Evaluation of Intel 3D-Xpoint NVDIMM Technology for Memory-Intensive Genomic Workloads

Daniel Waddington
IBM Research Almaden
San Jose, California
daniel.waddington@ibm.com

Mark Kunitomi
IBM Research Almaden
San Jose, California
mark.kunitomi@ibm.com

Clem Dickey
IBM Research Almaden
San Jose, California
dickeycl@us.ibm.com

Samyukta Rao
IBM Research Almaden
San Jose, California
rao.samyukta@ibm.com

Amir Abboud
IBM Research Almaden
San Jose, California
amir.abboud@ibm.com

Jantz Tran
Intel Corporation
Santa Clara, California
jantz.c.tran@intel.com

ABSTRACT

New 3D-XPoint^{TM} technology, developed by Intel and Micron, promises to deliver high-density, lower-cost, non-volatile storage with DRAM-like performance characteristics. This paper presents a detailed empirical evaluation of Intel's Optane DC Persistent Memory solution that provides 3D-XPoint NV-DIMMs, which are directly attached to the memory bus. We evaluate general performance through a set of micro-benchmarks and also evaluate application-specific performance through measurement of a production bioinformatics workload (genome K-mer analysis). This is a memory-intensive workload that does not scale-out well with conventional data-partitioning and therefore directly benefits from increased main memory capacity. Thus, for this workload, 3D-Xpoint is key to enabling previously unattainable results. We compare performance with existing DRAM memory, evaluate different modes of operation and examine multiple integration approaches.

CCS CONCEPTS

• **Applied computing** → **Bioinformatics**; • **Hardware** → **Biology-related information processing**; **Memory and dense storage**; • **Computer systems organization** → *Multicore architectures*;

KEYWORDS

3DXP, PCM, storage class memory, bioinformatics, genomics, k-mers

ACM Reference Format:
Daniel Waddington, Mark Kunitomi, Clem Dickey, Samyukta Rao, Amir Abboud, and Jantz Tran. 2019. Evaluation of Intel 3D-Xpoint NVDIMM Technology for Memory-Intensive Genomic Workloads. In *Proceedings of the International Symposium on Memory Systems (MEMSYS '19), September 30-October 3, 2019, Washington, DC, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3357526.3357528

MEMSYS '19, September 30-October 3, 2019, Washington, DC, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-7206-0/19/09…$15.00
https://doi.org/10.1145/3357526.3357528

1 INTRODUCTION

Over a decade ago, the term Storage Class Memory (SCM) was coined by IBM [3]. SCM is defined as memory technology that is non-volatile, uses a solid-state implementation, very low latency, low cost per bit and physical durability sufficient for practical use. Even though NAND-flash technology may be considered an early form of SCM, its durability ($\sim 10^4$ erase-write cycles) limits its practicality. 3D-XPoint is the first "real" contender for qualification as SCM. In the original definition by Freitas et al., SCM can be incorporated into a traditional storage device (block-oriented and attached to the IO bus) or alternatively as a primary memory situated below DRAM that is functioning as an L4 cache. The latter form, which we will qualify by using the term persistent memory (PM), provides a *byte-addressable* solution that is connected to the memory bus and thus responds directly to CPU load/store instructions.

The key promise of PM is its ability to provide low latency (in the order of hundreds of nanoseconds) and the ability to do reads and writes smaller than traditional block storage[1]. This fine-grained access capability means that small write updates can be achieved without incurring read-modify-write amplification penalties that arise with block-based access. Furthermore, direct accessibility by the CPU means that data can be stored, making it resilient against power failure, without the need to perform explicit transformation and serialization of data from memory space to the storage device. Nevertheless, PM requires closer programming language integration (e.g., support for persistent data types) and new approaches to data resilience and crash-consistency.

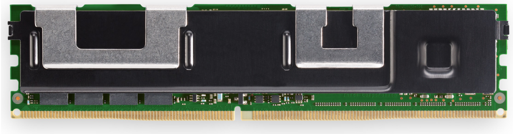


Figure 1: Optane DC Persistent Memory NV-DIMM

3D-XPoint also provides higher density than traditional DRAM (about x8). As a result, it is able to provide a increased main memory capacity and thus facilitate the processing of larger working

[1] Most devices today operate on a 4KiB storage block size, but many are increasing to larger sizes under the hood.

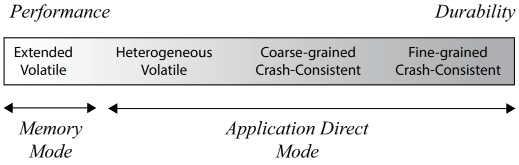


Figure 2: Persistent Memory Adoption Spectrum

sets. This enlarged memory is critical for certain classes of application that are inherently difficult to batch and/or pipeline. Examples include in-memory databases, HPC, interactive analytics and infomatics, and non-deterministic analytics workloads (e.g., large mathematical set construction).

In this paper, we experimentally evaluate Intel's Optane DC Persistent Memory (hereafter termed Optane-PMDC). Optane-PMDC provides 3D-XPoint in an NV-DIMM form factor that allows byte-addressable[2] access via the processor's load/store instructions (see Figure 1). Our experiments evaluate, with data from over 600 hours of experiments, both the fundamental performance metrics (e.g., throughput and latency) and performance for an exemplar memory-intensive genomics workload. We also examine different "modes of operation" and adoption strategies (e.g., data partitioning), and quantify the impact on performance of each. As a primary concern, we examine the use of Optane-PMDC as an form of volatile memory alternative to DRAM. This is the "low hanging fruit" and easiest path of adoption (see Figure 2). We also explore other modes of operation, including use of persistent memory as an explicit (heterogeneous) memory tier and a full, crash-consistent persistent memory paradigm. Specifically, we evaluate both custom and PMDK-based[3] crash-consistent data structures on persistent memory hardware.

2 BACKGROUND

2.1 3D-XPoint Persistent Memory

3D-XPoint is a new "Storage Class Memory" [3] technology developed by Intel and Micron, beginning around 2012. It is based on altering the resistance of chalcogenic materials by changing the material state (using heat) from amorphous to crystalline. The exact details have not been made public by Intel Corporation. In 2017, Intel launched its first product (trade marked Optane DC) around 3D-XPoint in the form of an NVMe-based SSD (P4800X). This later followed in 2018 with the announcement of Intel Optane DC Persistent Memory, an NV-DIMM product based on 3D-XPoint. This is the focus of this paper.

The principal modes of operation are: (a) as an extension to volatile DRAM through hardware-paging (known as *Memory-Mode*); (b) as a non-volatile (persistent) memory that is explicitly managed separately from DRAM (known as Application Direct or *App-Direct mode*); or (c) as a combination of the two, where resources are explicitly partitioned across the two modes (known as *Hybrid Mode*).

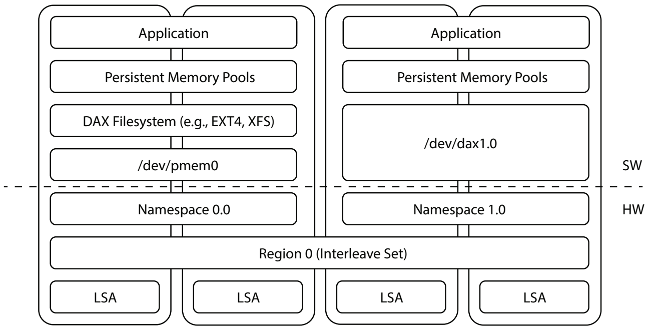


Figure 3: NV-DIMM Logical Configuration Elements

NV-DIMMs are provisioned into the appropriate mode using software configuration tools (e.g., `ipmctl`) that interact with the DIMM's firmware.

Configured in Memory-Mode, Optane-PMDC provides a transparent extension of main memory, whereby DRAM effectively becomes an L4 cache. A ratio of 8:1 (Optane-PMDC:DRAM) is recommended for Memory-Mode, and by default, data is set to interleave within a CPU socket at a granularity of 256B. For the current generation of Optane-PMDC, all DRAM is potentially pageable - there is no means to specify which regions of DRAM are paged to 3D-XPoint. The eviction policy is based on an adaptive LRU algorithm. From a software perspective, applications using Memory-Mode do not need to be modified. However, because this mode cannot be used to persist data, power-fail or reset results in all of the data being lost.

App-Direct mode supports memory heterogeneity in that both DRAM and 3D-XPoint memories are explicitly managed side-by-side. This allows the application to govern when and where to place data according to performance and space requirements. Applications must be modified to use App-Direct mode since separate memory heaps are needed to integrate Optane-PMDC.

At the socket level, Optane-PMDC in App-Direct mode can be configured and exposed to the operating system, either individually (as a region per DIMM) or using an *Interleave Set* that stripes data across multiple DIMMs into a single region, see Figure 3. By default, striping is in 4KiB pages. In Linux, regions are configured using the `ipmctl`[4] tool. Regions can be divided up further into *namespaces*, which are the fundamental unit of exposure of the hardware to the operating system. Namespaces are configured with the `ndctl` tool[5].

While Optane-PMDC can be configured as a conventional block device using the Linux Block Translation Table, in this paper we are more concerned with using memory mapping to expose access to the PM. To avoid the kernel caching persistent memory with the page cache, the DAX (Direct Access) feature is used [11]. DAX bypasses the page cache and exposes the underlying memory either as a plain device (device-DAX) or through a DAX-compatible filesystem (fs-DAX). The latter allows further partitioning of the memory resources together with support for conventional file-based access control. Figure 3 illustrates how device-DAX and fs-DAX differ.

[2] OptanePM actually reads and writes with 64-byte cache lines, unless caching is turned off for the memory region

[3] Persistent Memory Development Kit (previously known as pmem.io) [2]

[4] https://github.com/intel/ipmctl

[5] https://github.com/pmem/ndctl

To enable huge-page (2 MiB) fault handling, the filesystem should be formatted appropriately. For the experiments supporting this paper, the following XFS configuration was used:

```
mkfs.xfs -f -d su=2m,sw=1 /dev/pmem12
mount -o dax /dev/pmem12 /mnt/pmem0
xfs_io -c "extsize 2m" /mnt/pmem0
```

When applications cannot be made NUMA-aware (e.g., through data and thread partitioning), it is sometimes necessary to stripe across NUMA zones. In Linux, this can be configured with the Device Mapper (DM), which enables the creation of virtual block devices (logical volumes) from one or more underlying devices. For example, this can be used to stripe multiple NVDIMM regions (e.g., one per NUMA zone) into a single logical device. The DAX-aware filesystem is then layered on top of the logical volume. This uses software mapping and thus performance is impacted by the use of DM.

Optane-PMDC also provides on-DIMM performance counters and status registers that can be examined through software (`ipmctl`, `AEPWatch`). Performance counters include bytes read/written, reads/writes issued by host (as opposed to including those issued for maintenance), block reads/writes and read/write hit ratio for memory mode (measures prefetch efficiency). Status data includes DIMM health, media temperature, controller temperature, percentage remaining, wear level, dirty shutdowns, power-on time, up-time power cycles and error count.

All data written to Optane-PMDC DIMMs is encrypted (using XTS-AES-256) on the device. As additional processing, this adds to the latency data access compared to traditional DRAM.

2.1.1 Programming Model. As previously discussed, using Memory Direct mode is transparent to the application. For App-Direct mode, whether Optane-PMDC is being used as volatile or non-volatile memory, changes must be made to the application. For the volatile use case, the memory can be managed with the *libmemkind* [1] heterogeneous memory allocator. This allocator allows separate heaps to be established for both DRAM and 3D-Xpoint, and thus enabling the application to allocate memory from either (i.e. the application must be modified to allocate/free memory from the appropriate allocator). Libmemkind is not crash-consistent and thus cannot be used for a non-volatile mode of operation.

For the non-volatile use case, memory should be managed with a crash-consistent allocator such as provided by PMDK. However, in addition to allocator requirements, the application must use data structures that are also recoverable, relocatable and crash-consistent. These attributes can be achieved through software transactions and explicit cache flushing operations; such capabilities are also provided by PMDK, but their use generally comes at a cost to software performance.

2.2 Genomic K-mer Analysis Workload

For this paper, we are using a production genomic workload to evaluate the performance of Optane-PMDC. This specific workload is pertinent to large persistent memory because it does not easily parallelize (e.g., through data partitioning) and directly benefits from an increased working set size (improving both performance and capability).

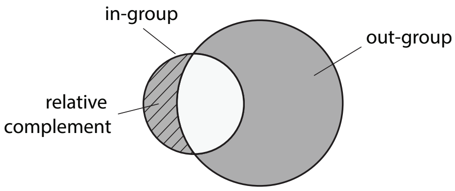


Figure 4: K-mer Set Analysis

The workload examined is used to facilitate the development of PCR-based (Polymerase Chain Reaction [7]) bacterial detection solutions by identifying a small *primer* substring (in this case a K-mer where K=100) of nucleotides that can be readily amplified by the PCR process and thus used as the basis for identification.

The aim of the bacterial identification is to determine whether a sample of bacteria belongs to a given “in-group” and does not belong to others in the broader “out-group”. For example, the in-group might be the serovar *Salmonella Virchow*, while the out-group might be the broader species *Salmonella*. Thus, the objective of the detection kit is to determine the presence of Salmonella Virchow and not be triggered, as a false-positive, by some other member of the broader Salmonella family.

From a computational perspective, the analysis involves creating a K-mer set of the *intersection* of all K-mers in all genomes of the in-group and a *union* K-mer set of all K-mers in the out-group. Both are sets of unique K-mers. From these two sets, the relative complement is performed in order to identify K-mers in the in-group that are not in the out-group (see hatched area in Figure 4). These K-mers are the *primer candidates* and are taken further into the primer design process.

Aside from PCR-primer identification, this type of K-mer based analysis is also increasingly attractive for comparison of genomic sequences that are subject to highly-changing mutations such as found in bacteria.

3 EXPERIMENTAL SYSTEM

The data collected in this paper is based on experiments run on Optane DC Persistent Memory (ES2 - Engineering Sample 2). Performance of ES2 is expected to be within 10% of the final qualified product.

Tables 1 and 2 provide details of the system configuration. CPU frequency is locked at 2.1GHz and the test system is unloaded.

Processor	Intel Xeon Scalable Platform x2 CPU (Cascade-Lake) packages 8 cores (16HT) each clock @ 2.1 GHz (max 4.0 GHz) - 4205 bogomips
Cache	L1 (32KiB), L2 (1MiB), L3 (39MiB)
Mainboard	S2600WFD (Lewisburg PCH)
DRAM	PC2400 DDR4 64GiB DIMMs (1.5TiB fully configured)
NVDIMM	Optane-PMDC 512GB 12x DIMMs (6TB)

Table 1: Test System HW Specification

Operating System	Linux Fedora 27
Kernel	4.15-6-300.fc27.x86_64
Compiler	GCC 7.3.1
Library Versions	memkind(v1.7.0), PMDK(stable-1.4)

Table 2: Test System SW Specification

Config.	DRAM	Optane-PMDC	Ratio
DRAM.768 + PM.0	64 GiB x12 (768 GiB)	None	n/a
DRAM.384 + PM.491	64 GiB x6 (384 GiB)	491.7 GiB x1	1:1.28
DRAM.384 + PM.29506	64 GiB x6 (384 GiB)	491.7 GiB x6 (2.88 TiB)	1:8
DRAM.96 + PM.491	16 GiB x6 (96 GiB)	491.7 GiB x1 (491.7 GiB)	1:5
DRAM.96 + PM.29506	16 GiB x6 (96 GiB)	491.7 GiB x6 (2.88 TiB)	1:30

Table 3: Per-Socket Memory Configurations

4 MICRO-BENCHMARK EVALUATION

To give the reader a sense of the performance envelopes of the Optane-PMDC technology, this section presents basic throughput and latency measurements. A cross-comparison with DRAM is included where possible. Here, we first explore using Optane-PMDC purely as volatile memory - data structures are not-crash consistent and cache flushing is not performed.

4.1 Memory-Mode

This section examines the performance of Optane-PMDC's Memory-Mode. To provide a basic measurement of memory latency and bandwidth we use the Intel®Memory Latency Checker (MLC) tool[6] version 3.4 with AVX512 build.

Basic latency measurement is for *idle memory latency* (with `--latency_matrix` command line option). Only a single measurement thread is active. For throughput measurement we run the *loaded latency* test (with `--loaded_latency`). This measures latency on a pinned measurement thread while other cores are running load-generation threads. The measurement thread traverses an array of pointers where each pointer is pointing to the next one, thereby creating a dependency in reads. Random fills of 4KiB blocks in 64B increments are used.

Mode	Configuration	Local NUMA Lat. (ns)	Remote NUMA Lat. (ns)	Loaded Max. Bandwidth (GB/s)
n/a	DRAM.768 + PM.0	121.5	196.2	127.67
MM	DRAM.96 + PM.491	514.8	587.4	30.62
MM	DRAM.96 + PM.29506	529.9	629.9	48.63

Table 4: MLC Test Results for Memory-Mode

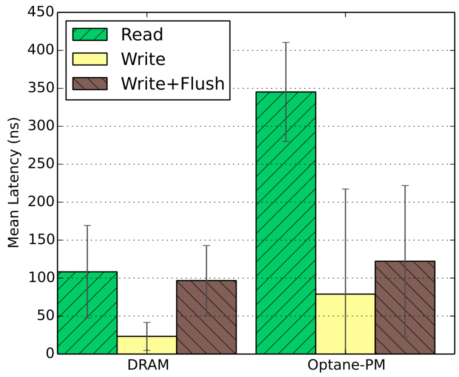


Figure 5: App-Direct Random 8-byte Latencies

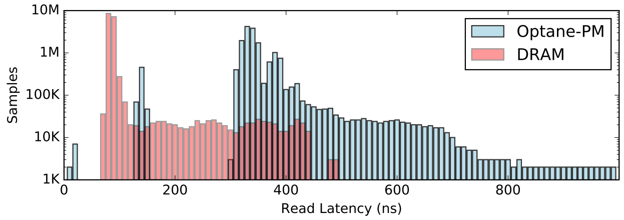


Figure 6: App-Direct Random 8-byte Read Latency Distribution

Data in Table 4 shows that Memory-Mode paging incurs a throughput slow-down factor of 2.63 and additional latency of 350-400ns, as compared to a DRAM only configuration.

Key Observation (1). *Memory-mode raw throughput performance is around half of that of DRAM. This requires striping across six Optane-PMDC DIMMs in the socket. Latency is approximately three times that of DRAM.*

4.2 App-Direct Mode

This section presents basic performance characteristics for Optane-PMDC operating in App-Direct mode.

4.2.1 Access Latency. To measure latency, a 1GiB region of memory is allocated in the local NUMA node and a random access pattern pre-calculated to ensure perfect (at most once) access to each "IO block" operation. Memory is huge page (2MiB) aligned and mapped. Optane-PMDC configuration is App-Direct with interleave set across six 512GiB DIMMs (configuration DRAM.96+PM.29506) with fs-DAX (XFS, 2MiB alignment). Workload is single-threaded and thus most optimistic.

Figure 5 shows fine-grained (8-byte) access latencies for DRAM and Optane-PMDC. Write latency of DRAM, without explicit flushing, is about one third that of OptanePM. However, with flushing the write latencies of the two are comparable. Read latency of Optane-PMDC is approximately three times that of DRAM; this is higher than many prior predictions.

[6] MLC is available from http://software.intel.com

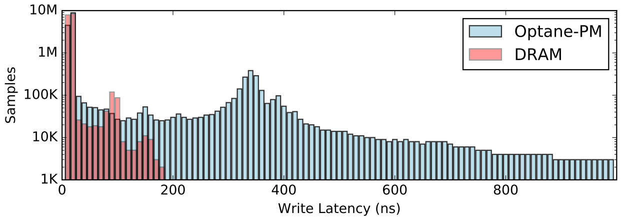


Figure 7: App-Direct Random 8-byte Write Latency Distribution

The latency distributions for 8B random read and write are given in Figures 6 and 7. Note that in both figures a log scale is used on the y-access.

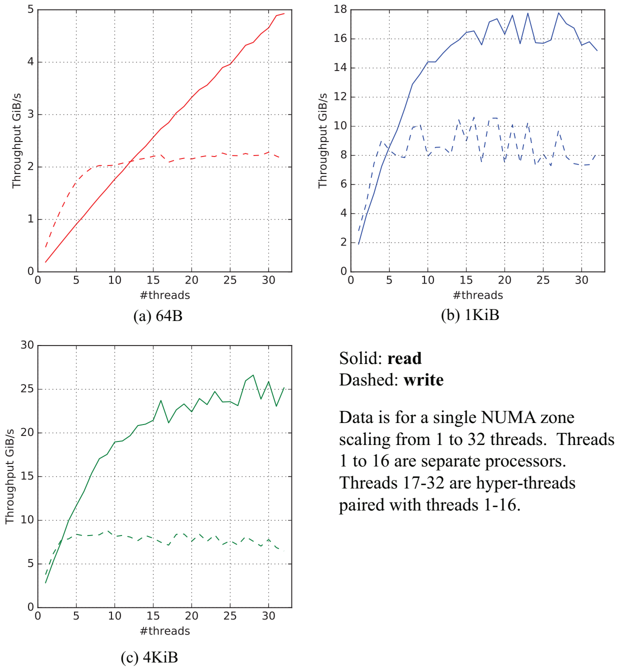
Table 5 presents data for different block sizes in both random and sequential access patterns. This data is for a DRAM.96+PM.29506 configuration (see Table3).

Operation	IO Size	DRAM	Optane-PMDC	Slow-down
	(bytes)	mean (ns)	mean (ns)	factor
rand read	64	103.57	343.74	3.3
	1024	223.37	535.87	2.4
	4096	593.82	1476.73	2.48
rand write	64	27.04	82.94	3.07
	1024	223.09	231.52	1.04
	4096	746.98	941.07	1.26
rand write+flush	64	98.81	131.48	1.33
	1024	333.96	360.07	1.07
	4096	985.83	1058.65	1.07
seq read	64	12.42	26.63	2.14
	1024	163.37	396.52	2.42
	4096	566.78	1448.40	2.55
seq write	64	10.24	31.84	3.11
	1024	173.44	230.20	1.33
	4096	757.23	919.33	1.21
seq write+flush	64	96.26	108.41	1.12
	1024	333.63	359.13	1.07
	4096	985.42	1052.64	1.06

Table 5: Single-thread Access Latency Summary

As a comparison point, latency for a random 4KiB read/write (using queue depth one) on an Intel Optane P4800X NVMe SSD, is between 7 and 10 microseconds depending on the device driver architecture (i.e. kernel vs. user-level).

4.2.2 Throughput. Throughput is measured for an increasing number of threads. Each thread allocates its own region of memory (1GiB) and performs random accesses to it. Threads are pinned to the core corresponding to their thread identifier shown on the X-axis.



(a) 64B

(b) 1KiB

(c) 4KiB

Solid: **read**
Dashed: **write**

Data is for a single NUMA zone scaling from 1 to 32 threads. Threads 1 to 16 are separate processors. Threads 17-32 are hyper-threads paired with threads 1-16.

Figure 8: Throughput Scaling of Optane-PMDC

Results show, that for a single socket (NUMA zone), Optane-PMDC is able to sustain nearly 25GiB/s 4K-random read, and approximately 7GiB/s 4K-random write [7]. Thus, read-write performance is not symmetrical. However, for small 64B operations, write performance exceeds that of read performance up to 10 threads. Write performance plateaus at around 2GiB/s.

Key Observation (2). *Raw write performance does not scale with increasing number of threads; scaling stops at around 4 threads. With a light-load (< 4 threads) write latency is less than that of read. This is likely due to internal buffering in Optane-PMDC. In App-Direct mode, single socket performance for large transfers can reach 25 GiB/s and 10 GiB/s for read and write respectively.*

5 GENOMICS WORKLOAD EVALUATION (AS VOLATILE MEMORY)

In this section, we present an evaluation of Optane-PMDC for our memory-intensive genomics workload (refer to Section 2.2). This workload is inherently random and cannot be easily predicted and thus renders pre-fetching ineffective.

Figure 9 illustrates the data path. First, data is pre-processed from standard gzipped FASTA files and saved into a key-value store in a 3-bit per nucleotide form (the additional bit is for the 'N' designator, which indicates any of the A, C, T and G bases). Additional benefits of using three-bits-per base is the ability to rapidly derive the complement of the base (performing XOR 0x7) and also to enable end and error marker codes.

[7] Note, that for 1K-random writes, a higher throughput of around 9.58GB/s is possible.

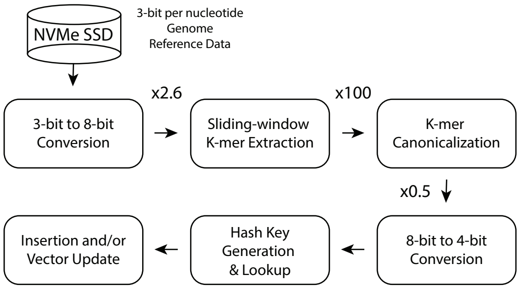


Figure 9: Genomic Workload Data Flow

On loading the reference data from the key-value store, each contiguous region (contig) is expanded into an 8-bit per base form to simplify the subsequent sliding window and canonicalization operations. The sliding window operation derives K-mers, while the canonicalization operation consistently selects one of the two nucleotide sequence forms (each the reverse-complement of the other). The canonical choice is made by a simple string comparison.

Each K-mer is constructed by converting the 8-bit per nucleotide form into a 4-bit per nucleotide form. K-mers are then inserted into a hash-table that maintains a frequency counter that defines the total number of genomes the K-mer is present in. To derive the union, all entries in the final table are included. For the intersection, K-mers that do not have a frequency counter equal to the total number of genomes are removed from the hash table. Finally, the relative complement is calculated by removing from the in-group set all K-mers that exist in the out-group set.

The workload is memory-intensive because data is "inflated" by a factor of 260 once it has been read in from the storage device. Thus, a typical NVMe SSD with a throughput of 2GiB/s can provide around 520GiB/s of processable data.

5.1 Datasets

For our experimentation, we are using data from the public NCBI GenBank (https://www.ncbi.nlm.nih.gov/genbank/) repository and also IBM's own bacterial reference data. Five reference data sets are selected, each with varying input and resulting K-mer set sizes (see Table 6). These data sets were selected because they are both relevant (i.e. represent useful analyses) and cover a spectrum of memory footprints/execution times.

Memory footprint for Intel TBB concurrent hash map and minimum time to execute the workload (i.e. best case) is provided in Table 7. All execution times are for DRAM except for the largest data set (Dataset #6), which uses Optane-PMDC Memory-Mode because of insufficient DRAM.

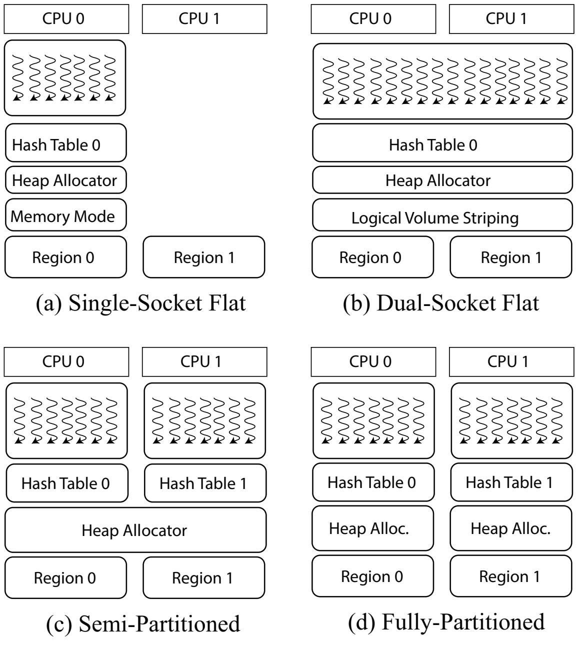
A thread-safe hash table implementation, based on Intel's Thread Building Blocks (`concurrent_hash_map`[8]) is used to ensure multiprocessor scaling. Furthermore, execution is carefully arranged to ensure that the counter is incremented only once per genome (genomes are partitioned across threads and each genome is processed by only a single thread).

Dataset	Genome Count	Base-Pairs	Unique K-mers
#1 - Salmonella (590)	48200	232B	605.9M
#2 - Salmonella Virchow (48409)	336	1.58B	25.7M
#3 - Salmonella Enterica (28901)	13014	60.6B	923.3M
#4 - S.Enterica + E.Coli (28901_562)	47635	237.4B	2.59B
#5 - S.Enterica + E.Coli + P.Aeruginosa (28901_562_287)	52538	269.5B	2.88B
#6 - GenBank (RefSeq Complete)	7641	32.0B	14.5B

Table 6: Experimental Genomic Datasets

Dataset	Data Size (GiB)	Best-case Execution Time (seconds)
#1 - Salmonella	68.8	2838
#2 - Salmonella Virchow	2.9	18
#3 - Salmonella Enterica	104	735
#4 - S.Enterica + E.Coli	294	3830
#5 - S.Enterica + E.Coli + P.Aerug.	327	4650
#6 - GenBank (RefSeq Complete)	939	2838

Table 7: Raw Memory Footprints (without hashtable overhead) and Execution Times



(a) Single-Socket Flat (b) Dual-Socket Flat

(c) Semi-Partitioned (d) Fully-Partitioned

Figure 10: Experimental Software Configurations

[8] https://www.threadingbuildingblocks.org/

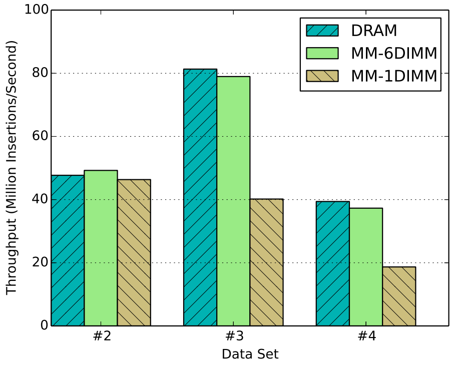


Figure 11: Single-Socket Performance of DRAM vs. Memory-Mode PM

5.2 Comparison of DRAM vs. Optane-PMDC Memory-Mode

Our initial set of experiments make a comparison of DRAM and Optane-PMDC configured in Memory-Mode. Specifically, we examine union set construction for data sets #2, #3 and #4 (see Table 6).

5.2.1 Single-Socket Performance. To eliminate any complications created by NUMA characteristics, we start by examining single-socket performance (Figure 10a). Experiments are run across three configurations; DRAM.768 + PM.0 (all DRAM), DRAM.384 + PM.491 (DRAM with single Optane-PMDC DIMM) and DRAM.384 + PM.29506 (DRAM with 6-way interleaved Optane-PMDC DIMM). Refer to Table 3 for configuration specifics. Mean throughput is given for ten iterations of the experiment.

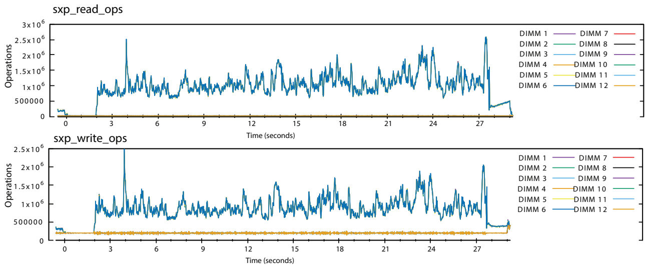


Figure 12: Single-Socket AEPWatch Data for Dataset #2 (Memory-Mode)

Figure 11 shows that using Optane-PMDC Memory-Mode with a full interleave set (x6 DIMMs) is able to achieve a throughput within 4% of a DRAM-only configuration. Performance degrades by around a factor of two when a single Optane-PMDC DIMM is used. For the small data set (#2), the fully interleaved Optane-PMDC configuration performs better than DRAM. This is likely because the DRAM DIMMs in the latter are smaller (16GiB as opposed to 64GiB) and thus throughput-per memory area is improved. That is, the work load for set #2 is targeting a single DIMM (since it fits in a single DIMM) in the DRAM case, whereas the MM-6DIMM stripes across 6 DIMMs.

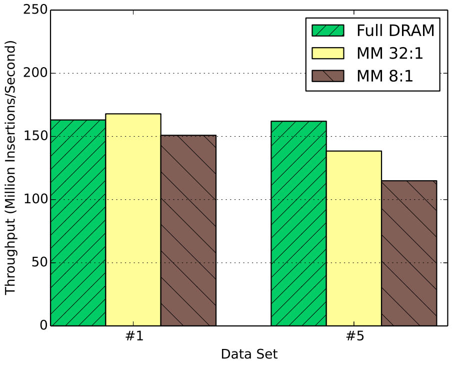


Figure 13: Dual-Socket Performance of DRAM vs. Memory-Mode Optane-PMDC

Key Observation (3). *In the context of our test workload that requires computation (i.e., it performs more than IO), Memory-Mode with interleaving is able to achieve nominally the same performance as would be expected from a pure DRAM solution.*

Figure 12 shows the total number of operations performed by the controller (sxp_read_ops and sxp_write_ops). This data is for the third configuration (DRAM.384 + PM.29506 6-way interleaved). Note that, due to interleaving, there are similar lines for each of the DIMMs; they are obscured by the uppermost line. Thus, the total number of operations is given by about 6 times that shown by the upper graph - between 6M and 9M operations per second. Each operation is 256B (x4 cache lines). Thus, using 10^6 as a nominal throughput sample, the system is moving data at around $256 * 10^6 * 6 bytes/sec \approx 1.5GB/s$ in both read and write directions.

5.2.2 Dual-Socket Performance. For dual-socket performance we analyze data sets #1 (Salmonella) and #5 (S.Enterica + E.Col + P.Aerug). These are relatively large data sets each with over 40K genomes but with varying K-mer commonality (and hence set size). The experiments are run with a fully-partitioned arrangement (refer to Figure 10d). This means that input reference data is distributed across the two sockets and there is no cross-socket memory access.

Three hardware configurations are used. 1) *Full DRAM* - DRAM-only configuration with x12 64GiB DIMMs per socket, totaling 1.5TiB of system memory; 2.) *MM 8:1* - DRAM and Optane-PMDC memory mode at a ratio of 8:1 (x6 64GiB DIMMs and x6 492GiB DIMMs per socket); 3.) *MM 32:1* - DRAM and Optane-PMDC memory mode at a ratio of 32:1 (x6 16GiB DIMMs and x6 492GiB DIMMs per socket).

Figure 13 shows that MM 32:1 out-performs Full DRAM, by a small margin, for data set #1. As noted previously, this is likely due to improved load-spreading across the DRAM DIMMs. Data set #1 has a smaller K-mer set size and therefore the working set footprint is smaller than that of data set #5. For data set #5 MM 32:1 is able to attain within 20% of the Full DRAM performance, while MM 8:1 incurs a 40% slow-down with respect to Full DRAM.

Key Observation (4). *Decreasing the ratio of DRAM to Optane-PMDC generally improves performance. As expected of any caching,*

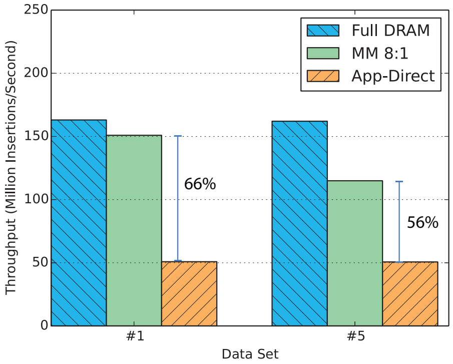


Figure 14: Comparison of DRAM, MM and App-Direct

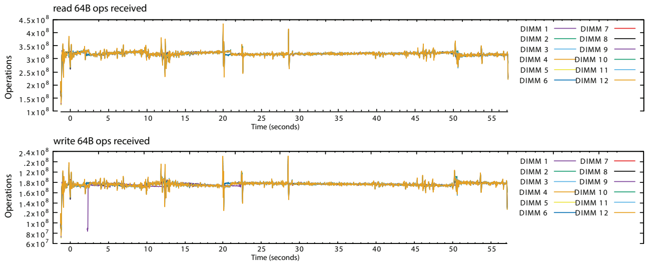


Figure 15: Throughput for App-Direct Data Set #5

the larger the data spread the less effective the caching and thus lower observed performance. In our experiments, an memory footprint inflation factor of 4.75 (set #1 to set #5) led to an additional 20% slowdown.

5.3 Comparison of DRAM vs. Optane-PMDC App-Direct

We now evaluate workload performance using Optane-PMDC configured in App-Direct mode, i.e. without transparent hardware paging. We expect this mode to perform less than Memory-Mode, since interaction with App-Direct is purely synchronous (although there does exists some hardened buffering) whereas Memory-Mode implicitly provides asynchrony. For these experiments, the data is fully partitioned with NUMA-aware memory allocation (see Figure 10d).

The data shows that using App-Direct mode results in a 66% and 56% reduction in performance, with respect to Memory-Mode, for data sets #1 (230GiB footprint) and #5 (1.29TiB footprint). Note that the footprints are larger than those defined in Table 7 because the hashtable is partitioned across NUMA zones and thus k-mer replicas can occur.

Figure 15 shows AEPWatch data for the experiment. The upper plot (read) shows a nominal throughput of around $3.3x10^8$ read 64B operations/second which is ~ 9.8GiB/s per socket. The lower plot shows an aggregate (two socket) throughput of around $1.8x10^8$ 64B write operations/seconds, which is ~5.3GiB/s per socket. Note that the request sizes are greater than 64B and possibly straddling cache lines, and hence this value is greater than that indicated in Figure 8a. According to Figure 8, which provides single-socket throughput data, in this experiment the Optane-PMDC bandwidth is not saturated.

5.4 Effect of NUMA-aware Memory Allocation and Partitioning

In this section, we evaluate the impact of using NUMA-aware memory allocation and table partitioning. NUMA-aware allocation means that a separate heap is established for each socket/NUMA-zone in the system and that all worker threads allocate from their local NUMA zone only. We use the *libmemkind* [1] heterogeneous memory allocator to create individual heaps for each socket. The "local" allocator for each thread is designated at thread creation (and saved in TLS).

Table partitioning means that genomes (input data streams) are partitioned across two individual tables. At the end of the processing the tables are merged (although this is not included in the measurement for these experiments). Using partitioned tables does help with memory localization, however, for this particular workload the processing footprint is inflated as a result. This is because there exists replication across the two tables (i.e. the same K-mer can exist in both).

MEMORY-MODE

Mode	Data #2	Data #5	Data #6
	Million IPS	Million IPS	Million IPS
None-Partitioned	156.24	159.29	41.96
Partitioned +NUMA	150.91	114.92	42.37
(Speedup)	(-3%)	(-27%)	(+1%)

APP-DIRECT

Mode	Data #2	Data #5	Data #6
	Million IPS	Million IPS	Million IPS
None-Partitioned	29.46	30.76	18.82
Partitioned +NUMA	50.85	50.66	35.47
(Speedup)	(72%)	(64%)	(88%)

Table 8: Impact of Partitioning on Performance

The "None-Partitioned" experiments for App-Direct mode use a striped logical volume (s/w RAID) to create a single memory region from two physical interleave sets (one-per socket), e.g., `/dev/pmem12` and `/dev/pmem13` (refer to (b) in Figure 10). Operating system software (device mapper) performs striping for the logical volume.

With respect to Memory-Mode, the use of partitioning does not demonstrate any consistent change in performance. The results, given in Table 8, show varying performance impact across the three datasets. However, the use of partitioning and NUMA-aware allocators brings clear speedup for App-Direct mode. The data shows between 64% and 88% increase in performance. Nevertheless, bear in mind, the additional cost of memory footprint incurred by the partitioning method (approximately twice[9]).

[9] There is no way to associate like genes a priori and thus the K-mer set sizes are in most cases approximately equal.

***Key Observation (5)**. Accessing memory across NUMA zones has little impact on Memory Mode and in fact, where data cannot be cleanly partitioned, reduces performance by limiting the effectiveness of the cache. However, for App-Direct, where no level 4 caching occurs, NUMA accesses can halve overall performance.*

6 GENOMICS WORKLOAD EVALUATION (AS PERSISTENT MEMORY)

In this section, we explore the cost of using Optane-PMDC as a true non-volatile (or persistent) memory. This approach is attractive because it allows data to be retained in memory and avoids the need to rebuild data in the event of machine power-fail or reset. As persistent memory, the data must be:

(1) *Recoverable* - data can be reloaded, using a global (object) instance identifier, after the program has restarted.
(2) *Atomic and Crash-consistent* - on recovery, the data can be reverted to a consistent state (e.g., rolled back via undo log).

Achieving these two attributes efficiently and correctly is an active research topic [4–6, 8, 10]. Nevertheless, we shall briefly explore the performance of two *persistent hash tables* with Optane-PMDC App-Direct mode.

The first variant, which we shall refer to as PMDK-HT, is based on PMDK and derived from the framework's example hash table implementation - specifically, we use the `hash map_tx` version. This table is fully generalized in that it supports dynamic expansion of the hash table and dynamic allocation of variable-sized keys and data values (using *libpmemobj*). It is based on closed-addressing, using linked-list chaining to resolve collisions. Expansion causes doubling of the number of buckets and is triggered when the entry count reaches twice the number of buckets. PMDK-HT uses fs-DAX with XFS configured with 2MiB paging. Internally, the PMDK allocator and transactional APIs incorporate an undo log that can be used to recover state in the event of power failure.

The second variant, which we shall refer to as Static-HT, leverages optimizations that are specific to the K-mer set construction use case (see Figure 16). A statically allocated slab is provisioned as device-DAX (configuration DRAM.96 + PM.29506) using 1GiB aligned (and thus 1GiB paging on x86_64). On top of this, the hash table uses open-addressing and quadratic probing (limited to a hop distance of 8M then resetting to one). Each "slot" in the hash table is 64 bytes in length and also 64-byte aligned. Fields in the slot art 64-bit aligned, which corresponds to the atomicity of Optane-PMDC writes and consequently there is no need to create an undo or redo log providing that the write ordering is enforced via cache flushing and memory fencing. A slot is assumed to be empty if the entry `count` field is zero. This does mean that initialization of the table must explicitly zero the memory (not included in the measurements). After a slot is written an explicit flush and fence is issued (`pmem_flush`). The Static-HT variant does not support dynamic expansion and therefore does not incorporate expansion locks.

Both hash-table variants use CityHash hash functions [9] to derive the bucket number based on hashing of the key. The key is the K-mer 50-byte value data (one nibble per nucleotide).

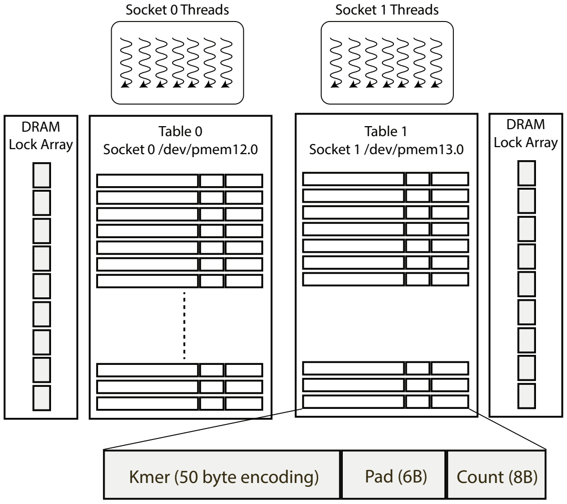


Figure 16: Static Persistent Hash-Table Design

6.1 Comparison of Volatile and Persistent Data Structures

Now, we examine the "cost" of using persistent data structures by comparing DRAM and Memory-Mode performance (using the Intel TBB hash-table), with the two different persistent memory hash-table implementations Static-HT and PMDK-HT. To eliminate any NUMA-awareness concerns, we limit the experiment to a single socket.

Table 9 provides a summary and breakdown of hardware configuration, mode and data structures. Figure 17 provides a visual comparison of relative performance.

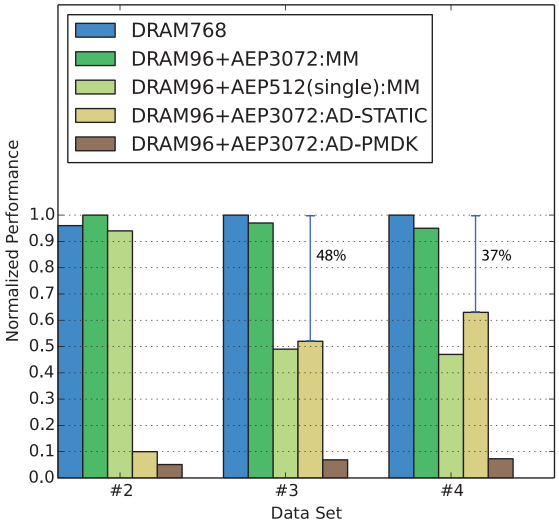


Figure 17: Normalized Single Socket Comparison

HW Config.	Mode	Data Structure	Data#2 (MIPS)	Data#3 (MIPS)	Data#4 (MIPS)
DRAM768	-	TBB	47.5	81.33	39.42
DRAM96+PM3072	MM	TBB	49.24	78.97	37.29
DRAM96+PM512	MM	TBB	46.35	40.19	18.68
DRAM96+PM3072	AD	Static-HT	4.91	42.73	24.58
DRAM96+PM3072	AD	PMDK-HT	2.54	5.66	2.89
DRAM96+PM-EMU	AD	Static-HT	8.9	90.09	-
DRAM96+PM-EMU	AD	PMDK-HT	5.34	7.85	-

Table 9: Single Socket Comparison for All Strategies

The data shows that for the small footprint workload (Data Set #2) there is a 10x decrease in performance even for the Static-HT design. In this case, the performance difference is exacerbated due to the cache-locality improvements exploited by the DRAM-based implementations. As a note, the memory footprint for Data Set #2 is only 2.9 GiB - the L3 cache in the system is 22MiB.

Data Sets #3 and #4 incur memory footprints of 104GiB and 294GiB respectively. For these larger data sets, Figure 17 shows the Static-HT design, on Optane-PMDC, is able to perform within 48% and 37% of its DRAM-based counterparts. This is congruent with expected performance based on the micro-benchmarking results given in Section 4. However, the PMDK-HT implementation on Optane-PMDC is only able to reach approximately 7% of DRAM performance. This is congruent with findings from prior work by Wang et al. [12].

Key Observation (6). *Realizing crash-consistency through PMDK (as a general solution) can reduce performance by an order of magnitude compared to a DRAM-based volatile equivalent. With lightweight data structure designs (e.g., Static-HT) performance is reduced to around 40-50% of DRAM performance.*

6.2 Isolation of Persistent Data Structure and Optane-PMDC Overheads

Given the previous results, the question arises, how much of the degradation in performance is caused by the use of persistent data structures versus the use of the Optane-PMDC media? To provide some supporting data to help answer this question, we directly compare the persistent hash-table implementations in DRAM and Optane-PMDC. The DRAM "emulation" uses the `memmap` kernel parameter[10] to isolate a region of physical memory to be exported by the kernel as a persistent memory device (i.e. `/dev/pmem0`). As with Optane-PMDC, the emulated persistent memory is used with *fs-DAX* and the XFS file system. When running the test program, the environment parameter `PMEM_IS_PMEM_FORCE` is set to one. This eliminates `msync` calls that would otherwise arise.

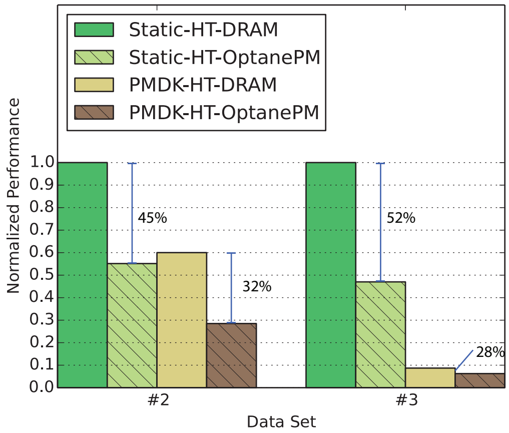


Figure 18: Comparison of OptanePM vs. Emulated PM for Persistent Data Structures

Figure 18 shows that from a best-case baseline (Static-HT on DRAM-based emulated persistent memory), the impact of using Optane-PMDC is 45% and 52% for Data Set #2 and #3 respectively. This is consistent with expectations given the micro-benchmark results. For the PMDK-HT implementation degradation resulting for Optane-PMDC is only 32% and 28%. Performance of the DRAM-emulated PMDK-HT implementation is only 5% more than the Optane-PMDC Static-HT implementation. The Static-HT implementation performs 1.93x and 7.55x better than the PMDK-HT implementation.

Key Observation (7). *For this application, removing the "cost" of crash-consistency software overhead, Optane-PMDC in AppDirect Mode is around half that of DRAM.*

7 CONCLUSION

Intel's Optane DC Persistent Memory is the first of a new breed of persistent memory that is directly connected to the memory-bus. Not only does this new memory differ from existing DRAM in its ability to provide *persistence*, it also provides *higher density* and higher capacity. However, these two attributes are independent. Optane-PMDC can be deployed, with only the latter in mind, as an extended volatile memory where crash-consistency and durability is not directly addressed. Memory-Mode is aimed at enabling this consumption model by transparently paging memory from faster DRAM (positioned as L4 cache) to Optane-PMDC.

[10] https://www.kernel.org/doc/html/v4.14/admin-guide/kernel-parameters.html

For the reviewed memory-intensive, random workload, Memory-Mode offers performance within a few percent of a DRAM-only solution, while offering up to 6TiB of memory in twelve 512GB DIMMs. For our genomics workload, this increased main memory capacity allows large data sets, that were previously infeasible to handle, to be successfully analyzed. Without Memory-Mode, we have shown that Optane-PMDC can be consumed as a heterogeneous memory with approximately half the throughput and double the latency of DRAM.

Our investigation also shows that effectively realizing *persistence* comes at a performance cost. We show that generalized software-transactional solutions (such as that offered by PMDK) can degrade performance by up to 10x of their volatile counterparts. Nevertheless, by careful design of the persistent data structures, better performance can be achieved. In our use-case, when comparing the trade-off between performance and fast-recoverability, we are inclined to choose the former since data is ultimately saved to larger capacity storage.

While Optane-PMDC technology brings new and exciting possibilities to the table, naïve consumption can result in losing the benefits of direct-to-memory attachment. Optane-PMDC's key strengths are enabling lower read and write amplification by eliminating traditional block IO boundaries, providing higher bandwidth to the CPU, and providing consistent latency by removing any complex storage controller with black-box firmware.

In our view, new research is needed to develop effective software strategies for the consumption of this new memory technology. This includes further examination of persistent memory data structures, operating system and compiler support for persistent memory, and programming language abstractions for multi-variable memory transactions. We also suggest that additional support from the hardware is necessary to reduce the cost of crash-consistency in general (e.g., through the introduction of non-volatile caches).

ACKNOWLEDGEMENT

We would like to thank the MEMSYS reviewers for their constructive feedback in reviewing this paper.

REFERENCES

[1] Christopher Cantalupo, Vishwanath Venkatesan, Jeff Hammond, Krzysztof Czurlyo, and Simon David Hammond. 2015. memkind: An Extensible Heap Memory Manager for Heterogeneous Memory Platforms and Mixed Memory Policies. (3 2015).

[2] Intel Corporation. 2015-2018. pmem.io: Persistent Memory Programming. https://pmem.io/pmdk/.

[3] R. F. Freitas and W. W. Wilcke. 2008. Storage-class Memory: The Next Storage System Technology. *IBM J. Res. Dev.* 52, 4 (July 2008), 439–447. https://doi.org/10.1147/rd.524.0439

[4] Aasheesh Kolli. 2017. Architecting Persistent Memory Systems. *Ph.D. Thesis* (2017).

[5] Mengxing Liu, Mingxing Zhang, Kang Chen, Xuehai Qian, Yongwei Wu, Weimin Zheng, and Jinglei Ren. 2017. DudeTM: Building Durable Transactions with Decoupling for Persistent Memory. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '17)*. ACM, New York, NY, USA, 329–343. https://doi.org/10.1145/3037697.3037714

[6] Youyou Lu, Jiwu Shu, and Long Sun. 2016. Blurred Persistence: Efficient Transactions in Persistent Memory. *Trans. Storage* 12, 1, Article 3 (Jan. 2016), 29 pages. https://doi.org/10.1145/2851504

[7] K Mullis, F Faloona, S Scharf, R Saiki, G Horn, and H Erlich. 1986. Specific Enzymatic Amplification of DNA In Vitro: The Polymerase Chain Reaction. *Cold Spring Harbor Symposia on Quantitative Biology* 51, 0 (Jan. 1986), 263–273.

[8] Matheus Ogleari, Ethan L. Miller, and Jishen Zhao. 2018. Steal but No Force: Efficient Hardware Undo+Redo Logging for Persistent Memory Systems. *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)* (2018), 336–349.

[9] Geoff Pike. 2018. CityHash: A family of hash functions for string hashing. https://github.com/google/cityhash.

[10] Jinglei Ren, Jishen Zhao, Samira Khan, Jongmoo Choi, Yongwei Wu, and Onur Mutlu. 2015. ThyNVM: Enabling Software-transparent Crash Consistency in Persistent Memory Systems. In *Proceedings of the 48th International Symposium on Microarchitecture (MICRO-48)*. ACM, New York, NY, USA, 672–685. https://doi.org/10.1145/2830772.2830802

[11] Andy Rudoff. 2017. Persistent Memory Programming. https://www.usenix.org/publications/login/summer2017/rudoff. *USENIX ;login:* 42, 2 (2017).

[12] William Wang and Stephan Diestelhorst. 2018. Quantify the Performance Overheads of PMDK. In *Proceedings of the International Symposium on Memory Systems (MEMSYS '18)*. ACM, New York, NY, USA, 50–52. https://doi.org/10.1145/3240302.3240423