An Empirical Guide to the Behavior and Use of Scalable Persistent Memory

Jian Yang, Juno Kim, and Morteza Hoseinzadeh, *UC San Diego;*
Joseph Izraelevitz, *University of Colorado, Boulder;* Steve Swanson, *UC San Diego*

https://www.usenix.org/conference/fast20/presentation/yang

This paper is included in the Proceedings of the
18th USENIX Conference on File and
Storage Technologies (FAST '20)

February 25–27, 2020 • Santa Clara, CA, USA

978-1-939133-12-0

An Empirical Guide to the Behavior and Use of Scalable Persistent Memory

Jian Yang[*][†], Juno Kim[†], Morteza Hoseinzadeh[†], Joseph Izraelevitz[§], and Steven Swanson[†]

{jianyang, juno, mhoseinzadeh, swanson}@eng.ucsd.edu[†] joseph.izraelevitz@colorado.edu[§]
[†]UC San Diego [§]University of Colorado, Boulder

Abstract

After nearly a decade of anticipation, scalable nonvolatile memory DIMMs are finally commercially available with the release of Intel's Optane DIMM. This new nonvolatile DIMM supports byte-granularity accesses with access times on the order of DRAM, while also providing data storage that survives power outages.

Researchers have not idly waited for real nonvolatile DIMMs (NVDIMMs) to arrive. Over the past decade, they have written a slew of papers proposing new programming models, file systems, libraries, and applications built to exploit the performance and flexibility that NVDIMMs promised to deliver. Those papers drew conclusions and made design decisions without detailed knowledge of how real NVDIMMs would behave or how industry would integrate them into computer architectures. Now that Optane NVDIMMs are actually here, we can provide detailed performance numbers, concrete guidance for programmers on these systems, reevaluate prior art for performance, and reoptimize persistent memory software for the real Optane DIMM.

In this paper, we explore the performance properties and characteristics of Intel's new Optane DIMM at the micro and macro level. First, we investigate the basic characteristics of the device, taking special note of the particular ways in which its performance is peculiar relative to traditional DRAM or other past methods used to emulate NVM. From these observations, we recommend a set of best practices to maximize the performance of the device. With our improved understanding, we then explore and reoptimize the performance of prior art in application-level software for persistent memory.

1 Introduction

Over the past ten years, researchers have been anticipating the arrival of commercially available, scalable non-volatile main memory (NVMM) technologies that provide "byte-addressable" storage that survives power outages. With the arrival of Intel's Optane DC Persistent Memory Module (which we refer to as Optane DIMMs), we can start to understand the real capabilities, limitations, and characteristics of these memories and start designing systems to fully leverage them.

We have characterized the performance and behavior of Optane DIMMs using a wide range of microbenchmarks, benchmarks, and applications. The data we have collected demonstrate that many of the assumptions that researchers have made about how NVDIMMs would behave and perform are incorrect. The widely expressed expectation was that NVDIMMs would have behavior that was broadly similar to DRAM-based DIMMs but with lower performance (i.e., higher latency and lower bandwidth). These assumptions are reflected in the methodology that research studies used to emulate NVDIMMs, which include specialized hardware platforms [21], software emulation mechanisms [12,32,36,43,47], exploiting NUMA effects [19,20,29], and simply pretending DRAM is persistent [8,9,38].

We have found the actual behavior of Optane DIMMs to be more complicated and nuanced than the "slower, persistent DRAM" label would suggest. Optane DIMM performance is much more strongly dependent on access size, access type (read vs. write), pattern, and degree of concurrency than DRAM performance. Furthermore, Optane DIMM's persistence, combined with the architectural support that Intel's latest processors provide, leads to a wider range of design choices for software designers.

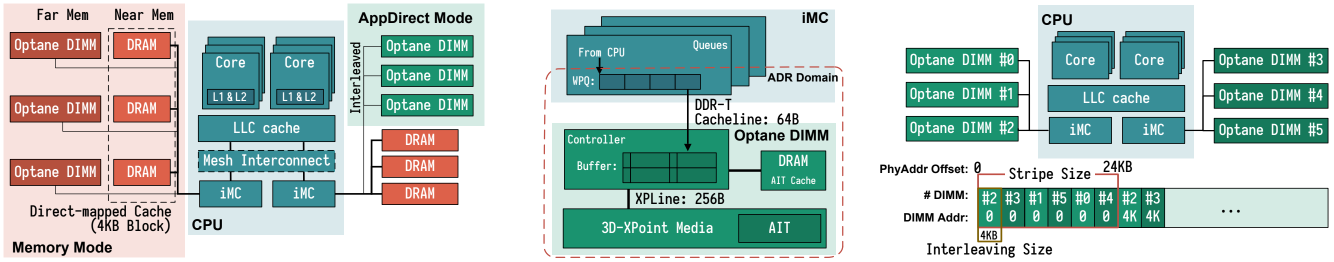
This paper presents a detailed evaluation of the behavior and performance of Optane DIMMs on microbenchmarks and applications and provides concrete, actionable guidelines for how programmers should tune their programs to make the best use of these new memories. We describe these guidelines, explore their consequences, and demonstrate their utility by using them to guide the optimization of several NVMM-aware software packages, noting that prior methods of emulation have been unreliable.

The paper proceeds as follows. Section 2 provides architectural details on our test machine and the Optane DIMM. Section 3 presents experiments on basic microarchitectural parameters, and Section 4 focuses on how Optane DIMM is different from DRAM and other emulation techniques. Section 5 uses these results to posit best practices for programmers on the Optane DIMM. In this section, we first justify each guideline with a microbenchmark demonstrating the root cause. We then present one or more case studies where guideline influences a previously proposed Optane-aware software system. Section 6 provides discussion as to how our guidelines extend to future generations of NVM. Section 7 describes related work in this space, and Section 8 concludes.

2 Background and Methodology

In this section, we provide background on Intel's Optane DIMM, describe the test system, and then describe the configurations we use throughout the rest of the paper.

[*] Now at Google



(a) Optane Platform Modes (Memory and AppDirect)

(b) Optane DIMM Overview

(c) Interleaved Optane DIMMs

Figure 1: **Overview of (a) Optane platform, (b) Optane DIMM and (c) how Optane memories interleave across channels** Optane DIMM can work as either volatile far memory with DRAM as cache (memory mode), or persistent memory with DRAM as main memory (AppDirect mode).

2.1 Optane Memory

The Optane DIMM is the first scalable, commercially available NVDIMM. Compared to existing storage devices (including the Optane SSDs) that connect to an external interface such as PCIe, the Optane DIMM has lower latency, higher read bandwidth, and presents a memory address-based interface instead of a block-based NVMe interface. Compared to DRAM, it has higher density and persistence. At its debut, the Optane DIMM is available in three different capacities: 128, 256, and 512 GB.

2.1.1 Intel's Optane DIMM

Like traditional DRAM DIMMs, the Optane DIMM sits on the memory bus, and connects to the processor's integrated memory controller (iMC) (Figure 1(a)). Intel's Cascade Lake processors are the first (and only) CPUs to support Optane DIMM. On this platform, each processor contains one or two processor dies which comprise separate NUMA nodes. Each processor die has two iMCs, and each iMC supports three channels. Therefore, in total, a processor die can support a total of six Optane DIMMs across its two iMCs.

To ensure persistence, the iMC sits within the *asynchronous DRAM refresh (ADR)* domain — Intel's ADR feature ensures that CPU stores that reach the ADR domain will survive a power failure (i.e., will be flushed to the NVDIMM within the hold-up time, $< 100\ \mu s$) [42]. The iMC maintains read and write pending queues (RPQs and WPQs) for each of the Optane DIMMs (Figure 1(b)), and the ADR domain includes WPQs. Once data reaches the WPQs, the ADR ensures that the iMC will flush the updates to 3D-XPoint media on power failure. The ADR domain does not include the processor caches, so stores are only persistent once they reach the WPQs.

The iMC communicates with the Optane DIMM using the DDR-T interface in cache-line (64-byte) granularity, which shares a mechanical and electrical interface with DDR4 but uses a different protocol that allows for asynchronous command and data timing.

Memory accesses to the NVDIMM (Figure 1(b)) arrive first at the on-DIMM controller (referred as *XPController* in this paper), which coordinates access to the Optane media. Similar to SSDs, the Optane DIMM performs an internal address translation for wear-leveling and bad-block management, and maintains an *address indirection table* (AIT) for this translation [7].

After address translation, the actual access to storage media occurs. As the 3D-XPoint physical media access granularity is 256 bytes (referred as *XPLine* in this paper), the XPController translates smaller requests into larger 256-byte accesses, causing write amplification as small stores become read-modify-write operations. The XPController has a small write-combining buffer (referred as *XPBuffer* in this paper), to merge adjacent writes.

It is important to note that all updates that reach the XPBuffer are already persistent since XPBuffer resides within the ADR. Consequently, the NVDIMM can buffer and merge updates regardless of ordering requirements that the program specifies with memory fences.

2.1.2 Operation Modes

Optane DIMMs can operate in two modes (Figure 1(a)): Memory and App Direct.

Memory mode uses Optane to expand main memory capacity without persistence. It combines an Optane DIMM with a conventional DRAM DIMM on the same memory channel that serves as a direct-mapped cache for the NVDIMM. The cache block size is 64 B, and the CPU's memory controller manages the cache transparently. The CPU and operating system simply see the Optane DIMM as a larger (volatile) portion of main memory.

App Direct mode provides persistence and does not use a DRAM cache. The Optane DIMM appears as a separate, persistent memory device. A file system or other management layer provides allocation, naming, and access to persistent data.

In both modes, Optane memory can be (optionally) interleaved across channels and DIMMs (Figure 1(c)). On our platform, the only supported interleaving size is 4 kB, which ensures that accesses to a single page fall into a single DIMM. With six DIMMs, an access larger than 24 kB will access all the DIMMs.

2.1.3 ISA Support

In App Direct mode, applications and file systems can access the Optane DIMMs with CPU instructions. The extended Instruction Set Architecture (ISA) offers programmers a number of options to control store ordering.

Applications access the Optane DIMM's content using store instructions, and those stores will, eventually, become persistent. The cache hierarchy, however, can reorder stores, making recovery after a crash challenging [12, 28, 33, 40, 49]. Current Intel ISA provides `clflush` and `clflushopt` instructions to flush cache lines back to memory with `clflushopt` having weaker ordering constraints, and `clwb` can write back (but not evict) cache lines. Alternatively, software can use non-temporal stores (e.g., `ntstore`) to bypass the cache hierarchy and write directly to memory. All these instructions are non-blocking, so the program must issue an `sfence` to ensure that a previous cache flush, cache write back, or non-temporal store is complete and persistent.

2.2 System Description

We performed our experiments on a dual-socket evaluation platform provided by Intel Corporation. The CPUs are 24-core Cascade Lake engineering samples with the similar spec as the previous-generation Xeon Platinum 8160. Each CPU has two iMCs and six memory channels (three channels per iMC). A 32 GB Micron DDR4 DIMM and a 256 GB Intel Optane DIMM are attached to each of the memory channels. Thus the system has 384 GB (2 socket × 6 channel × 32 GB/DIMM) of DRAM, and 3 TB (2 socket × 6 channel × 256 GB/DIMM) of NVMM. Our machine runs Fedora 27 with Linux kernel version 4.13.0 built from source.

2.3 Experimental Configurations

As the Optane DIMM is both persistent and byte-addressable, it can fill the role of either a main memory device (i.e., replacing DRAM) or a persistent device (i.e., replacing disk). In this paper, we focus on the persistent usage, and defer discussion on how our results apply to using Optane DIMM as volatile memory to Section 6.

Linux manages persistent memory by creating `pmem` namespaces over a contiguous span of physical memory. A namepace can be backed by interleaved or non-interleaved Optane memory, or emulated persistent memory backed by DRAM. In this study, we configure Optane memory in App Direct mode and create a namespace for each type of memory.

Our baseline (referred as *Optane*) exposes six Optane DIMMs from the same socket as a single interleaved namespace. In our experiments, we used local accesses (i.e., from the same NUMA node) as the baseline to compare with one or more other configurations, such as access to Optane memory on the remote socket (*Optane-Remote*) or DRAM on the local or remote socket (*DRAM* and *DRAM-Remote*). To better understand the raw performance of Optane memory without interleaving, we also create a namespace consisting of a single Optane DIMM and denote it as *Optane-NI*.

3 Performance Characterization

In this section, we measure Optane's performance along multiple axes to provide the intuition and data that programmers and system designers will need to effectively utilize Optane. We find that Optane's performance characteristics are surprising in many ways, and more complex than the common assumption that Optane behaves like slightly-slower DRAM.

3.1 LATTester

Characterizing Optane memory is challenging for two reasons. First, the underlying technology has major differences from DRAM but publicly-available documentation is scarce. Secondly, existing tools measure memory performance primarily as a function of locality and access size, but we have found that Optane performance depends strongly on memory interleaving and concurrency as well. Persistence adds an additional layer of complexity for performance measurements.

To fully understand the behavior of the Optane memory, we built a microbenchmark toolkit called LATTester. To accurately measure the CPU cycle count and minimize the impact from the virtual memory system, LATTester runs as a dummy file system in the kernel and accesses pre-populated (i.e., no page-faults) kernel virtual addresses of Optane DIMMs. LATTester also pins the kernel threads to fixed CPU cores and disables IRQ and cache prefetcher. In addition to simple latency and bandwidth measurements, LATTester collects a large set of hardware counters at both the CPU and NVDIMM.

Our investigation of Optane memory behavior proceeds in two phases. First, we performed a broad, systematic "sweep" over Optane configuration parameters including access patterns (random vs. sequential), operations (loads, stores, fences, etc.), access size, stride size, power budget, NUMA configuration, and address space interleaving. Using this data, we designed targeted experiments to investigate anomalies and verify or disprove our hypotheses about the underlying causes. Between our initial sweep and the follow-up tests, we collected over ten thousand data points. The program and dataset are available at https://github.com/NVSL/OptaneStudy.

3.2 Typical Latency

Read and write latencies are key memory technology parameters. We measured read latency by timing the average latency

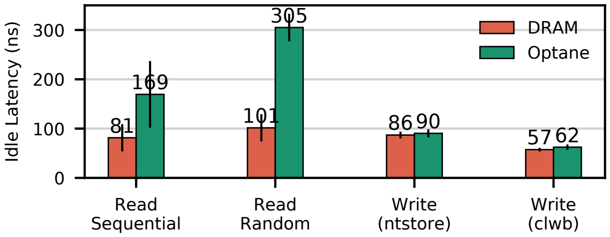


Figure 2: **Best-case latency** An experiment showing random and sequential read latency, as well as write latency using cached write with `clwb` and `ntstore` instructions. Error bars show one standard deviation.

for individual 8-byte load instructions to sequential and random memory addresses. To eliminate caching and queuing effects, we empty the CPU pipeline and issue a memory fence (`mfence`) between measurements (`mfence` serves the purpose of serialization for reading timestamps). For writes, we load the cache line into the cache and then measure the latency of one of two instruction sequences: a 64-bit store, a `clwb`, and an `mfence`; or a non-temporal store followed by an `mfence`.

These measurements reflect the load and store latency as seen by software rather than those of these underlying memory devices. For loads, the latency includes the delay from the on-chip interconnect, iMC, XPController and the actual 3D-XPoint media. Our results (Figure 2) show the read latency for Optane is 2×–3× higher than DRAM. We believe most of this difference is due to Optane's longer media latency. Optane memory is also more pattern-dependent than DRAM. The random-vs-sequential gap is 20% for DRAM but 80% for Optane memory, and this gap is a consequence of the XPBuffer. For stores, the memory store and fence instructions commit once the data reaches the ADR at the iMC, so both DRAM and Optane show a similar latency. Non-temporal stores are more expensive than writes with cache flushes (`clwb`).

In general, the latency variance for Optane is extremely small, save for an extremely small number of "outliers", which we investigate in the next section. The sequential read latencies for Optane DIMMs have higher variances, as the first cache line access loads the entire XPLine into XPBuffer, and the following three accesses read data in the buffer.

3.3 Tail Latency

Memory and storage system tail latency critically affects response times and worst-case behavior in many systems. In our tests, we observed a very consistent latency for loads and stores except a few "outliers", which increase in number for stores when accesses are concentrated in a "hot spot".

Figure 3 measures the relationship between tail latency and access locality. The graph shows the 99.9th, 99.99th, and maximal latencies as a function of hot spot size. We allocate a circular buffer with each hot spot size, and use a single thread to issue 20 million 64-byte writes. The number of outliers (especially for the ones over 50μs) reduces as the hotspot size increases and do not exist for DRAM. These spikes are rare (0.006% of the accesses), but their latency are 2 orders of magnitude higher than a common case Optane access. We suspect this effect is due to remapping for wear-leveling or thermal concerns, but we cannot be sure.

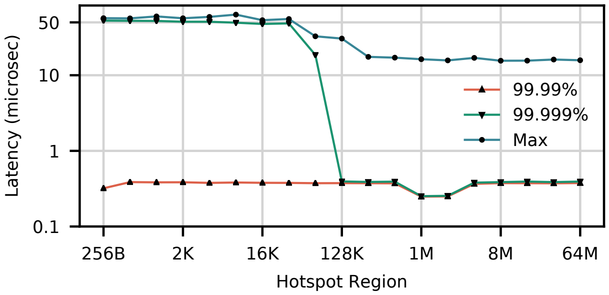


Figure 3: **Tail latency** An experiment showing the tail latency of writing to a small area of memory (hotspot) sequentially. Optane memory has rare "outliers" where a small number of writes take up to 50 μs to complete (an increase of 100× over the usual latency).

3.4 Bandwidth

Detailed bandwidth measurements are useful to application designers as they provide insight into how a memory technology will impact overall system throughput. First, we measured Optane and DRAM bandwidth for random and sequential reads and writes under different levels of concurrency.

Figure 4 shows the bandwidth achieved at different thread counts for sequential accesses with 256 B access granularity. We show loads and stores (`Write(ntstore)`), as well as cached writes with flushes (`Write(clwb)`). All experiments use AVX-512 instructions. The left-most graph plots performance for interleaved DRAM, while the center and right-most graphs plot performance for non-interleaved and interleaved Optane. In the non-interleaved measurements all accesses hit a single DIMM.

Figure 5 shows how performance varies with access size. The graphs plot aggregate bandwidth for random accesses of a given size. We use the best-performing thread count for each curve (given as "<load thread count>/<`ntstore` thread count>/<`store+clwb` thread count>" in the figure). Note that the best performing thread count for Optane(Read) varies with different access sizes for random accesses, where 16 threads show good performance consistently.

The data shows that DRAM bandwidth is both higher than Optane and scales predictably (and monotonically) with thread count until it saturates the DRAM's bandwidth, which is mostly independent of access size.

The results for Optane are wildly different. First, for a single DIMM, the maximal read bandwidth is 2.9× of the maxi-

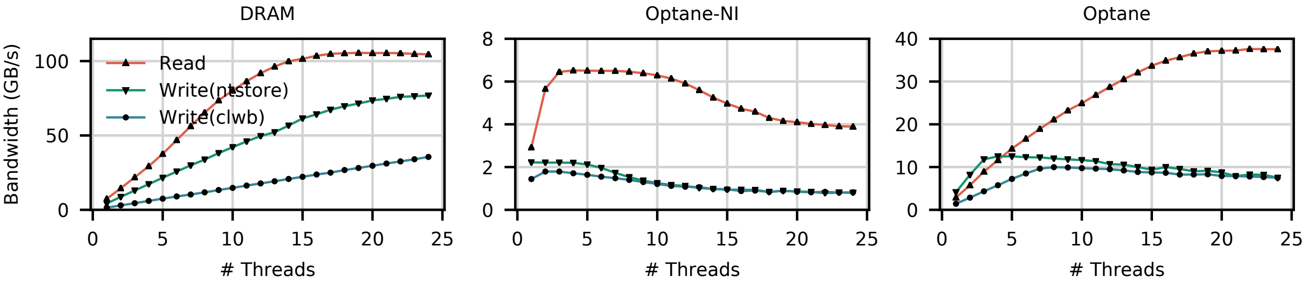


Figure 4: **Bandwidth vs. thread count** An experiment showing the maximal bandwidth as thread count increases (from left to right) on local DRAM, non-interleaved and interleaved Optane memory. All threads use a 256 B access size. (Note the difference in vertical scales).

mal write bandwidth (6.6 GB/s and 2.3 GB/s, respectively), where DRAM has a smaller gap (1.3×) between read and write bandwidth. Second, with the exception of interleaved reads, Optane performance is non-monotonic with increasing thread count. For the non-interleaved (i.e., single-DIMM) cases, performance peaks at between one and four threads and then tails off. Interleaving pushes the peak to twelve threads for `store+clwb`. We will return to the negative impact of rising thread count on performance in Section 5.1. Third, Optane bandwidth for random accesses under 256 B is poor. This "knee" corresponds to XPLine size. DRAM bandwidth does not exhibit a similar "knee" at 8 kB (the typical DRAM page size), because the cost of opening a page of DRAM is much lower than accessing a new page of Optane.

Interleaving (which spreads accesses across all six DIMMs) adds further complexity: Figure 4 (right) and Figure 5 (right) measure bandwidth across six interleaved NVDIMMs. Interleaving improves peak read and write bandwidth by 5.8× and 5.6×, respectively. These speedups match the number of DIMMs in the system and highlight the per-DIMM bandwidth limitations of Optane. The most striking feature of the graph is a dip in performance at 4 kB — this dip is an emergent effect caused by contention at the iMC, and it is maximized when threads perform random accesses close to the interleaving size. We further discuss this issue in Section 5.3.

4 Comparison to Emulation

Non-volatile memory research has been popular in recent years (e.g. [9, 18, 28, 31, 32, 36, 38, 40, 43, 49, 51, 54, 56, 60]). However, since scalable NVDIMMs have not been available, most of the NVM based systems have been evaluated on emulated NVM. Common ways to emulate NVM include adding delays to memory accesses in software [32, 36, 49], using software emulators [43,47], using software simulation [12, 31, 40], using hardware emulators such as Intel's Persistent Memory Emulator Platform (PMEP) [21] to limit latency and bandwidth [54, 55, 60], using DRAM on a remote socket (DRAM-Remote) [19, 20, 29], underclocking DRAM [28] or just using plain DRAM [9, 34, 38, 56] or battery-backed DRAM [18]. Below, we compare these emulation techniques to real Optane using microbenchmarks and then provide a case study in how those differences can affect research results.

4.1 Microbenchmarks in Emulation

Figure 6 (left) shows the write latency/bandwidth curves for NVM emulation mechanisms (e.g. PMEP, DRAM-Remote, DRAM) in comparison to real Optane memory. Figure 6 (right) shows the bandwidth with respect to the number of reader/writer threads (all experiments use a fixed number of threads that give maximum bandwidth). Our PMEP configuration adds a 300 ns latency on load instructions and throttles write bandwidth at 1/8 of DRAM bandwidth as this configuration is the standard used in previous works [54, 55, 59, 60]. Note that PMEP is a specialized hardware platform, so its performance numbers are not directly comparable to the system we used in our other experiments.

The data in these figures shows that none of the emulation mechanisms captures the details of Optane's behavior — all methods deviate drastically from real Optane memory. They fail to capture Optane memory's preference for sequential accesses and read/write asymmetry, and give wildly inaccurate guesses for device latency and bandwidth.

4.2 Case Study: Optimizing RocksDB

The lack of emulation fidelity can have a dramatic effect on the performance of software. Results may be misleading, especially those based on simple DRAM. In this section, we revisit prior art in NVM programming in order to demonstrate that emulation is generally insufficient to capture the performance of Optane DIMMs and that future work should be validated on real hardware.

RocksDB [22] is a high-performance embedded key-value store, designed by Facebook and inspired by Google's LevelDB [16]. RocksDB's design is centered around the log-structured merge tree (LSM-tree), designed for block-based storage devices, which absorbs random writes and converts them to sequential writes to maximize disk bandwidth.

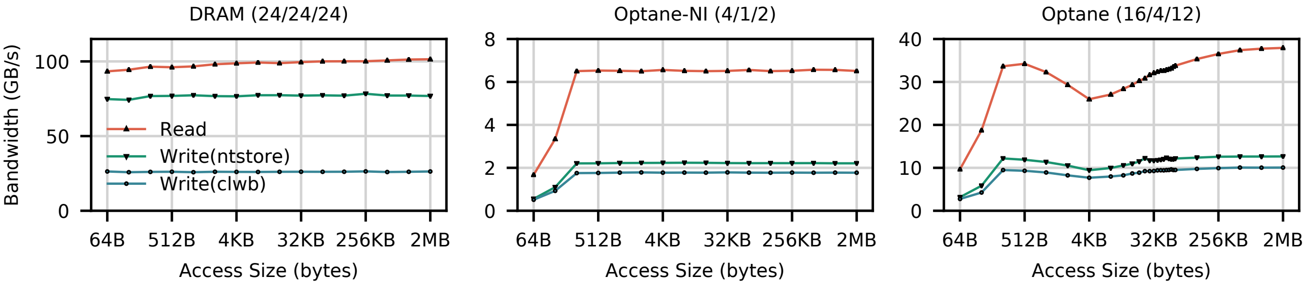


Figure 5: **Bandwidth over access size** An experiment showing maximal bandwidth over different access sizes on (from left to right) local DRAM, interleaved and non-interleaved Optane memory. Graph titles include the number of threads used in each experiment (`Read/Write(ntstore)/Write(clwb)`).

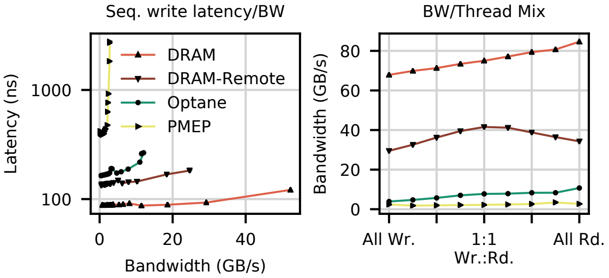


Figure 6: **Microbenchmarks under emulation** The emulation mechanisms used to evaluate many projects do not accurately capture the complexities of Optane performance.

A recent study [53] compared two strategies for adapting RocksDB to use persistent memory. The first used a fine-grained persistence approach to migrate RockDB's "memtable" to persistent memory, eliminating the need for a file-based write-ahead log. The second approach moved the write-ahead log to persistent memory and used a simpler acceleration technique called FLEX to improve logging performance. The study used DRAM as a stand-in for Optane, and found that fine-grained persistence offered 19% better performance.

We replicated these experiments on real Optane DIMMs. We used the RocksDB test `db_bench` on SET throughput with 20-byte key size and 100-byte value size, and sync'ed the database after each SET operation; the results are shown in Figure 7. With real Optane, the result is the opposite: FLEX performs better than fine-grained persistence by 10%. These results are not surprising given Optane memory's preference for sequential accesses and its problem with small random writes.

5 Best Practices for Optane DIMMs

Section 3 highlights the many differences between Optane and conventional storage and memory technologies, and Section 4 shows how these differences can manifest to invalidate macro-level results. These differences mean that existing intuitions

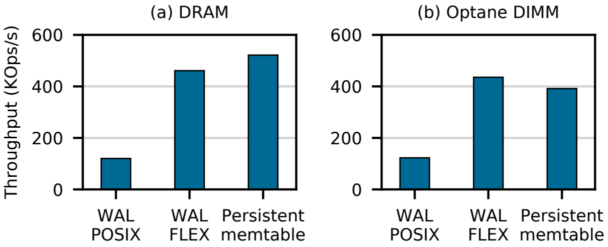


Figure 7: **Migrating RocksDB to Optane Memory** Optane memory is sufficiently different from DRAM to invert prior conclusions. Using a persistent memtable works best for DRAM emulating persistent memory, but on real Optane memory the conclusion is reversed.

about how to optimize software for disks and memory do not apply directly to Optane. This section distills the results of our characterization experiments into a set of four principles for how to build and tune Optane-based systems.

1. **Avoid random accesses smaller than 256 B.**

2. **Use non-temporal stores when possible for large transfers, and control cache evictions.**

3. **Limit the number of concurrent threads accessing an Optane DIMM.**

4. **Avoid NUMA accesses (especially read-modify-write sequences).**

Below, we describe the guidelines in detail, give examples on how to implement them, and provide case studies in their application.

5.1 Avoid small random accesses

Internally, Optane DIMMs update Optane contents at a 256 B granularity. This granularity, combined with a large internal store latency, means that smaller updates are inefficient since

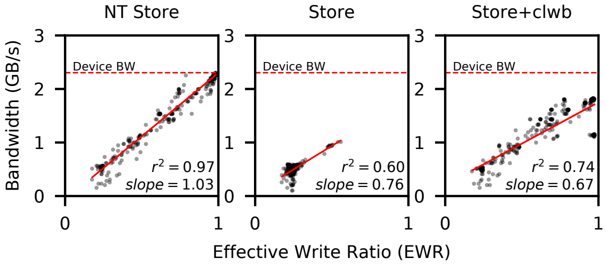


Figure 8: **Relationship between EWR and throughput on a single DIMM** Each dot represents an experiment with different access size, thread count and power budget configurations. Note the correlation between the metrics.

they require the DIMM to perform an internal read-modify-write operation causing write amplification. The less locality the accesses exhibit, the more severe the performance impact.

To characterize the impact of small stores, we performed two experiments. First, we quantify the inefficiency of small stores using a metric we have found useful in our study of Optane DIMMs. The *Effective Write Ratio(EWR)* is the ratio of bytes issued by the iMC divided by the number of bytes actually written to the 3D-XPoint media (as measured by the DIMM's hardware counters). EWR is the inverse of write amplification. EWR values below one indicate the Optane DIMM is operating inefficiently since it is writing more data internally than the application requested. The EWR can also be greater than one, due to write-combining at the XPBuffer. In Memory Mode, DRAM caching may also introduce a higher EWR.

Figure 8 plots the strong correlation between EWR and effective device bandwidth for a single DIMM for all the measurements in our systematic sweep of Optane performance. Based on this relationship, we conclude that working to maximize EWR is a good way to maximize bandwidth.

In general, small stores exhibit EWR's less than one. For example, when using a single thread to perform non-temporal stores to random accesses, it achieves an EWR of 0.25 for 64-byte accesses and 0.98 for 256-byte accesses.

Notably, 256-byte updates are efficient, even though the iMC only issues 64 B to accesses the DIMM — the XPBuffer is responsible for buffering and combining 64 B accesses into 256 B internal writes. As a consequence, Optane DIMMs can efficiently handle small stores, if they exhibit sufficient locality. To understand how much locality is sufficient, we crafted an experiment to measure the size of the XPBuffer. First, we allocate a contiguous region of N XPLines. During each "round" of the experiment, we first update the first half (128 B) of each XPLine in turn. Then, we update the second half of each XPLine. We measured the EWR for each round. Figure 9 shows the results. Below $N = 64$ (that is, a region size of 16 kB), the EWR is near unity, suggesting that the accesses to the second halves are hitting in the XPBuffer. Above $N = 64$, write amplification jumps, indicating a sharp rise in the miss rate. This result implies the XPBuffer is approximately 16 kB in size. Further experiments demonstrate that reads also compete for space in the XPBuffer.

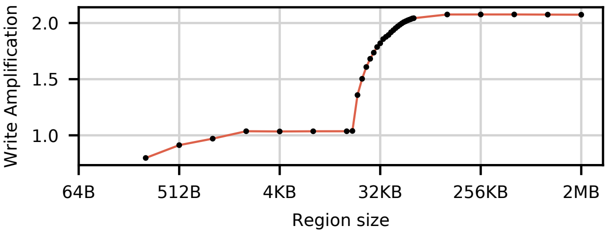


Figure 9: **Inferring XPBuffer capacity** The data shows that the Optane DIMM can use the XPBuffer to coalesce writes spread across up to 64 XPLines.

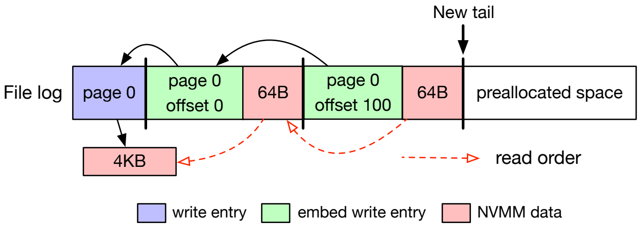


Figure 10: **NOVA-datalog mechanism** Sequentially embedding data along with metadata turns random writes into sequential writes. This figure illustrates how NOVA-datalog appends two 64 B random writes (at 0 and 100, respectively) into the log of a 4 kB file.

Together these results provide a specific guidance for maximizing Optane store efficiency: Avoid small stores, but if that is not possible, limit the working set to 16 kB per Optane DIMM.

5.1.1 Case Study: RocksDB

The correlation between EWR bandwidth explains the results for RocksDB seen in Section 4 and Figure 7. The persistent memtable resulted in many small stores with poor locality, leading to a low EWR of 0.434. In contrast, the FLEX-based optimization of WAL uses sequential (and larger) stores, resulting in an EWR of 0.999.

5.1.2 Case Study: The NOVA filesystem

NOVA [54, 55] is a log-structured, NVMM file system that maintains a separate log for each file and directory, and uses copy-on-write for file data updates to ensure data consistency. The original NOVA studies used emulated NVMM for their evaluations, so NOVA has not been tuned for Optane.

The original NOVA design has two characteristics that degrade performance on Optane. First, the log entries NOVA appends for each metadata update are small – 40-64 B, and since NOVA uses many logs, log updates exhibit little locality,

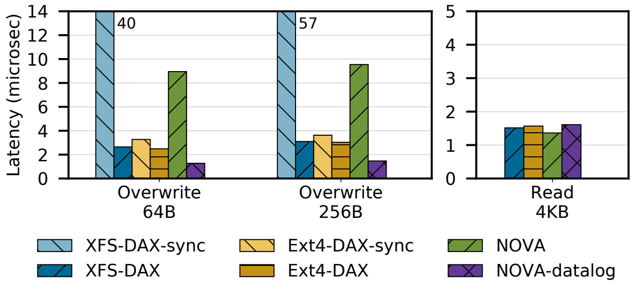


Figure 11: **File IO latency** NOVA-datalog significantly speeds up small random writes, but adds a slight overhead in the read path. Like NOVA and unlike `Ext4` or `XFS`, NOVA-datalog still provides data consistency.

especially when the file system is under load. Second, NOVA uses copy-on-write to 4 kB pages for file data updates, resulting in useless stores. This inefficiency occurs regardless of the underlying memory technology, but Optane's poor store performance exacerbates its effect.

We address both problems by increasing the size of log entries and avoiding some copy-on-write operations. Our modified version of NOVA – *NOVA-datalog* – embeds write data for sub-page writes into the log (Figure 10). Unlike the log's normal *write entry*, which contains a pointer to a new copy-on-write 4 kB page and its offset within the file, an *embed write entry* contains a page offset, an address within the page, and is followed by the actual contents of the write. This optimization requires several subsidiary changes to the original NOVA design. In particular, NOVA must merge sub-page updates into the target page before memory-mapping or reading the file.

Figure 11 shows the latencies of random overwrites and reads for three file systems with different modes. For `XFS-DAX` and `Ext4-DAX` (the two NVM-based file systems included in Linux), we measured both normal `write` and `write` followed by `fsync` (labeled with "-sync"). NOVA-datalog improves write performance significantly compared to the original design (by 7×, 6.5× for 64 byte, 256 byte writes, respectively) and meets (for 256 B) or exceeds (for 64 B) performance for the other file systems (which do not provide data consistency). Read latency increases slightly compared to the original NOVA. EWR measurements generally mirror the performance gains.

5.2 Use non-temporal stores for large writes

The choice of how programs perform and order updates to Optane has a large impact on performance. When writing to persistent memory, programmers have several options. After a regular `store`, programmers can either evict (`clflush`, `clflushopt`) or write back (`clwb`) the cache line to move the data into the ADR and eventually the Optane DIMM. Alternatively, the `ntstore` instruction writes directly to persistent memory, bypassing the cache hierarchy. For all these instructions, a subsequent `sfence` ensures that the effects of prior evictions, write backs, and non-temporal stores are persistent.

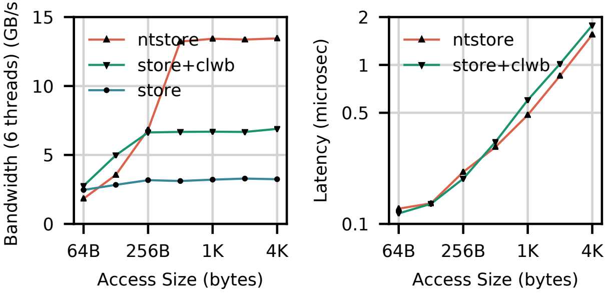


Figure 12: **Performance achievable with persistence instructions** Flush instructions have lower latency for small accesses, but `ntstore` has better latency for larger accesses. Using `ntstore` also avoids an additional read of the cache line from Optane memory, resulting in higher bandwidth.

In Figure 12, we compared achieved bandwidth (left) and latency (right) for sequential accesses using AVX-512 stores with three different instruction sequences: `ntstore`, `store + clwb`, and `store`, followed by a `sfence`. Our bandwidth test used six threads as it gives good results for all instructions. The data show that flushing after each 64 B store improves the bandwidth for accesses larger than 64 B. We believe this occurs because letting the cache naturally evict cache lines adds nondeterminism to the access stream that reaches the Optane DIMM. Proactively cleaning the cache ensures that accesses remain sequential. The EWR correlates this hypothesis: Adding flushes increases EWR from 0.26 to 0.98.

The data also shows that non-temporal stores have lower latency than `store + clwb` for accesses over 512 B. Non-temporal stores also have highest bandwidth for accesses over 256 B. Here, the performance boost is due to the fact that a `store + clwb` must load the cache line into the CPU's local cache before executing store, thereby using up some of the Optane DIMMs bandwidth. As `ntstores` bypass the cache, they will avoid this extraneous read and can achieve higher bandwidth.

In Figure 13, we show how `sfences` affect performance. We used a single thread to issue sequential writes of different sizes on Optane-NI. We issued `clwb` during the write of each cache line (every 64B), or after the entire write (write size). At the end of the write we issued a single `sfence` to ensure the entire write is persistent (we call this entire operaton an "`sfence` interval"). The result shows the bandwidth peaks when the write size is 256 B. This peak is a consequence of the semantics of `clflushopt` which is tuned for moderately sized writes [1]. Flushing during or after a medium sized write (beyond 1 kB) does not affect the bandwidth, but when the write size is over 8 MB, flushing after the write causes performance degradation as we incurred cache capacity invalidations and a higher EWR.

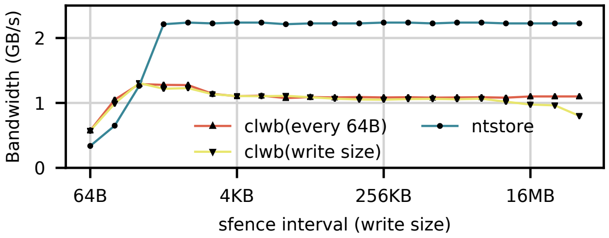


Figure 13: **Bandwidth over `sfence` intervals.** The bandwidth of Optane memory decreases when `sfence` interval increases, causing implicit cache evictions.

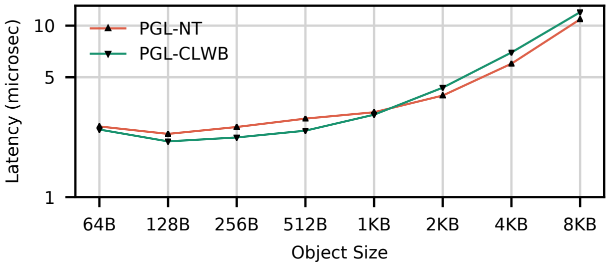


Figure 14: **Persistence instructions for micro-buffering** Using `ntstores` for large writes, and using `clwb` for small ones can improve performance even at macro-level. (Y-axis is in log scale)

5.2.1 Case Study: Micro-buffering for PMDK

Our analysis of the relative merits of non-temporal versus normal stores provides an opportunity to optimize existing work. For example, recent work proposed the "micro-buffering" [58] technique for transactionally updating persistent objects. That work modified the Intel's PMDK [14] transactional persistent object library to copy objects from Optane to DRAM at the start of a transaction rather than issuing loads directly to Optane. On transaction commit, it writes back the entire object at once using non-temporal stores.

The original paper only used non-temporal stores, but our analysis suggests micro-buffering would perform better if it used normal stores for small objects as long as it flushed the affected cache lines immediately after updating them. Figure 14 compares the latency of a no-op transaction for objects of various sizes for unmodified PMDK and micro-buffering with non-temporal and normal store-based write back. The crossover between normal stores and non-temporal stores for micro-buffering occurs at 1 kB.

5.3 Limit the number of concurrent threads accessing a Optane DIMM

Systems should minimize the number of concurrent threads targeting a single DIMM simultaneously. An Optane DIMM's limited store performance and limited buffering at the iMC and on the DIMMs combine to limit its ability to handle accesses from multiple threads simultaneously. We have identified two distinct mechanisms that contribute to this effect.

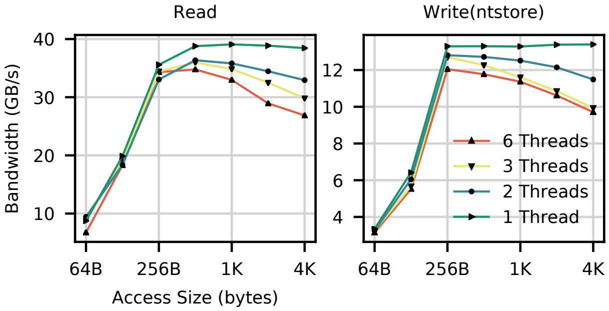


Figure 15: **Plotting iMC contention.** With a fixed number of 6 threads, as the number of DIMMs accessed by each thread grows, the bandwidth drops. For maximal bandwidth, threads should be pinned to DIMMs.

Contention in the XPBuffer Contention for space in the XPBuffer will lead to increased evictions and write backs to 3D-XPoint media, which will drive down EWR. Figure 4 (center) shows this effect in action: the performance does not scale with higher thread counts. For example, having 8 threads issuing sequential non-temporal stores achieves an EWR of 0.62 and 69% bandwidth compared to a single thread, which has an EWR of 0.98.

Contention in the iMC Figure 15 illustrates how limited queue capacity in the iMC also hurts performance when multiple cores target a single DIMM. The figure shows an experiment that uses a fixed number of threads (24 for read and 6 for ntstore) to read/write data to 6 interleaved Optane DIMMs. We let each thread access *N* DIMMs (with even distribution across threads) randomly. As *N* rises, the number of writers targeting each DIMM grows, but the per-DIMM bandwidth drops. A possible culprit is the limited capacity of the XPBuffer, but EWR remains very close to 1, so the performance problem must be in the iMC.

On our platform, the WPQ buffer queues up to 256 B data issued from a single thread. Our hypothesis is that, since Optane DIMMs are slow, they drain the WPQ slowly, which leads to head-of-line blocking effects. Increasing *N* increases contention for the DIMMs and the likelihood that any given processor will block waiting for stores ahead of it to complete.

Figure 5 (right) shows another example of this phenomenon — Optane bandwidth falls drastically when doing random 4 kB accesses across interleaved Optane DIMMs. Optane memory interleaving is similar to RAID-0 in disk arrays: The chunk size is 4 kB and the stripe size is 24 kB (Across the 6 DIMMs on the socket, each gets a 4 kB contiguous block). The workload in Figure 5 (right) spreads accesses across these interleaved DIMMs, and will lead to spikes in contention for particular DIMMs.

Thread starvation occurs more often as the access size

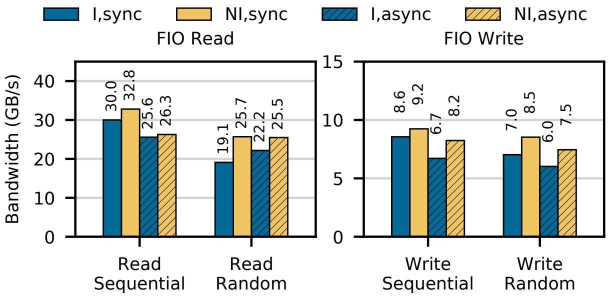


Figure 16: **Multi-DIMM NOVA** We make NOVA multi-DIMM aware by evenly loading the NVDIMMs, and get improve performance by an average of 17% on the FIO benchmark.

grows, reaching maximum degradation at the interleaving size (4 kB). For accesses larger than the interleaving size, each core starts spreading their accesses across multiple DIMMs, evening out the load. The write data also show small peaks at 24 kB and 48 kB where accesses are perfectly distributed across the six DIMMs.

This degradation effect will occur whenever 4 kB accesses are distributed non-uniformly across the DIMMs. Unfortunately, this is probably a common case in practice. For instance, a page buffer with 4 kB pages would probably perform poorly in this configuration.

5.3.1 Case Study: Multi-NVDIMM NOVA

The original NOVA design did not attempt to limit the number of writers per DIMM. In fact, it tends to allocate pages for a file from contiguous regions which, via interleaving, tends to spread those pages across the DIMMs. To fix this issue, we configured our machine to pin writer threads to non-interleaved Optane DIMMs. This configuration ensures an even matching between threads and NVDIMMs, thereby leveling the load and maximizing bandwidth at each NVDIMM.

Figure 16 shows the result. Our experiment uses the FIO benchmark [6] to test the optimization and uses 24 threads. We plot the bandwidth of each file operation with two different IO engines: `sync` and `libaio` (async). By being Multi-NVDIMM aware, our optimization improves NOVA's bandwidth by between 3% and 34%.

5.4 Avoid mixed or multi-threaded accesses to remote NUMA nodes

NUMA effects for Optane are much larger than they are for DRAM, so designers should work even harder to avoid cross-socket memory traffic. The cost is especially steep for accesses that mix load and stores and include multiple threads. Between local and remote Optane memory, the typical read latency difference is 1.79× (sequential) and 1.20× (random), respectively. For writes, remote Optane's latency is 2.53× (ntstore) and 1.68× higher compared to local. For bandwidth, remote Optane can achieve 59.2% and 61.7% of local read and write bandwidth at optimal thread count (16 for local read, 10 for remote read, and 4 for local and remote write).

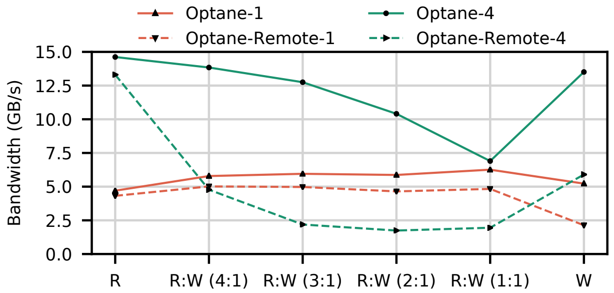


Figure 17: **Memory bandwidth on Optane and Optane-Remote** This chart shows bandwidth as we varied the mix of accesses for one and four threads. Pure reads or pure writes perform better on NUMA than mixed workloads, and increased thread count generally hurts NUMA performance.

The performance degradation ratio above is similar to remote DRAM to local DRAM. However, the bandwidth of Optane memory is drastically degraded when either the thread count increases or the workload is read/write mixed. Based on the results from our systematic sweep, the bandwidth gap between local and remote Optane memory for the same workload can be over 30×, while the gap between local and remote DRAM is, at max, only 3.3×.

In Figure 17, we show how the bandwidth changes for Optane on both local and remote CPUs by adjusting the read and write ratio. We show the performance of a single thread and four threads, as local Optane memory performance increases with thread count up to four threads for all the access patterns tested. Single-threaded bandwidth is similar for local and remote accesses. For multi-threaded accesses, remote performance drops off more quickly as store intensity rises, leading to lower performance relative to the local case.

5.4.1 Case Study: PMemKV

Intel's Persistent Memory Key-Value Store (PMemKV [15]) is an NVMM-optimized key-value data-store. It implements various index data structures and uses PMDK [14] to manage its persistent data. We used the concurrent hash map (`cmap`) in our tests as it is the only structure that supports concurrency.

To test the effect of Optane's NUMA imbalance on PMemKV, we varied the location of the server relative to the `pmem` pool; Figure 18 shows the result on an included benchmark with mixed workload (`overwrite`) that repeatedly performs read-modify-write operations. In this test, using a remote Optane DIMM drops the application's performance beyond two threads. Optane performance is far more impacted by the migration (loss of 75%) than DRAM (loss of 8%).

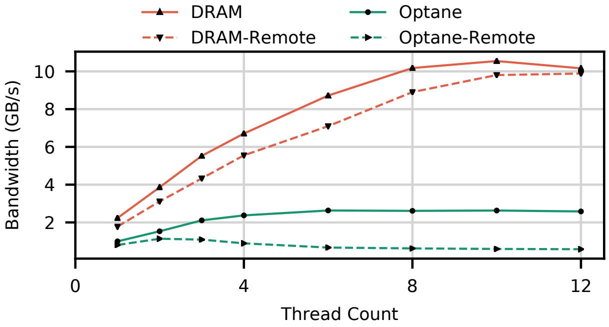


Figure 18: **NUMA degradation for PmemKV** Optane memory experiences greater NUMA-based degradation than DRAM. Migrating to a remote Optane node reduces PmemKV performance by up to 4.5× (18× vs. DRAM).

6 Discussion

The guidelines in Section 5 provide a starting point for building and tuning Optane-based systems. By necessity, they reflect the idiosyncrasies of a particular implementation of a particular persistent memory technology, and it is natural to question how applicable the guidelines will be to both other memory technologies and future versions of Intel's Optane memory. It is also important to note that we have only studied the guidelines in the context of App Direct mode, since the large DRAM cache that Memory Mode provides mitigates most or all of the effects they account for. We believe that our guidelines will remain valuable both as Optane evolves and as other persistent memories come to market.

The broadest contribution of our analysis and the resulting guidelines is that they provide a road map to potential performance problems that might arise in future persistent memories and the systems that use them. Our analysis shows how and why issues like interleaving, buffering on and off the DIMM, instruction choice, concurrency, and cross-core interference can affect performance. If future technologies are not subject the precisely the same performance pathologies as Optane, they may be subject to similar ones.

Ultimately it is unclear how scalable persistent memories will evolve. Several of our guidelines are the direct product of (micro)architectural characteristics of the current Optane incarnation. The size of the XPBuffer and iMC's WPQ might change in future implementations which would limit the importance of minimizing concurrent threads and reduce the importance of the 256 B write granularity. However, expanding these structures would increase the energy reserves required to drain the ADR during a power failure. Despite this, there are proposals to extend the ADR down to the last-level cache [37,61] which would eliminate the problem. An even more energy-intensive change would be to make the DRAM cache that Optane uses in Memory mode persistent.

Increasing or decreasing the 256 B internal write size is likely to be expensive. It is widely believed that Optane is phase-change memory and the small internal page size has long been a hallmark of the phase change memory [2] due to power limitations. Smaller internal page sizes are unlikely because the resulting memories are less dense.

A different underlying memory cell technology (e.g., spin-torque MRAM) would change things more drastically. Indeed, battery-backed DRAM is a well-known and widely deployed (although not very scalable or cost-effective) persistent memory technology. For it, most of our guidelines are unnecessary, though non-temporal stores are still more efficient for large transfers due to restrictions in the cache coherency model.

7 Related Work

With the release of the Optane DIMM in April 2019, early results on the devices have begun to be published. For instance, Van Renan et al. [44] have explored logging mechanisms for the devices. We expect additional results to be published in the near future as the devices become more widely available.

Prior art in persistent memory programming has spanned the system stack, though until very recently these results were untested on real Optane memory. A large body of work has explored transactional memory-type abstractions for enforcing a consistent persistent state [5,9,12,25,26,28,33,49]. Various authors have built intricate NVM data structures for logging, data storage, and transaction processing [3,4,10,11,17,23,24,27,30,35,38,39,41,45,46,50,57]. Custom NVM file systems have also been explored [13,21,34,48,52,54–56,62].

8 Conclusion

This paper has described the performance of Intel's new Optane DIMMs across micro- and macro-level benchmarks. In doing so, we have extracted actionable guidelines for programmers to fully utilize these devices' strengths. The devices have performance characteristics that lie in-between traditional storage and memory devices, yet they also present interesting performance pathologies. We believe that the devices will be useful in extending the quantity of memory available and in providing low-latency storage.

Acknowledgments

We thank our shepherd, Ric Wheeler and the reviewers for their insightful comments and suggestions. We are thankful to Subramanya R. Dulloor, Sanjay K. Kumar and Karthik B. Sukumar from Intel for their support and help with accessing the test platform. We would like to thank Jiawei Gao, Xiao Liu, Amirsaman Memaripour, Yun Joon Soh, Zixuan Wang, Yi Xu and Lu Zhang for suggestions on improving the the experiments and writing. This work was supported in part by CRISP, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA.

References

[1] Intel® 64 and IA-32 Architectures Optimization Reference Manual. 2019.

[2] Ameen Akel, Adrian M. Caulfield, Todor I. Mollov, Rajesh K. Gupta, and Steven Swanson. Onyx: A protoype phase change memory storage array. In *Proceedings of the 3rd USENIX Conference on Hot Topics in Storage and File Systems*, HotStorage'11, Berkeley, CA, USA, 2011.

[3] Joy Arulraj, Justin Levandoski, Umar Farooq Minhas, and Per-Ake Larson. BzTree: A high-performance latch-free range index for non-volatile memory. *Proc. VLDB Endow.*, 11(5):553–565, January 2018.

[4] Joy Arulraj, Andrew Pavlo, and Subramanya R. Dulloor. Let's talk about storage: Recovery methods for non-volatile memory database systems. In *SIGMOD*, Melbourne, Australia, 2015.

[5] Hillel Avni and Trevor Brown. Persistent hybrid transactional memory for databases. *Proc. VLDB Endow.*, 10(4):409–420, November 2016.

[6] Jens Axboe. Flexible I/O Tester, 2017. https://github.com/axboe/fio.

[7] Brian Beeler. Intel Optane DC persistent memory module (PMM). https://www.storagereview.com/intel_optane_dc_persistent_memory_module_pmm. Accessed 1/1/2020.

[8] Kumud Bhandari, Dhruva R Chakrabarti, and Hans-J Boehm. Makalu: Fast recoverable allocation of non-volatile memory. In *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pages 677–694, 2016.

[9] Dhruva R. Chakrabarti, Hans-J. Boehm, and Kumud Bhandari. Atlas: Leveraging locks for non-volatile memory consistency. In *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications*, OOPSLA '14, pages 433–452, New York, NY, USA, 2014. ACM.

[10] Andreas Chatzistergiou, Marcelo Cintra, and Stratis D. Viglas. REWIND: Recovery write-ahead system for in-memory non-volatile data-structures. *Proc. VLDB Endow.*, 8(5):497–508, January 2015.

[11] Shimin Chen and Qin Jin. Persistent B+-trees in non-volatile main memory. *Proc. VLDB Endow.*, 8(7):786–797, February 2015.

[12] Joel Coburn, Adrian M. Caulfield, Ameen Akel, Laura M. Grupp, Rajesh K. Gupta, Ranjit Jhala, and Steven Swanson. NV-Heaps: Making persistent objects fast and safe with next-generation, non-volatile memories. In *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '11, pages 105–118, New York, NY, USA, 2011. ACM.

[13] Jeremy Condit, Edmund B. Nightingale, Christopher Frost, Engin Ipek, Benjamin Lee, Doug Burger, and Derrick Coetzee. Better I/O through byte-addressable, persistent memory. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*, SOSP '09, pages 133–146, New York, NY, USA, 2009. ACM.

[14] Intel Corporation. Persistent Memory Development Kit. http://pmem.io/pmdk/.

[15] Intel Corporation. pmemkv: key/value datastore for persistent memory. https://github.com/pmem/pmemkv.

[16] Jeffrey Dean and Sanjay Ghemawat. LevelDB. https://github.com/google/leveldb.

[17] Justin DeBrabant, Joy Arulraj, Andrew Pavlo, Michael Stonebraker, Stan Zdonik, and Subramanya R. Dulloor. A prolegomenon on OLTP database systems for non-volatile memory. *Proc. VLDB Endow.*, 7(14), 2014.

[18] Mingkai Dong and Haibo Chen. Soft updates made simple and fast on non-volatile memory. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pages 719–731, Santa Clara, CA, 2017. USENIX Association.

[19] Mingkai Dong, Qianqian Yu, Xiaozhou Zhou, Yang Hong, Haibo Chen, and Binyu Zang. Rethinking benchmarking for NVM-based file systems. In *Proceedings of the 7th ACM SIGOPS Asia-Pacific Workshop on Systems*, APSys '16, pages 20:1–20:7, New York, NY, USA, 2016. ACM.

[20] Z. Duan, H. Liu, X. Liao, and H. Jin. HME: A lightweight emulator for hybrid memory. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1375–1380, March 2018.

[21] Subramanya R. Dulloor, Sanjay Kumar, Anil Keshavamurthy, Philip Lantz, Dheeraj Reddy, Rajesh Sankaran, and Jeff Jackson. System Software for Persistent Memory. In *Proceedings of the Ninth European Conference on Computer Systems*, EuroSys '14, pages 15:1–15:15, New York, NY, USA, 2014. ACM.

[22] Facebook. RocksDB, 2017. http://rocksdb.org.

[23] Ru Fang, Hui-I Hsiao, Bin He, C. Mohan, and Yun Wang. High performance database logging using storage class memory. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 1221–1231, April 2011.

[24] Michal Friedman, Maurice Herlihy, Virendra Marathe, and Erez Petrank. A persistent lock-free queue for non-volatile memory. In *Proceedings of the 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '18, pages 28–40, New York, NY, USA, 2018. ACM.

[25] Eric R. Giles, Kshitij Doshi, and Peter Varman. SoftWrAP: A lightweight framework for transactional support of storage class memory. In *2015 31st Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–14, May 2015.

[26] Terry Ching-Hsiang Hsu, Helge Bruegner, Indrajit Roy, Kimberly Keeton, and Patrick Eugster. NVthreads: Prac-

tical persistence for multi-threaded applications. In *Proceedings of the 12th ACM European Systems Conference*, EuroSys 2017, Belgrade, Republic of Serbia, 2017.

[27] Jian Huang, Karsten Schwan, and Moinuddin K. Qureshi. NVRAM-aware logging in transaction systems. In *Proceedings of the VLDB Endowment*, 2014.

[28] Joseph Izraelevitz, Terence Kelly, and Aasheesh Kolli. Failure-Atomic Persistent Memory Updates via JUSTDO Logging. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '16, pages 427–442, New York, NY, USA, 2016. ACM.

[29] Sudarsun Kannan, Ada Gavrilovska, and Karsten Schwan. pVM: Persistent virtual memory for efficient capacity scaling and object storage. In *Proceedings of the Eleventh European Conference on Computer Systems*, EuroSys '16, pages 13:1–13:16, New York, NY, USA, 2016. ACM.

[30] Hideaki Kimura. FOEDUS: OLTP engine for a thousand cores and NVRAM. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, pages 691–706, New York, NY, USA, 2015. ACM.

[31] A. Kolli, J. Rosen, S. Diestelhorst, A. Saidi, S. Pelley, S. Liu, P. M. Chen, and T. F. Wenisch. Delegated persist ordering. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–13, Oct 2016.

[32] Youngjin Kwon, Henrique Fingler, Tyler Hunt, Simon Peter, Emmett Witchel, and Thomas Anderson. Strata: A cross media file system. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, pages 460–477, New York, NY, USA, 2017. ACM.

[33] Qingrui Liu, Joseph Izraelevitz, Se Kwon Lee, Michael L. Scott, Sam H. Noh, and Changhee Jung. iDO: Compiler-directed failure atomicity for nonvolatile memory. In *51st IEEE/ACM International Symposium on Microarchitecture*, MICRO '18, October 2018.

[34] Youyou Lu, Jiwu Shu, Youmin Chen, and Tao Li. Octopus: an RDMA-enabled distributed persistent memory file system. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pages 773–785, Santa Clara, CA, 2017. USENIX Association.

[35] Virendra J. Marathe, Margo Seltzer, Steve Byan, and Tim Harris. Persistent memcached: Bringing legacy code to byte-addressable persistent memory. In *9th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 17)*, Santa Clara, CA, 2017. USENIX Association.

[36] Iulian Moraru, David G. Andersen, Michael Kaminsky, Niraj Tolia, Parthasarathy Ranganathan, and Nathan Binkert. Consistent, durable, and safe memory management for byte-addressable non volatile main memory. In *Proceedings of the First ACM SIGOPS Conference on Timely Results in Operating Systems*, TRIOS '13, pages 1:1–1:17, New York, NY, USA, 2013. ACM.

[37] Dushyanth Narayanan and Orion Hodson. Whole-system persistence with non-volatile memories. In *Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2012)*. ACM, March 2012.

[38] Faisal Nawab, Joseph Izraelevitz, Terence Kelly, Charles B. Morrey, Dhruva Chakrabarti, and Michael L. Scott. Dalí: A periodically persistent hash map. In *31st International Symposium on Distributed Computing*, DISC '17, October 2017.

[39] Ismail Oukid, Johan Lasperas, Anisoara Nica, Thomas Willhalm, and Wolfgang Lehner. FPTree: A hybrid SCM-DRAM persistent and concurrent B-tree for storage class memory.

[40] Steven Pelley, Peter M. Chen, and Thomas F. Wenisch. Memory persistency. In *Proceeding of the 41st Annual International Symposium on Computer Architecture*, ISCA '14, pages 265–276, Piscataway, NJ, USA, 2014. IEEE Press.

[41] Steven Pelley, Thomas F. Wenisch, Brian T. Gold, and Bill Bridge. Storage management in the NVRAM era. In *Proc. VLDB Endow.*, October 2014.

[42] Andy Rudoff. Deprecating the PCOMMIT instruction. https://software.intel.com/en-us/blogs/2016/09/12/deprecate-pcommit-instruction, 2016. Accessed 1/1/2020.

[43] Jihye Seo, Wook-Hee Kim, Woongki Baek, Beomseok Nam, and Sam H. Noh. Failure-atomic slotted paging for persistent memory. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '17, pages 91–104, New York, NY, USA, 2017. ACM.

[44] Alexander van Renen, Lukas Vogel, Viktor Leis, Thomas Neumann, and Alfons Kemper. Persistent memory I/O primitives. *arXiv:1904.01614*, 2019.

[45] Shivaram Venkataraman, Niraj Tolia, Parthasarathy Ranganathan, and Roy Campbell. Consistent and durable data structures for non-volatile byte-addressable memory. In *Proceedings of the 9th USENIX Conference on File and Storage Technologies*, FAST '11, San Jose, CA, USA, February 2011. USENIX Association.

[46] Stratis D Viglas. Write-limited sorts and joins for persistent memory. *Proc. VLDB Endow.*, 7(5):413–424, 2014.

[47] Haris Volos, Guilherme Magalhaes, Ludmila Cherkasova, and Jun Li. Quartz: A lightweight performance emulator for persistent memory software. In *Proceedings of the 16th Annual Middleware Conference*, Middleware '15, pages 37–49, New York, NY, USA, 2015. ACM.

[48] Haris Volos, Sanketh Nalli, Sankarlingam Panneerselvam, Venkatanathan Varadarajan, Prashant Saxena, and

Michael M. Swift. Aerie: Flexible file-system interfaces to storage-class memory. In *Proceedings of the Ninth European Conference on Computer Systems*, EuroSys '14, pages 14:1–14:14, New York, NY, USA, 2014. ACM.

[49] Haris Volos, Andres Jaan Tack, and Michael M. Swift. Mnemosyne: Lightweight persistent memory. In *ASPLOS '11: Proceeding of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems*, New York, NY, USA, 2011. ACM.

[50] Tianzheng Wang and Ryan Johnson. Scalable logging through emerging non-volatile memory. *Proc. VLDB Endow.*, 7(10):865–876, June 2014.

[51] Mingyu Wu, Ziming Zhao, Haoyu Li, Heting Li, Haibo Chen, Binyu Zang, and Haibing Guan. Espresso: Brewing java for more non-volatility with non-volatile memory. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '18, pages 70–83, New York, NY, USA, 2018. ACM.

[52] Xiaojian Wu and A. L. Narasimha Reddy. SCMFS: A file system for storage class memory. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 39:1–39:11, New York, NY, USA, 2011. ACM.

[53] Jian Xu, Juno Kim, Amirsaman Memaripour, and Steven Swanson. Finding and fixing performance pathologies in persistent memory software stacks. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, pages 427–439, New York, NY, USA, 2019. ACM.

[54] Jian Xu and Steven Swanson. NOVA: A log-structured file system for hybrid volatile/non-volatile main memories. In *14th USENIX Conference on File and Storage Technologies (FAST 16)*, pages 323–338, Santa Clara, CA, February 2016. USENIX Association.

[55] Jian Xu, Lu Zhang, Amirsaman Memaripour, Akshatha Gangadharaiah, Amit Borase, Tamires Brito Da Silva, Steven Swanson, and Andy Rudoff. NOVA-Fortis: A fault-tolerant non-volatile main memory file system. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, pages 478–496, New York, NY, USA, 2017. ACM.

[56] Jian Yang, Joseph Izraelevitz, and Steven Swanson. Orion: A distributed file system for non-volatile main memory and RDMA-capable networks. In *Proceedings of the 17th USENIX Conference on File and Storage Technologies (FAST '19)*, 2019.

[57] Jun Yang, Qingsong Wei, Cheng Chen, Chundong Wang, Khai Leong Yong, and Bingsheng He. NV-Tree: Reducing consistency cost for NVM-based single level systems. In *13th USENIX Conference on File and Storage Technologies*, FAST '15, pages 167–181, Santa Clara, CA, February 2015. USENIX Association.

[58] Lu Zhang and Steven Swanson. Pangolin: A fault-tolerant persistent memory programming library. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 897–912, Renton, WA, July 2019.

[59] Yiying Zhang and Steven Swanson. A study of application performance with non-volatile main memory. In *Proceedings of the 2015 IEEE Symposium on Mass Storage Systems and Technologies (MSST'15)*, 2015.

[60] Yiying Zhang, Jian Yang, Amirsaman Memaripour, and Steven Swanson. Mojim: A reliable and highly-available non-volatile memory system. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '15, pages 3–18, New York, NY, USA, 2015. ACM.

[61] Jishen Zhao, Sheng Li, Doe Hyun Yoon, Yuan Xie, and Norman P. Jouppi. Kiln: Closing the performance gap between systems with and without persistence support. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-46, pages 421–432, New York, NY, USA, 2013. ACM.

[62] Shengan Zheng, Morteza Hoseinzadeh, and Steven Swanson. Ziggurat: A tiered file system for non-volatile main memories and disks. In *17th USENIX Conference on File and Storage Technologies (FAST 19)*, pages 207–219, Boston, MA, 2019. USENIX Association.