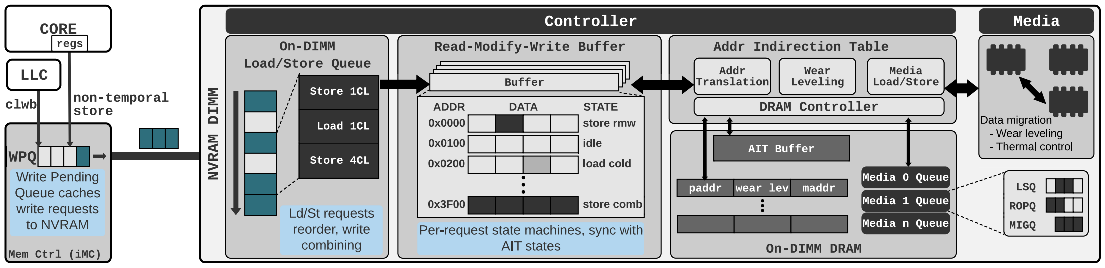


Fig. 8. VANS overview.

the latency of each 256B write. We observe a long tail latency every $\sim 14,000$ write iterations (i.e., every 3.4MB for 256B overwrite test), which incurs over $100\times$ latency penalty on average. We do not observe similar trends on DRAM. Moreover, L2 TLB miss rate remains stable in the overwrite experiment as shown in Figure 7d. Therefore, we consider wear-leveling as a major contributor to the tail latency.

To study the size of the blocks that wear-leveling tracks, we increase the size of the overwrite test region and count the frequency of the long tail latency. Each test writes the same amount of data to NVRAM. As shown in Figure 7c, the frequency of long tail latency dramatically drops, once we overwrite on 64KB or larger memory regions. This indicates that a possible block size for wear-leveling is 64KB.

IV. VANS: Validated NVRAM Simulator

Based on our profiling observations, we build *Validated cycle-Accurate NVRAM Simulator* (VANS) that models Optane DIMM's microarchitectural design and its parameters. Figure 8 depicts an overview of VANS (CPU core and LLC are not part of VANS). We adopt a modular implementation by decoupling the architecture's modules as much as possible, in order to allow users to modify or extend the simulator with various architectural designs. VANS also offers an interface to be attached to full-system simulators, such as gem5 [4].

Because no publicly available formal verification tool exists for NVRAM DIMMs, we validate VANS's accuracy by comparing the simulated bandwidth, load and store latency, and various SPEC CPU benchmarks performance results with Optane DIMM real-system profiling. In addition, the on-DIMM DRAM model is verified by Micron's verification model [38] and Cadence toolchain [5].

A. Simulation Model Design

VANS models both released Optane DIMM hardware components (e.g., WPQ in iMC) and undocumented microarchitectural components (e.g., LSQ, multi-level buffers). The iMC in VANS supports multi-DIMM control, which controls multiple DRAM and NVRAM DIMMs to provide both Memory and App Direct modes. We implement the WPQ in iMC with Asynchronized DRAM Refresh (ADR). The NVRAM DIMM model consists of an LSQ, an RMW Buffer, and an AIT. The LSQ serves as the highest-level storage in the DIMM, directly queuing load/store requests from the iMC. During each scheduling epoch, the LSQ performs write combining to reduce the number of read-modify-write operations.

The RMW Buffer receives read and write requests from the LSQ, and accesses the AIT Buffer at certain granularity (by default 256B based on our Optane DIMM analysis). The RMW Buffer performs read-modify-write operations if write requests are smaller than 256B. We place this buffer in SRAM, based on our Optane DIMM profiling. The AIT consists of a translation table and a data buffer (AIT Buffer). The translation table stores the records of CPU address to media address translation. It also stores the media wear-leveling records in each table entry. If one media block is wearing out, AIT stalls the inflight CPU writes to this block, migrates the data into another media block, updates the translation record, and then resumes the CPU write. The AIT Buffer stores data from the media to accommodate read and write requests from RMW Buffer. We place the AIT table and buffer in the on-DIMM DRAM, based on our Optane DIMM profiling.

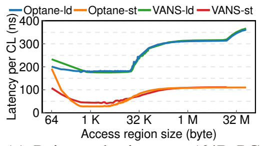
VANS adopts a first-come-first-serve scheduling policy by default inside the NVRAM, based on our Optane DIMM profiling. In addition, on the DIMMs, both RMW and AIT buffers maintain a state machine for each buffer entry running in parallel. The RMW Buffer issues FIFO requests to the AIT Buffer. The AIT Buffer issues FIFO requests to the NVRAM media. The scheduling policy can be modified by users to adapt to other NVRAM DIMM devices (Section IV-E).

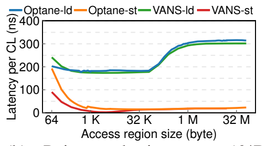
The iMC and the NVRAM DIMM communicate by a request/grant scheme [49]: for example, when the iMC needs to read a cache line from NVRAM DIMM, it sends a cache line read request and waits for NVRAM DIMM's response. When the NVRAM DIMM finishes fetching this cache line into the RMW Buffer, it notifies the iMC by sending a data-is-ready signal and requests an entry in the iMC's read pending queue (RPQ). It then sends data to the RPQ, after the iMC allocates the space for it.

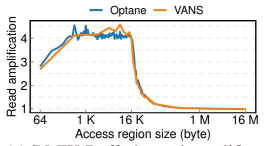
Optane DIMM adopts an unpublished DDR-T protocol, which extends DDR4 with persistence-related commands. Therefore, we model the on-DIMM DRAM timing based on the DDR4 protocol.

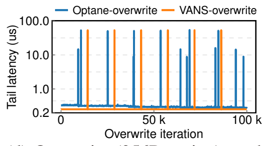
B. DRAM Model Verification

To verify our DRAM model, we first catch memory traces from SPEC2006 and SPEC2017 benchmarks listed in Table IV; these traces are fed into VANS to obtain internal DRAM command traces. Then, we employ Micron's DDR4 verification model [38] with Cadence toolchain [5] to test these command traces. The results demonstrate that our model does


(a) Pointer chasing test (64B PC-Block size) on non-interleaved DIMM.


(b) Pointer chasing test (64B PC-Block size) on interleaved 6DIMMs.


(c) RMW Buffer's read amplification in Figure 9a test.


(d) Overwrite (256B region), each iteration is one 256B write.

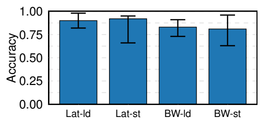
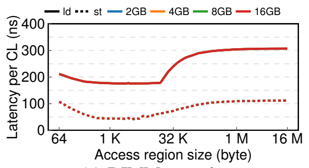

(e) VANS accuracy over different metrics with different access sizes.

Fig. 9. VANS performance validation with microbenchmarks: (a-c) pointer chasing and (d) overwrite. (e) VANS overall accuracy on load/store latency and bandwidth.


(a) DIMM capacity.

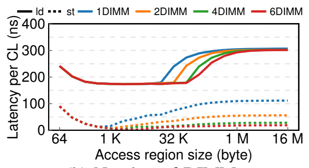

(b) Number of DIMMs.

Fig. 10. Sensitivity study of memory configurations.

not generate any illegal DDR4 command, showing that our on-DIMM DRAM model complies with DDR4 specification.

C. Validation with Microbenchmarks

We use microbenchmarks to validate the accuracy of VANS's architecture model. We catch memory traces of LENS microbenchmarks and feed them into VANS. We run VANS alone in trace mode[2] and compare the simulated load/store latency, bandwidth, and read amplification with Optane DIMM-based real-machine results. Overall, our evaluation shows that VANS achieves an average 86.5% accuracy across four metrics (Figure 9e). Figure 9 (a-d) shows various performance curves generated by simulation and real-system profiling.

Figure 9a shows the load and store latency of the pointer chasing microbenchmark across different access region sizes with single-DIMM configuration. The store latency curves of VANS match the curves of real-system profiling well with a difference lower than 10% across all the access region sizes. The reason for the deviation of store latency at access region sizes smaller than 4KB (31.54%) is that small-sized write tests do not overflow the NVRAM buffers or queues on the real machine. Therefore, the store latency collected from real machine experiments is dominated by CPU on-core latency (e.g., the latency of executing `mfence`), which is not evaluated by this set of simulations without attaching a CPU model.

Figure 9b shows the pointer chasing latency of six DIMMs with interleaved configuration. We apply the 4KB-based interleaving scheme in VANS. Figure 9b shows an average 12% difference between simulation and real machine profiling.

Figure 9c validates the RMW Buffer read amplification in pointer chasing. We employ Intel's in-house performance profile tool to investigate Optane DIMM's read amplification in RMW Buffer. We compare the results with the corresponding statistics generated by VANS. The difference across the curves is within 9%. Due to the limitation of Intel's profiling tool, we cannot compare other hardware counters.

[2]We run VANS alone without gem5, because gem5 does not support the non-temporal AVX512 instructions in the microbenchmarks.

TABLE IV
EVALUATED SPEC CPU BENCHMARKS.

SPEC	Workload	LLC MPKI	Footprint
2006	gcc	2.9	1.2 GB
	mcf	27.1	9.1 GB
	sjeng	2.7	0.63 GB
	libquantum	3.4	2.3 GB
	omnetpp	2.1	1.4 GB
	cactusADM	2.0	2.2 GB
	lbm	7.7	2.9 GB
	wrf	2.4	1.0 GB
2017	gcc	21.5	1.1 GB
	mcf	26.3	8.7 GB
	omnetpp	2.1	0.96 GB
	deepsjeng	2.5	0.58 GB
	xz	2.7	1.8 GB

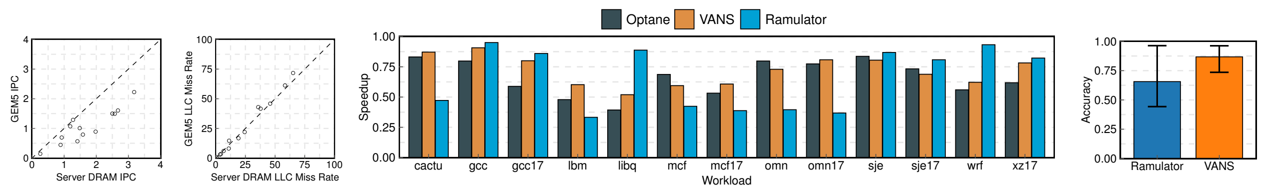
Figure 9d shows the tail latency for a 256B overwrite test. Each "iteration" performs a 256B write to a single NVRAM DIMM. The simulation matches the real-machine profiling in terms of tail latency length and interval.

We validate VANS accuracy with 4GB DIMM capacity (i.e., 4GB NVRAM media size), with a 256GB Optane DIMM media size. To study the impact of media size, we investigate pointer chasing benchmark on VANS with different media sizes (buffer and queue sizes remain unchanged). Figure 10a compares VANS-ld and VANS-st curves in Figure 9a (noted as ld and st in Figure 10a) with curves generated by various NVRAM media capacities. Media capacity does not affect the latency curves, because the media latency is hidden by on-DIMM buffers and queues. Figure 10b shows a sensitivity study on the number of DIMMs. With more DIMMs, the buffering effect on load latency is postponed, and the store latency reduces once memory access overflows the WPQ.

D. Validation with SPEC CPU Benchmarks

To further validate VANS, we compare VANS+gem5 full system simulation with real-machine profiling on SPEC CPU benchmarks [50], [51].

Benchmarks. We employ SPEC CPU 2006 [50] and 2017 [51] benchmark suites in our experiments. We first run all SPEC CPU benchmarks in speed mode on the Optane DIMM-based server machine. We use Linux perf [15] and Intel's Emon [22] tools to profile each benchmark. Based on the profiling, we select the memory-intensive workloads that have at least two last-level cache (LLC) misses per thousand instructions (MPKI) for the rest of our experiments. Table IV shows the selected benchmarks, as well as their LLC MPKI statistics and main memory footprints.



(a) IPC. (b) LLC Miss. (c) Speedup comparison between simulators and Optane. (d) Accuracy.

Fig. 11. VANS performance validation with SPEC CPU 2006 and SPEC CPU 2017. (a,b) IPC and LLC miss ratio comparison between DRAM-based simulation and the server with DRAM main memory. (c) Speedup comparison between NVRAM-based simulation and the server with Optane DIMM main memory. (d) Accuracy (geometric mean).

TABLE V
SIMULATED SYSTEM CONFIGURATION.

CPU	
Core	4 cores, out-of-order, 2.2GHz
ROB-SQ-LQ	224-56-72 entries
L1 Cache	I$ 32KB 8-way, D$ 32KB 8-way, private
L2 Cache	1MB, 16-way, private
L3 Cache	32MB, 16-way, shared
TLB	L1I 128 entries 8-way, L1D 64 entries 4-way L2TLB 1536 entries (1024 d-entries, 512 i-entries)
WPQ	512B
DRAM Main Memory	
Configuration	DDR4 2666MHz 4-channel, 4GB/channel
Timing	tCAS(19) tRCD(19) tRP(19) tRAS(43)
NVRAM Main Memory	
Configuration	2666MHz 6-channel, 4GB/channel
Interleaving	4KB interleaving
LSQ	64 entries, 64B line
RMW Buffer	64 entries, 256B line
AIT Buffer	4096 entries, 4KB line
Internal DRAM	512MB DDR4 2666MHz
Operation Mode	AppDirect
Software	
Linux Kernel	v5.1.15
GNU gcc	v9.1

Simulator and Real Machine Configurations. We attach VANS to gem5 to conduct full system simulations. We modify gem5 to model the Cascade Lake microarchitecture [60] that we adopt in our server machine. Table V shows the details of our server machine configurations. We perform two sets of experiments on the server with (1) pure DDR4 DRAM DIMM main memory and (2) Optane DIMM-based NVRAM main memory, respectively. The Optane DIMMs operate in *Memory mode* and are exposed as NUMA nodes [16]. Each workload is binded to a local Optane NUMA node. Our full system simulation adopts the same system configurations. The experiments on the DRAM-based server are used to evaluate the accuracy of our gem5 modification. To this end, we ensure the accuracy of the DRAM timing simulation attached to gem5 by modifying VANS to simulate DDR4 DIMMs with a formally verified DDR4 protocol timing model (Section IV-B). In experiments on NVRAM DIMMs, we configure VANS to have the same microarchitecture and parameters of Optane DIMM, based on industrial documents and LENS characterization observations (Section III). To compare VANS with other memory simulators, we attach Ramulator [32] PCM model to gem5.

Results. Each simulation has two stages: (1) a warm-up stage that employs the gem5 simple CPU model to achieve a stable LLC MPKI; (2) an execution stage that executes at least two billion instructions on gem5 detailed CPU model. We first evaluate the accuracy of our gem5 modification by DDR4 DRAM-based system experiments. We compare the execution stage instruction per cycle (IPC) and LLC miss rate of each benchmark between DRAM-based simulation and server profiling. Figure 11a shows that the accuracy of IPC is 61.2% on average (geometric mean). As we ensure the accuracy of our DRAM timing simulation by formal verification, the main reason for the accuracy loss is that we cannot fully implement Cascade Lake microarchitecture, due to the limited architecture details that can be modeled in gem5. Figure 11b shows a comparison of LLC miss rate (LLC miss / LLC total references). The average accuracy is 85.5%. Note that we turn off the cache prefetchers on both our server machine and simulation to avoid noises. Figure 11c shows a comparison of our NVRAM-based simulation with Optane DIMM server profiling. We calculate speedup as

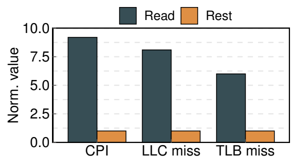
$$Speedup(workload) = \frac{ExecTime_{DRAM}(workload)}{ExecTime_{NVRAM}(workload)}$$

VANS+gem5 achieves an accuracy of 87.1% on average (geometric mean) across various benchmarks, despite the less accurate CPU simulation model used. Ramulator (PCM)+gem5 only achieves an average accuracy of 65.6%.

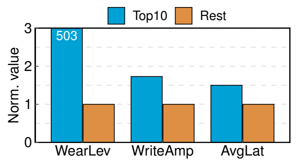
E. Discussion

Simulator validation and verification methodologies. We adopt the same verification and validation methodologies used by various existing architecture simulators. DRAM simulators, such as DRAMSim2 [46] and Ramulator [32], also used the Micron's verification model to verify their DRAM models. Our performance-based validation methodology is also adopted by several widely-used architecture simulators [14], [52]: Amber [14], an SSD simulator, is validated by comparing simulation results with SSD products on (i) microbenchmark latency and bandwidth and (ii) bandwidth, power, and system performance of real workloads. MGPUSim [52], a multi-GPU simulator, employs (i) microbenchmarks to stress test each microarchitecture component and (ii) benchmarks, selected from two benchmark suites, to validate the performance accuracy of the simulator.

Modeling Other NVRAM DIMMs. VANS is initially developed to model Optane DIMM. But it can be flexibly adapted to other NVRAM DIMM architectures. To do so, users first



(a) Redis



(b) YCSB

Fig. 12. Redis and YCSB profiling. Each metric value is normalized to the corresponding *Rest* case. "Top10" stands for the ten most frequently written cache lines.

need to run LENS on the target NVRAM system to obtain insights on its microarchitecture design, e.g., internal latency and buffer structures. Then, the users can reconfigure VANS based on the new parameters and microarchitecture design. Our modular software design allows users to add additional architecture components.

V. Cloud Workload Optimizations

The advantages of capacity and persistence make NVRAM a natural fit for memory-hungry cloud applications [30]. In this section, we use VANS to study the use of NVRAM for cloud applications. We identify the performance overhead due to applications' memory access patterns and alleviate the overhead with two architecture optimizations.

A. Performance Inefficiency

To understand the NVRAM architectural behavior when running cloud applications, we run two widely used cloud workloads, Redis [25] and YCSB [10], in VANS+gem5 full system simulation with the configuration in Table V. The results show that the pointer chasing access patterns and the wear-leveling triggered write amplification are the two primary sources of inefficiency.

Notable Read Misses. Figure 12a shows that read operations dominate the execution overhead in Redis. Cycles Per Instruction (CPI) of read operations is $8.8\times$ higher than CPI of other workload activities. The major overhead originates from reads leading to misses in LLC and TLB, due to the pointer chasing memory access pattern that repeatedly shifts between random memory regions. Such a pattern is commonly observed in data structures that are often used in cloud workloads, such as B-Trees and hash tables [12], [13], [18].

Write Amplification Overhead. Figure 12b shows that most YCSB writes are concentrated in ten cache lines, which are written over $100\times$ more than the total of other cache lines. Compared with accesses to other cache lines, the writes to the Top10 cache lines, which form only 15% of total memory traffic, trigger $503\times$ more wear-leveling operations. These operations lead to higher write amplification than others and elevate average memory access latency.

B. In Memory Pre-Translation

To address the read miss induced overhead, we propose *Pre-translation* to optimize the address translation in the read operations. We obverse that the NVRAM performs a "page translation" within the NVRAM DIMM that translates the physical address to the media address[3]. To allow the CPU to concurrently receive data for the current memory access and the TLB entry for the next memory access, Pre-translation combines the NVRAM translation with the CPU translation. As shown in Figure 13a, Pre-translation consists of two hardware structures – the Pre-translation table and the Read Lookaside Buffer (RLB) – and a new instruction `mkpt`.

The Pre-translation table is stored in the on-DIMM DRAM as a part of an AIT entry [3]. This table maps a physical address (paddr) to a page frame number (pfn) pointed to by paddr. In practice, it stores only the pfn and employs paddr as an index; this allows the integrated DRAM to store more Pre-translation entries. To take advantage of the existing translations in AIT, we add a pointer to each AIT entry, so that it takes only one more DRAM access to find the corresponding Pre-translation table entry (Figure 13b).

Similar to the TLB, RLB buffers the Pre-translation table and is stored in SRAM. Each RLB entry consists of a page frame number from a Pre-translation table entry and a physical address that is used as the index to the table entry.

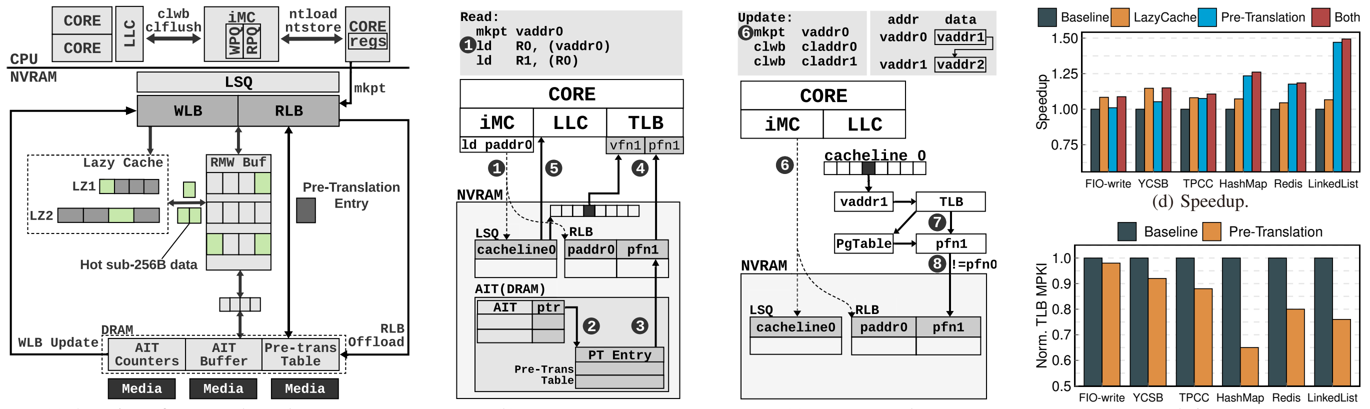
The new instruction `mkpt` is used to update Pre-translation table. This instruction takes one virtual address as an argument and hints the memory controller to read or update the Pre-translation entry. To use this instruction, programmers or compiler need to insert it before load and store instructions that lead to a pointer chasing access.

A Pre-translation Involved Example. We use an example of a linked list traversal to demonstrate the detailed hardware operations in the Pre-translation, as shown in Figure 13b and Figure 13c. In this example, the traversal currently visits the node stored in address `vaddr0` (top right Figure 13c), and generates a pointer chasing access.

To use Pre-translation (Figure 13b), `mkpt` is invoked to "mark" a read access to `vaddr0` (❶). When NVRAM receives the read request with this mark, it checks the RLB or Pre-translation table for an entry, using `paddr0` as an index (❷). If the Pre-translation entry is found (❸), NVRAM uses the data at `paddr0` and page frame number `pfn1` from the Ptr-translation entry to create a TLB entry (❹); this entry is sent to the CPU's TLB when returning the requested data (❺). In this case, the CPU receives the data at `vaddr0` and the TLB entry for the next access to `vaddr1` simultaneously.

The Pre-translation table is updated by `mkpt` (Figure 13c): each time when `mkpt` is invoked with an address `vaddr0` (❻), the CPU checks if the corresponding Pre-translation entry is missing or out-of-date. If so, the CPU updates the Pre-translation entry: it first translates `vaddr0` to `paddr0` and fetches the data at this address, then it uses this data as a virtual address (`vaddr1`) to find out the page frame number `pfn1` (❼). Finally, the CPU writes the entry `<paddr0, pfn1>` to RLB, if `pfn1` is not equal to `pfn0` (❽).

[3]"Physical address" refers to the address translated by MMU. The "media address" refers to the real physical address used in the NVRAM media, which is transparent to CPU memory controllers.



(a) Overview of Lazy cache and Pre-trans.

(b) Pre-trans usage.

(c) Pre-trans update.

(d) Speedup.

(e) Pre-translation TLB MPKI.

Fig. 13. Customized Lazy cache(a) for write intensive workloads and Pre-translation(a–c) for pointer chasing read intensive workloads. Baseline in (d–e) is measured without any optimization mechanism.

Validating Pre-translation Entry. Pre-translation entries may become stale, when the CPU updates the page table. To reverse the effect of stale TLB entries, fetched from NVRAM, we propose a "check-before-read" mechanism, which relies on the existing NVRAM memory request queue in the CPU's memory controller[4]. We extend each entry in the NVRAM request queue with a new "uncertain" bit. When CPU gets a TLB entry from the NVRAM, the memory management unit (MMU) uses this entry to translate the virtual address and issues a read request to NVRAM. This read request enters the NVRAM request queue with the uncertain bit set. In the meantime, the MMU starts an asynchronized page table walk to determine whether this TLB entry from NVRAM matches the latest record in the page table. If this TLB entry is stale, the memory controller updates the corresponding NVRAM read request with the latest translation info, issues the updated read request to NVRAM, and sets the uncertain bit to 0. If the TLB is up-to-date, then the memory controller just resets the uncertain bit to 0. The data is not fetched from NVRAM if the corresponding uncertain bit is 1, so the CPU never gets a wrong cache line caused by a stale TLB entry.

C. Lazy cache

To reduce the write amplification in cloud workloads, we add a Lazy cache to NVRAM (Figure 13a) to cache the frequently written data. It adopts a 2-level inclusive cache, LZ1 and LZ2, with 64B and 128B granularity, respectively. It employs a Write Lookaside Buffer (WLB) to store the addresses of Lazy cache entries.

We reuse AIT's wear-out record to update Lazy cache: once a write triggers wear-leveling, AIT calculates the Lazy cache priority during the data migration; it requires Lazy cache to cache the incoming write to this location, if the priority is higher than a certain threshold. This Lazy cache update does not exacerbate the performance and storage overhead, as AIT migration is significantly slower than the Lazy cache access and we reuse the existing AIT records.

[4]According to Intel's announcement [49], CPU and NVRAM work in a request/grant scheme. This indicates that the CPU maintains a memory request queue in iMC for NVRAM accesses.

Lazy cache relies on the existing ADR [48] to ensure data persistence in the face of power outages. In Section V-D, we show that a small Lazy cache (3KB) is efficient in improving performance and reducing memory traffic. The 3KB Lazy cache is much smaller than other on-DIMM buffers (16KB and 16MB). Thus ADR is sufficient to ensure its persistence.

D. Evaluation

To evaluate the performance of Pre-translation and Lazy cache, we use fio [2], two PMDK [24] microbenchmarks (HashMap and LinkedList), and three cloud workloads from WHISPER benchmarking suite [39] (Redis, TPCC and YCSB). Table V lists our simulation setup. We run unmodified workloads without any architectural optimization to obtain the baseline performance.

To compare with baseline, we configure (1) Pre-translation with a 1KB RLB and a 16MB Pre-translation table and (2) Lazy cache with a 1KB L1 cache and a 2KB L2 cache. We modify the workloads' source code to use Pre-translation, and use unmodified workloads on Lazy cache.

Figure 13d shows the evaluation results of Pre-translation, Lazy cache, and a combination of both. Pre-translation achieves 1% to 48% speedup across all six workloads. It efficiently optimizes TLB misses to achieve this speedup: as shown in Figure 13e, Pre-translation reduces the TLB misses per thousand instruction (MPKI) by 17% on average. Lazy cache achieves an average of 10% speedup across all six workloads. It achieves this by detecting memory patterns that trigger wear leveling on a small region, and caching these writes accordingly, thereby reducing the write amplification overhead. Using both Pre-translation and Lazy cache achieves 8% to 49% speedup across all workloads. It does not achieve the ideal speedup, because enabling both optimizations leads to higher memory bus contention.

Overall, Pre-translation optimizes pointer chasing, as it reduces the read operation overhead as well as the TLB miss rate. Lazy cache is able to reduce the write operation overhead and write traffic. The combination of these two mechanisms benefits cloud workloads running on real NVRAM-based system under various settings.

VI. Related Work

To our knowledge, this is the first paper to unveil the architecture details of real NVRAM DIMM products in server systems and develop an architecture-level simulation model for the device. This section discusses related works.

NVRAM Emulation and Simulation. Besides PMEP [11] and Quartz [56], previous works also explored using FPGA to emulate NVRAM, e.g., Lloyd et al. [35] proposed an FPGA emulator to model the latency projected for NVRAM. However, most of the emulators model NVRAM by injecting delays to mimic the NVRAM latency. Previous memory architecture simulators used to model NVRAM, such as NVMain [44], DRAMSim2 [46], and Ramulator [32], model NVRAM based on conventional DRAM DIMM architecture and timing. However, as discussed in Section II, these emulators and simulators do not fully model the performance and microarchitecture characteristics of Optane DIMM.

Optane DIMM System Studies. Most of the existing Optane DIMM system studies focus on system-level performance profiling and system software design. Recent works observed inefficient performance on Optane DIMM by running the existing NVRAM-aware system software with certain access patterns. For example, several studies [7], [28], [65] observed that repeated writes to a concentrated Optane DIMM memory area lead to high write latency. FlatStore [7] addresses this issue by grouping small-sized writes and evenly spreading writes to multiple Optane DIMMs. To reduce the overhead caused by the long write latency, MOD [17] proposes a framework to build persistent data structures with minimal ordering instructions. Other studies observed that concurrent or mixed reads and writes to the Optane DIMM in a remote NUMA node lead to performance degradation with high-performance computing and storage workloads [41], [59], [65]. In addition, multi-threaded memory accesses do not scale well on Optane DIMM systems [7], [28], [42], [58]. A recent study [65] shows that one reason behind this issue is the contention in the WPQ and RMW buffers. Our study shows that the contention in the AIT Buffer and the LSQ exacerbates this scaling issue. Recent work [62] investigated two data structure design strategies for RocksDB: Memtable that generates small random accesses and FLEX that generates sequential accesses. DRAM-emulated NVRAM experiments show that Memtable is 19% faster than FLEX; but experiments on Optane DIMM show that FLEX is 10% faster than Memtable [62]. Our study indicates that one main reason is that the small random accesses of Memtable do not fully utilize the buffer capacity and access granularity. Existing NVRAM-aware system software design may also lead to high read and write amplification on Optane DIMM [28], [65]. However, these studies only show the inefficiency from the system software design perspective. Our study indicates that one main reason for the amplification is the large on-DIMM buffer access granularity.

Pointer Chasing Optimizations. Jump pointer [47] proposes a data prefetching scheme for linked data structures. It requires a new data field in each data structure. Moreover, it does not reduce the TLB miss overhead. Our Pre-translation reduces TLB miss overhead without requiring modification on data structures. ASAP [36] proposes a prefetcher for page table walks to reduce the execution time for each TLB miss. Pre-translation also reduces the TLB miss rate.

VII. Conclusion

We design LENS and VANS, a low-level performance profiler for NVRAM systems and an architecture-level memory simulator that models the recently released Optane DIMM memory system. Using LENS, we perform a detailed characterization of the Optane DIMM microarchitecture design. LENS allows users to characterize the performance and reverse engineer the architecture design of real NVRAM DIMM systems. VANS allows researchers who do not have access to Optane DIMM physical devices to explore and evaluate new design ideas on architecture and systems. Furthermore, both LENS and VANS are flexible to be used with other NVRAM systems beyond the current Optane DIMM design.

Acknowledgment

We thank the anonymous reviewers for their valuable feedback. This paper is supported in part by NSF grants 1829524, 1817077, and SRC/DARPA Center for Research on Intelligent Storage and Processing-in-memory.

References

[1] "Open source code repository for LENS and VANS," 2020. [Online]. Available: https://github.com/TheNetAdmin/LENS-VANS

[2] J. Axboe, "Flexible I/O tester," 2019. [Online]. Available: https://github.com/axboe/fio

[3] B. Beeler, "Intel Optane DC persistent memory module (PMM)," 2019. [Online]. Available: https://www.storagereview.com/news/intel-optane-dc-persistent-memory-module-pmm

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

[5] Cadence Design Systems, Inc., "Simulation and testbench verification," 2019.

[6] E. Chen, D. Apalkov, Z. Diao, A. Driskill-Smith, D. Druist, D. Lottis, V. Nikitin, X. Tang, S. Watts, S. Wang, S. A. Wolf, A. W. Ghosh, J. W. Lu, S. J. Poon, M. Stan, W. H. Butler, S. Gupta, C. K. A. Mewes, T. Mewes, and P. B. Visscher, "Advances and future prospects of spin-transfer torque random access memory," *IEEE Transactions on Magnetics*, 2010.

[7] Y. Chen, Y. Lu, F. Yang, Q. Wang, Y. Wang, and J. Shu, "FlatStore: An efficient log-structured key-value storage engine for persistent memory," in *ASPLOS*, 2020.

[8] Y. Chen, Y. Lu, B. Zhu, and J. Shu, "Kernel/user-level collaborative persistent memory file system with efficiency and protection," *arXiv*, 2019.

[9] J. Coburn, A. M. Caulfield, A. Akel, L. M. Grupp, R. K. Gupta, R. Jhala, and S. Swanson, "NV-Heaps: Making persistent objects fast and safe with next-generation, non-volatile memories," in *ASPLOS*, 2011.

[10] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *SoCC*, 2010.

[11] S. R. Dulloor, S. Kumar, A. Keshavamurthy, P. Lantz, D. Reddy, R. Sankaran, and J. Jackson, "System software for persistent memory," in *EuroSys*, 2014.

[12] R. Elmasri and S. B. Navathe, *Fundamentals of Database Systems*. Pearson, 2015.

[13] B. Fitzpatrick, "Distributed caching with Memcached," *Linux Journal*, 2004.

[14] D. Gouk, M. Kwon, J. Zhang, S. Koh, W. Choi, N. S. Kim, M. Kandemir, and M. Jung, “Amber: Enabling precise full-system simulation with detailed modeling of all SSD resources,” in *MICRO*, 2018.

[15] B. Gregg, “Linux perf examples,” 2019. [Online]. Available: http://www.brendangregg.com/perf

[16] D. Hansen, “Allow persistent memory to be used like normal RAM,” 2019. [Online]. Available: https://patchwork.kernel.org/cover/10829019/

[17] S. Haria, M. D. Hill, and M. M. Swift, “MOD: Minimally ordered durable datastructures for persistent memory,” in *ASPLOS*, 2020.

[18] K. Hsieh, S. Khan, N. Vijaykumar, K. K. Chang, A. Boroumand, S. Ghose, and O. Mutlu, “Accelerating pointer chasing in 3D-stacked memory: Challenges, mechanisms, evaluation,” in *ICCD*, 2016.

[19] Intel, “2nd generation Intel® Xeon® Scalable processors with Intel® C620 series chipsets (purley refresh),” 2019.

[20] Intel, “Intel® 64 and IA-32 architectures software developer’s manuals,” 2019.

[21] Intel, “Intel memory latency checker,” 2019.

[22] Intel, “Intel Vtune amplifier,” 2019.

[23] Intel, “Intel® Optane™ DC Persistent Memory,” 2019. [Online]. Available: https://www.intel.com/content/www/us/en/architecture-and-technology/optane-dc-persistent-memory

[24] Intel, “Persistent memory development kit,” 2019. [Online]. Available: https://pmem.io/

[25] Intel, “Pmem-Redis: A version of Redis that uses persistent memory,” 2019. [Online]. Available: https://github.com/pmem/pmem-redis

[26] Intel, “A utility for configuring and managing Intel Optane DC persistent memory modules,” 2019. [Online]. Available: https://github.com/intel/ipmctl

[27] Intel, “Utility library for managing the libnvdimm (non-volatile memory device) sub-system in the Linux kernel,” 2019. [Online]. Available: https://github.com/pmem/ndctl

[28] J. Izraelevitz, J. Yang, L. Zhang, J. Kim, X. Liu, A. Memaripour, Y. J. Soh, Z. Wang, Y. Xu, S. R. Dulloor, J. Zhao, and S. Swanson, “Basic performance measurements of the Intel Optane DC persistent memory module,” *arXiv*, 2019.

[29] R. Kadekodi, S. K. Lee, S. Kashyap, T. Kim, A. Kolli, and V. Chidambaram, “SplitFS: Reducing software overhead in file systems for persistent memory,” in *SOSP*, 2019.

[30] S. Kannan, A. Gavrilovska, K. Schwan, D. Milojicic, and V. Talwar, “Using active NVRAM for I/O staging,” in *PDAC*, 2011.

[31] P. Kennedy, “A close look at Intel Optane DC persistent memory modules,” 2019. [Online]. Available: https://www.servethehome.com/a-close-look-at-intel-optane-dc-persistent-memory-modules/

[32] Y. Kim, W. Yang, and O. Mutlu, “Ramulator: A fast and extensible DRAM simulator,” *CAL*, 2016.

[33] E. Kültürsay, M. Kandemir, A. Sivasubramaniam, and O. Mutlu, “Evaluating STT-RAM as an energy-efficient main memory alternative,” in *ISPASS*, 2013.

[34] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, “Architecting phase change memory as a scalable DRAM alternative,” in *ISCA*, 2009.

[35] S. Lloyd and M. Gokhale, “Evaluating the feasibility of storage class memory as main memory,” in *MEMSYS*, 2016.

[36] A. Margaritov, D. Ustiugov, E. Bugnion, and B. Grot, “Prefetched address translation,” in *MICRO*, 2019.

[37] Micron Technology, Inc., “3D XPoint technology,” 2018. [Online]. Available: https://www.micron.com/products/advanced-solutions/3d-xpoint-technology

[38] Micron Technology, Inc., “DDR4 SDRAM Verilog model,” 2018.

[39] S. Nalli, S. Haria, M. D. Hill, M. M. Swift, H. Volos, and K. Keeton, “An analysis of persistent memory use with WHISPER,” in *ASPLOS*, 2017.

[40] M. A. Ogleari, E. L. Miller, and J. Zhao, “Steal but no force: Efficient hardware undo+redo logging for persistent memory systems,” in *HPCA*, 2018.

[41] O. Patil, L. Ionkov, J. Lee, F. Mueller, and M. Lang, “Performance characterization of a DRAM-NVM hybrid memory architecture for HPC applications using Intel Optane DC persistent memory modules,” in *MEMSYS*, 2019.

[42] I. B. Peng, M. B. Gokhale, and E. W. Green, “System evaluation of the Intel Optane byte-addressable NVM,” in *MEMSYS*, 2019.

[43] P. Pessl, D. Gruss, C. Maurice, M. Schwarz, and S. Mangard, “DRAMA: Exploiting DRAM addressing for cross-CPU attacks,” in *SEC*, 2016.

[44] M. Poremba, T. Zhang, and Y. Xie, “NVMain 2.0: A user-friendly memory simulator to model non-volatile memory systems,” *CAL*, 2015.

[45] J. Ren, J. Zhao, S. Khan, J. Choi, Y. Wu, and O. Mutiu, “ThyNVM: Enabling software-transparent crash consistency in persistent memory systems,” in *MICRO*, 2015.

[46] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, “DRAMSim2: A cycle accurate memory system simulator,” *CAL*, 2011.

[47] A. Roth and G. S. Sohi, “Effective jump-pointer prefetching for linked data structures,” in *ISCA*, 1999.

[48] A. M. Rudoff, “Deprecating the pcommit instruction,” 2016. [Online]. Available: https://software.intel.com/en-us/blogs/2016/09/12/deprecate-pcommit-instruction

[49] ServeTheHome, “Intel Optane DCPMM uth DDR T protocol,” 2019. [Online]. Available: https://www.servethehome.com/2nd-gen-intel-xeon-scalable-launch-cascade-lake-details-and-analysis/intel-optane-dcpmm-uth-ddr-t-protocol/

[50] Standard Performance Evaluation Corporation, “SPEC CPU 2006,” 2019. [Online]. Available: https://www.spec.org/cpu2006/

[51] Standard Performance Evaluation Corporation, “SPEC CPU 2017,” 2019. [Online]. Available: https://www.spec.org/cpu2017/

[52] Y. Sun, T. Baruah, S. A. Mojumder, S. Dong, X. Gong, S. Treadway, Y. Bao, S. Hance, C. McCardwell, V. Zhao, H. Barclay, A. K. Ziabari, Z. Chen, R. Ubal, J. L. Abellán, J. Kim, A. Joshi, and D. Kaeli, “MGPUSim: Enabling multi-GPU performance modeling and optimization,” in *ISCA*, 2019.

[53] K. Suzuki and S. Swanson, “The non-volatile memory technology database (NVMDB),” Department of Computer Science & Engineering, University of California, San Diego, Tech. Rep. CS2015-1011, 2015. [Online]. Available: http://nvmdb.ucsd.edu

[54] E. Vianello, O. Thomas, G. Molas, O. Turkyilmaz, N. Jovanović, D. Garbin, G. Palma, M. Alayan, C. Nguyen, J. Coignus, B. Giraud, T. Benoist, M. Reyboz, A. Toffoli, C. Charpin, F. Clermidy, and L. Perniola, “Resistive memories for ultra-low-power embedded computing design,” in *IEDM*, 2014.

[55] V. Viswanathan, “Disclosure of hardware prefetcher control on some intel processors,” 2014. [Online]. Available: https://software.intel.com/en-us/articles/disclosure-of-hw-prefetcher-control-on-some-intel-processors

[56] H. Volos, G. Magalhaes, L. Cherkasova, and J. Li, “Quartz: A lightweight performance emulator for persistent memory software,” in *Middleware*, 2015.

[57] H. Volos, A. J. Tack, and M. M. Swift, “Mnemosyne: Lightweight persistent memory,” in *ASPLOS*, 2011.

[58] D. Waddington, M. Kunitomi, C. Dickey, S. Rao, A. Abboud, and J. Tran, “Evaluation of Intel 3D-Xpoint NVDIMM technology for memory-intensive genomic workloads,” in *MEMSYS*, 2019.

[59] M. Weiland, H. Brunst, T. Quintino, N. Johnson, O. Iffrig, S. Smart, C. Herold, A. Bonanni, A. Jackson, and M. Parsons, “An early evaluation of Intel’s Optane DC persistent memory module and its impact on high-performance scientific applications,” in *SC*, 2019.

[60] WikiChip, “Cascade Lake - microarchitectures - Intel,” 2020.

[61] H. . P. Wong, H. Lee, S. Yu, Y. Chen, Y. Wu, P. Chen, B. Lee, F. T. Chen, and M. Tsai, “Metal-oxide RRAM,” *IEEE*, 2012.

[62] J. Xu, J. Kim, A. Memaripour, and S. Swanson, “Finding and fixing performance pathologies in persistent memory software stacks,” in *ASPLOS*, 2019.

[63] J. Xu and S. Swanson, “NOVA: A log-structured file system for hybrid volatile/non-volatile main memories,” in *FAST*, 2016.

[64] J. Yang, J. Izraelevitz, and S. Swanson, “Orion: A distributed file system for non-volatile main memory and RDMA-capable networks,” in *FAST*, 2019.

[65] J. Yang, J. Kim, M. Hoseinzadeh, J. Izraelevitz, and S. Swanson, “An empirical guide to the behavior and use of scalable persistent memory,” in *FAST*, 2020.

[66] V. Young, Z. A. Chishti, and M. K. Qureshi, “TicToc: Enabling bandwidth-efficient DRAM caching for both hits and misses in hybrid memory systems,” in *ICCD*, 2019.

[67] L. Zhang and S. Swanson, “Pangolin: A fault-tolerant persistent memory programming library,” in *ATC*, 2019.

[68] J. Zhao, S. Li, D. H. Yoon, Y. Xie, and N. P. Jouppi, “Kiln: Closing the performance gap between systems with and without persistence support,” in *MICRO*, 2013.

[69] J. Zhao, O. Mutlu, and Y. Xie, “FIRM: Fair and high-performance memory control for persistent memory systems,” in *MICRO*, 2014.

[70] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, “A durable and energy efficient main memory using phase change memory technology,” in *ISCA*, 2009.