Performance characterization of a DRAM-NVM hybrid memory architecture for HPC applications using Intel Optane DC Persistent Memory Modules

Onkar Patil[1]
[1]North Carolina State University
opatil@ncsu.edu

Abstract

Non-volatile, byte-addressable memory (NVM) has been introduced by Intel in the form of NVDIMMs named Intel® Optane™ DC PMM. This memory module has the ability to persist the data stored in it without the need for power. This expands the memory hierarchy into a hybrid memory system due the differences in access latency and memory bandwidth from DRAM, which has been the predominant byte-addressable main memory technology. The Optane DC memory modules have up to 8x the capacity of DDR4 DRAM modules which can expand the byte-address space up to 6 TB per node. Many applications can now scale up the their problem size given such a memory system. We evaluate the capabilities of this DRAM-NVM hybrid memory system and its impact on High Performance Computing (HPC) applications. We characterize the Optane DC in comparison to DDR4 DRAM with a STREAM-like custom benchmark and measure the performance for HPC mini-apps like VPIC, SNAP, LULESH and AMG under different configurations of Optane DC PMMs. We find that Optane-only executions are slower in terms of execution time than DRAM-only and Memory-mode executions by a minimum of 2 to 16% for VPIC and maximum of 6x for LULESH.

CCS Concepts • **Computer systems organization** → **Heterogeneous (hybrid) systems**; • **Hardware** → **Memory and dense storage**; • **Computing methodologies** → *Massively parallel and high-performance simulations*;

Keywords NVM, Persistent Memory, Intel Optane DC, Memory Allocation, Hybrid Memory, NUMA, SICM

ACM Reference Format:
Onkar Patil[1]. 2019. Performance characterization of a DRAM-NVM hybrid memory architecture for HPC applications using Intel Optane DC Persistent Memory Modules. In *Proceedings of the International Symposium on Memory Systems (MEMSYS '19), September 30-October 3, 2019, Washington, DC, USA.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3357526.3357541

MEMSYS '19, September 30-October 3, 2019, Washington, DC, USA
2019. ACM ISBN 978-1-4503-7206-0/19/09…$15.00
https://doi.org/10.1145/3357526.3357541

1 Introduction

Memory hierarchies have been constantly evolving since computers were introduced and the von Neumann architecture was adopted. Today, semiconductor memory is dominated by dynamic random access memory (DRAM) as its density is high while its cost is low [21]. DRAM is volatile, prone to soft errors, and more power-consuming due to the constant refreshing required to retain the stored data. As processor clock frequencies were scaled up, static random access memory (SRAM) was introduced as a caching layer to bridge the latency gap. The memory hierarchy kept expanding as multi-level caches were introduced, high-bandwidth memories were added, and the main memory sizes kept increasing. Data persistence, or non-volatility, is a feature of data storage, which is the secondary level of the current memory hierarchy. Most non-volatile devices are not on the memory bus like DRAM, but are much further away in terms of latency. The difference in access latencies between DRAM and other technologies used in storage made it cumbersome to scale the capacity or add persistence to the primary memory hierarchy.

Supercomputers are built with individual nodes, which have their own memory hierarchy. The nodes are connected to other nodes by high-speed interconnects, allowing for direct memory access or remote direct memory access. The combined memory of a cluster is greater than that of a single node but requires complex software and additional hardware and the scale comes with its own share of problems. For example, Oak Ridge National Laboratory's Titan [30], a petaflop machine, which is now decommissioned, is one of the fastest supercomputers on the TOP500 November 2018 list [31]. Each node has 38 GB of DRAM. As a cluster of 18,688 nodes connected over an interconnect, Titan has 710 TB of DRAM. However, the sheer number of DRAM modules used in the system causes it to be susceptible to soft errors and hard faults. Additionally, memory is one of the main components contributing to the power consumption of Titan, which can reach up to 8.2 MW at its peak. Due to the higher number of DRAM modules that will be required to achieve exascale memory requirements, the cost to build and operate a larger machine with a similar memory architecture will increase

significantly. This also increases the likelihood of failures [8]. Thus, an exascale machine with a similar architecture may require hardware innovations to address the challenge of resiliency and power.

Over the past decade, memory technologies such as phase change memory (PCM) and spin-transfer torque RAM have been developed and are now used to make byte-addressable non-volatile memory devices [1, 25]. Although they are slower, they have higher densities than DRAM. This trade-off requires a detailed analysis to evaluate the benefit of these new memory technologies. Intel has been the first-to-market with their Intel® Optane™ DC Persistent Memory Module (PMM), which is based on PCM technology. The Optane DC is plugged directly into the memory bus via traditional DIMM slots. It has 8 times higher density than DRAM and is cheaper per GB. The Optane DC PMMs can be used to expand the capacity of primary memory hierarchy with data persistence, or can be used as a traditional NVM block device.

Using memories with higher density that of DRAM will allow different design points for exascale computers. Fewer nodes can be used to reach higher aggregate memory capacities. Fewer nodes means fewer components, which in turn can lower the cost to build the system, reduce the overall power consumption of the system, and increase resiliency. Additionally, the data persistence of these new types of memory can also assist in the development of new fault-tolerance mechanisms.

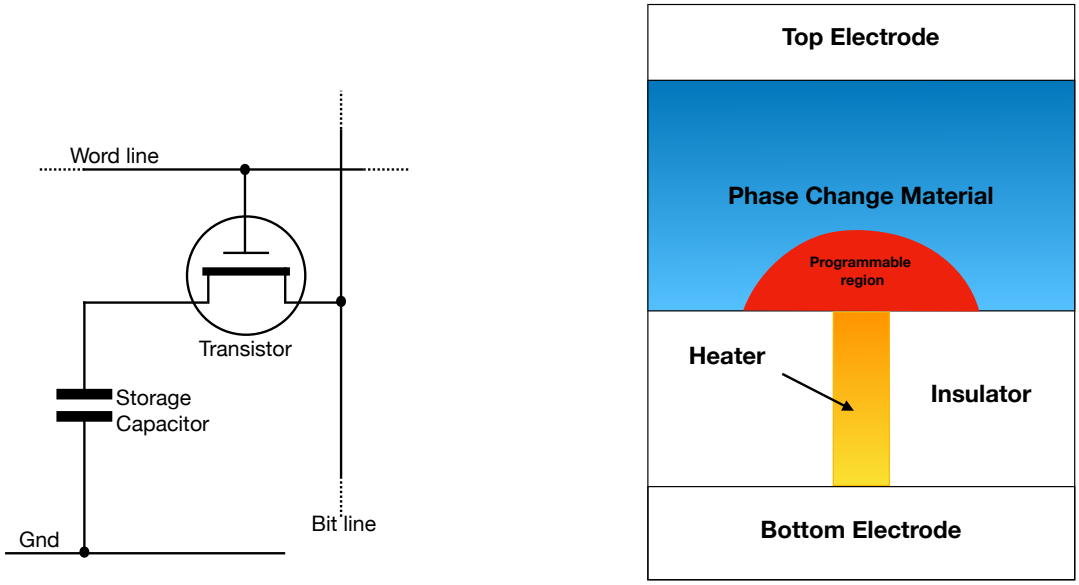
In this paper, we take a closer look at the Optane DC PMMs, its underlying technology, its operation, the different modes that it can operate in, and evaluate its performance for HPC applications. We focus on evaluating its use as main memory instead of part of the storage system. We have characterized the performance of Optane DC by using a custom benchmark inspired by the STREAM [18, 19], which has access streams that are frequently used in HPC applications. We have evaluated the overall system performance with HPC applications like VPIC [2], and proxy applications such as AMG [39], LULESH [13] and SNAP [29]. In Section 2, we review research related to persistent memory systems and their evaluation. In Section 3, we go over the background of non-volatile memory, and in Section 4 we focus on the Optane DC memory architecture. In Sections 5 and 6, we evaluate the performance of Optane DC. In Section 7, we present potential future work and follow up with the conclusion in Section 9.

2 Related Work

Due to the recent launch of Optane DC PMMs, there are not many previous works that evaluate the performance characteristics of the device.

In [10], Izraelevitz et al. evaluated the read and write characteristics of the Optane DC PMM. They evaluated the performance of Optane DC on all the modes available using the SPEC2017 benchmark suite. They found that applications experience a 15-61% slowdown with NVM-only allocations when compared to DRAM-only allocations. They also compared the performance of different filesystems and database applications like Mongo DB and MySQL by using Optane DC as persistent storage and persistent memory respectively. They found that Optane DC boosts the performance of filesystems and database applications due to lower latencies than storage devices. We are evaluating Optane DC for different streams that we encounter in different HPC applications and focus on using Optane DC as an extended address space for the same. Gill et al. [7] used the Optane DC PMM to evaluate shared-memory graph frameworks like Galois on real world web-crawls. They found that Optane DC PMM yields performance and cost benefit for massive graph analytics when compared to a distributed graph frameworks on existing production clusters. Our work focuses on HPC problems that are mostly stencil codes and matrix operations. Psaropoulos et al. [24] worked on hiding the latency difference between Optane DC and DRAM for database applications by interleaving the execution of parallel work in index joins and tuple reconstruction using coroutines. They accelerated end-to-end query runtimes on both NVM and DRAM by up to 1.7x and 2.6x, respectively. Van Renen et al. [33] performed performance evaluations of Optane DC in terms of bandwidth and latency and developed guidelines for efficient usage of Optane DC and two tuned I/O primitives, namely log writing and block flushing. Their work is primarily based on the App-direct mode (modes are explains in Section 4) and is aimed to improve the performance of file-systems. Wu et al. [37] studied the I/O performance of an early version of Optane DC, which was 3D-Xpoint with NFS and PVFS [14] as the filesystems. It operated similar to the current App Direct Mode in Optane DC.

There has been a lot of work in providing software support for byte-addressable non-volatile memories. Volos et al. [35] created a simple interface for programming with persistent memory called Mnemosyne. It allows programmers to allocate global persistent and dynamic data structures and also primitives to modify the data structures. Coburn et al. [4] implemented a lightweight, persistent object system called NV-heaps. It provides transactional semantics that prevent errors and a persistence model for heap objects. Chakrabarti et al. [3] proposed a system with durability semantics for a lock-based code called Atlas. It automatically maintains a globally consistent state in the presence of failures. Dulloor et al. [6] implemented a POSIX file system, PMFS, that exploits persistent memory's byte-addressability to avoid overheads of block-oriented



(a) DRAM cell **(b)** PCM cell

Figure 1. Memory cells of DRAM and PCM

storage and enable direct persistent memory access by applications with memory-mapped I/O. Yang et al. [38] implemented and evaluated the performance of a non-volatile B^+-Tree called NV-Tree and a key-value store based on it for NVDIMM-based servers. Shull et al. [27] proposed a user-friendly NVM framework for Java that ensures consistent stores for crash-recovery operations.

Various ways have been proposed to use NVM for HPC systems. Vetter et al. [34] evaluate the potential for NVM systems for extreme-scale HPC. They look at various persistence devices for integration of NVM in HPC and also look at integrating the functionalities of NVM. Kannan et al. [12] optimized checkpoints for HPC application using NVM as a virtual memory and provide frequent, low overhead checkpoints. Patil et al. [23] proposed a novel programming technique for stencil codes that guarantees fault tolerance against two hard failures on a shared non-volatile memory pool. Li et al. [16] proposed a fault tolerance process model based on NVRAM, which provides an elegant way for the applications to tolerate system crashes. Wang et al. [36] proposed a novel approach for exploiting NVM as a secondary memory partition so that applications can explicitly allocate and manipulate memory regions therein. It had a library that enabled access to a distributed NVM storage system.

3 Background

The memory hierarchy in modern architectures is complex and deep [21]. Register memory is closest to the processor, which is used to load and store operators, operands and instructions. It is implemented using flip-fops or an array of SRAM cells. It is the fastest memory, very expensive and consumes a lot of chip space and power. Most modern CPUs have 16 to 32 registers, which can hold 32 or 64 bits each. The access time for register memory is less than 1 ns. As computer programs require a lot more memory than there is on registers, we use a cheaper and higher density memory than registers as our main memory.

Main memory is comprised of arrays of DRAM cells as shown in Fig. 1a. It is a semiconductor-based memory technology that stores one bit of data in a capacitor within an integrated circuit. It is a rectangular array of cells that store a charge and are made of a capacitor and transistor per data bit. The number of cells define the capacity of a DRAM chip. There are positive and negative bit lines that connect all the cells in a column. A pair of cross connected inverters between the bit lines, called a sense amplifier, are used to stabilize the charges stored in the cells. DRAM has to be constantly refreshed to maintain its state due to the charge leak in the cells [9]. The JEDEC standard [11] specifies that each row has to be refreshed every 64 ms or less. Due to this constant need of refreshing the charges, this memory uses much more energy. The access time to DRAM using the DDR4 protocol is approximately 50-100 ns. Access to DRAM is 10-15x slower than register access time, which can lead to many CPU stalls. Processors use SRAM caches as buffers to hide this latency. SRAM uses latches to store each bit. They are volatile and lose their state when the memory is not powered. It is termed as static because it does not require refreshes. Modern memory architectures have leveraged multi-level caches to reduce the effective latency between the main memory and the processor. The

access time to SRAM cache is on an average 1-10 ns depending on the level of cache. SRAM is more expensive in chip space and power than DRAM.

This memory hierarchy has evolved and become more complex over multiple decades. It has grown in size and also become faster due to the scaling of DRAM capacity and frequency over the last 2 decades. But DRAM scaling has been approximately 33% slower than core count scaling over the same period of time. Also, due to the increased number of memory cells and higher refresh rates, DRAM energy consumption has increased [17, 20, 26]. Even though DRAM capacity has increased, due to higher densities, these memories have become less reliable [8]. Extreme scale problems in HPC, machine learning, graph analytics, and other fields can exhaust in-node memory capacity and processing. [34]. Using NVM as a supplement to DRAM, in order to increase the size of main memory in a compute node, has been considered to be a viable option [15]. Of all the NVM technologies, PCM has evolved best in terms of engineering [15]. PCM is a resistive memory whereas DRAM is a charge memory. As shown in Fig. 1b, PCM has bit-line, which is a metal connected to a phase-changing material via a heater. When a current pulse is passed through the bit-line, a phase is set in the phase-change material and stored there until another current pulse is passed. The phase is read by detecting the resistance of the material through the access line. The phase-change material retains its phase for more than 10 years at ambient temperature [15]. This property gives PCM its non-volatility. PCM is expected to scale down to 9 nm whereas scaling DRAM smaller than 40-35 nm is challenging [20].

However, PCM has its own challenges and shortcomings. It has a higher write latency than that of DRAM due to the thermal activation required to change the phase-change material. PCM also suffers from wear due to thermal expansion and contractions of the contacts between the bit-line and phase-change material. The write-durability of PCM memory cell is approximately 10^8, which means frequent device replacements are required that can add to the cost [15]. Despite these disadvantages, PCM provides the capability of scaling the main memory capacity required to match core count scaling. Intel's Optane DC PMMs are based on PCM technology.

4 Architecture

The system that we use for our experiments is a single node provided by the Intel Corporation. As described in Table 1, this node has 2 CPU sockets that are equipped with Intel's Xeon® 8260L (codenamed Cascade Lake). Each chip has 24 cores with a clock frequency of 2.4 GHz. Each core has 2 processing units for a total of 96 CPUs. Each core has a 32 KB private L1 instruction cache, a 32 KB private data cache, and a private 1 MB L2 cache. There is a 36 MB L3 cache shared between all cores. Each socket has 12 DIMM slots. 6 of the slots are occupied by 16 GB DDR4 DRAM modules and the other 6 slots are occupied by 128 GB Optane DC modules. That totals up to 192 GB of DRAM and 1.5 TB of non-volatile memory. The node has 4 memory controllers in total. Two of the memory controllers are connected to 6 DRAM DIMMs each, and the other two, known as iMC, are connected to 6 NVDIMMs each.

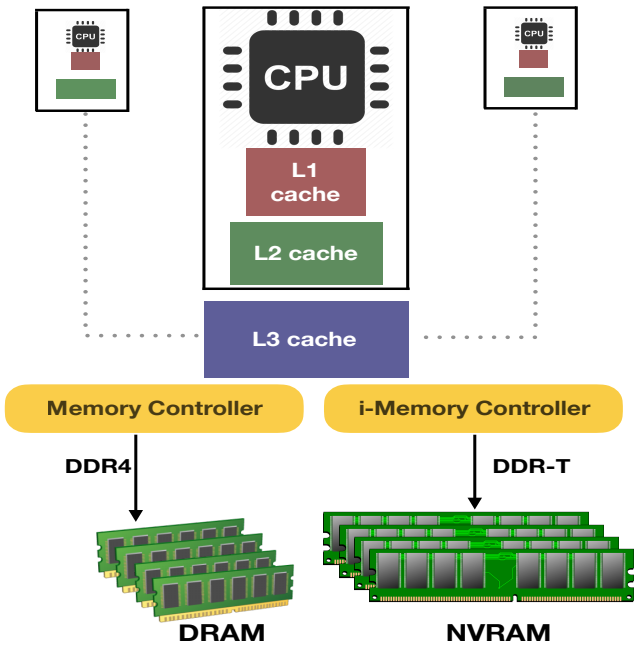


Figure 2. Memory architecture of Intel's Optane DC Node

Table 1. Experiment Platforms

Specifications	Optane Node
Model name	Intel(R) Xeon(R) 8260L @ 2.40GHz
Architecture	x86_64
CPUs	96
Sockets	2
Cores per socket	24
NUMA nodes	4
L1d cache	32 KB
L1i cache	32 KB
L2 cache	1 MB
L3 cache	35.3 MB
Memory Controllers	4
Channels/controller	6
DIMM protocol	DDR4
DRAM size	192 GB
NVDIMM protocol	DDR-T
NVRAM size	1.5 TB
Operating System	Fedora 27

As shown in Figure 2, the processor communicates differently with Optane DC DIMMs than with DRAM. For DRAM, it uses the standard DDR4 protocol via the regular

memory controller whereas for Optane DC it uses the DDR-T protocol via the i-memory controller (iMC). Using this proprietary add-on protocol to the DDR4 protocol, the Optane DC achieves asynchronous command/data timing and variable-latency memory transactions. To communicate with the iMC, the module controller in Optane DC PMM uses a request/grant scheme. The direction and timing of the data bus on which Optane DC resides is controlled by the processor. The processor sends a command packet per request to the Optane DC memory controller. The modules use 256 byte cache line access granularity which is larger than the 64 byte cache line access granularity used in DRAM.[10]. Intel's asynchronous DRAM refresh (ADR) guarantees the CPU stores that reach it, will survive a power failure. The stores are flushed to NVDIMMs in less than 100 µs, which is the hold-up time. The iMC falls in the ADR domain but the caches do not. So, the stores will be persistent only after they reach iMC for which it uses a 72-bit data bus and transfers data in cache line granularity for CPU loads and stores. Optane DC has an on-DIMM Apache Pass controller that handles memory access requests and the processing required on NVDIMM. The on-DIMM controller internally translates the addresses of all access requests for wear-levelling and bad-block management. It maintains an address indirection table on-DIMM that translates the DIMM's physical addresses to an internal device address. The table is also backed up on DRAM. Accessing data on Optane DC occurs after the translation. The controller translates 64 byte load/stores into 256 byte accesses due to the higher cache line access granularity of Optane DC which causes write amplification [10].

The Optane DC operates in three different modes. With a minor Linux kernel modification, we have configured the Optane DC to operate in a fourth mode. The configurations are described below.

4.1 Memory Mode

In Memory Mode, the Optane DC modules act as volatile main memory. DRAM acts as a direct-mapped cache for Optane DC with a block size of 4 KB and is managed by the CPU's memory controller. DRAM is no longer directly accessible, but allows for cache hits to be as fast as a DRAM access. Cache misses, however, can take as long as a DRAM cache miss plus an Optane DC access.

4.2 App Direct Mode

In App Direct Mode, the Optane DC modules act as persistent memory devices that are separate from main memory. DRAM is used as main memory. However, the Optane DC DIMMs are used through the block device entries that are created in the kernel. Once filesystems are installed on each device, the Optane DC modules are used as filesystems with significantly shorter access times than that of regular storage devices.

4.3 Mixed Mode

The Optane DC modules can also be partitioned to use part of the memory for persistent memory while the other part is used as volatile main memory. DRAM is still used as a cache for main memory rather than exposed as it would be in full App Direct Mode.

4.4 DRAM-NVM Hybrid Mode (Flat Mode)

Accessing both DRAM and Optane DC at the same time under a unified byte-addressable address space is not possible under the previous configurations. In [10], Izraelevitz et al. ran experiments with their Optane DC modules not being cached by the system's DRAM by modifying the Linux kernel to recognize the Optane DC modules as RAM instead of persistent memory. We applied the changes to our node's kernel and set the DIMMs to App Direct mode, allowing us to see the Optane DC modules on NUMA nodes in addition to the DRAM NUMA nodes, which results in a combined main memory of the DRAM capacity plus the Optane DC capacity rather than only one or the other.

Table 2. Optane DC operation modes

Operation mode	Functionality
Memory mode	Optane DC PMMs act as volatile, byte-addressable main memory. DRAM acts as a cache for Optane DC and is not visible to the user
App Direct mode	Optane DC PMMs act as persistent storage separate from the primary memory hierarchy. Managed by file systems installed on it. DRAM acts as main memory
Mixed mode	Part of Optane DC PMMs can be used as main memory and the remaining part can be used as persistent storage. DRAM acts as cache for Optane DC
Flat mode	DRAM and Optane DC PMMs are part of the same address space and can be used as heap memory

5 Experiments

Our aim is to evaluate Optane DC as an address space extender for main memory in HPC systems. We used an HPC application (VPIC), three HPC proxy-apps (AMG, LULESH, and SNAP) and a custom benchmark to evaluate the performance of Optane DC. We modified these

applications[1] so we can allocate all the data either on the DRAM, or on Optane DC. We then compared the statistics we collected for both configurations. We performed a preliminary performance characterization using a custom STREAM-like benchmark that evaluates the performance of different types of kernels encountered in HPC applications. Memory bandwidth information was collected for every stream used in a kernel that was parallelized using OpenMP [5]. The streams used in the benchmarks are representative of most of the streams found in HPC applications. We focus on different access patterns of data structures like sequential, strided, and random access. We also collect bandwidth numbers for matrix accesses and operations, for example, row-major access and stencil operations. We have a test where we bypass the L3 cache by accessing elements that are apart by number of elements that fit in L3 cache.

Our experiments ran on the Optane DC node described in Section 4 and Table 1. We set the Optane DC modules in Memory Mode and the DRAM-NVM Hybrid Mode (Flat Mode), and compared the performance of all the applications for each mode. In Flat mode, we allocated memory on NVM and DRAM for different runs. In memory mode, memory was only allocated on NVM as DRAM was used as cache for NVM.

We performed strong and weak scaling for all of the HPC mini-apps and measured the total execution times, memory bandwidth, power consumption, last-level cache misses and double-precision floating point operations per second. We used LIKWID [32] to collect performance counters. For our custom benchmark, we collected only the memory bandwidth for different kernels that we test.

Keeping in mind that we had Optane DC PMMs on only a single node, our experiments were not conducted on large number of processes or memory sizes. We ensured that our problem sizes were big enough to not fit into the last-level cache and so we can get a fair depiction of the performance of the different memories. Our problem sizes lie in a small/medium range as recommended by the authors of the mini-apps. We did not scale the number of processes to more than 48, i.e., half the number of processing units to avoid oversubscribing of resources. This was done in order to get correct performance numbers from the hardware performance counters. For the custom benchmark, we averaged the bandwidth measurements over 10 runs for every kernel which had a standard deviation up to 8%. For the HPC mini-apps, we average all measurements over 4 runs with a standard deviation of 11% for execution time. We describe the applications we use for our experiments below.

[1]In order to have the applications allocate their data on Optane DC or DRAM only, we modified the applications to use the Simple Interface for Complex Memory (SICM) [28] library, a NUMA-aware arena allocator for heterogeneous memory.

5.1 AMG

AMG is a parallel algebraic multi-grid solver for linear systems arising from problems on unstructured grids [39]. It was developed at Lawrence Livermore National Laboratory (LLNL). It is an SPMD code that uses MPI and OpenMP threading within MPI tasks. AMG is a highly synchronous code. The communication and computation patterns exhibit the surface-to-volume relationship common to many parallel scientific codes. We use the default Laplace type problem on a cube with a 27-point stencil.

5.2 LULESH

LULESH [13] is a highly simplified application, hard-coded to only solve a simple Sedov blast problem with analytic answers. It is C++ based applications. It was developed at LLNL as a part of co-design efforts for exascale computations. LULESH approximates the hydrodynamics equations discretely by partitioning the spatial problem domain into a collection of volumetric elements defined by a mesh. It uses MPI and OpenMP for parallelization and is also memory bound.

5.3 VPIC

Vector Particle-In-Cell (VPIC) [2] is a simulation code developed at Los Alamos National Laboratory (LANL). It is an application that models kinetic plasmas in 1 to 3 dimensions. It uses MPI and OpenMP for parallelism. The code is comprised of kernels that compute multiple data streams at the same time and operate on entire data structures. The data structures scale based on the input decks and hence make VPIC memory bound.

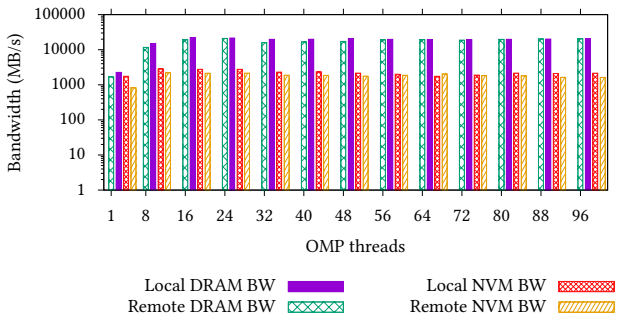
5.4 SNAP

SNAP [29] is based on the PARTISN code from LANL. SNAP mimics the computational workload, memory requirements, and communication patterns of PARTISN. The equation it solves has been composed to use the same number of operations, uses the same data layout, and loads elements of the arrays in approximately the same order. SNAP uses MPI to scale for HPC. We use the SNAP-C code. It is also a memory-bound application but is more bound by bandwidth than latency.

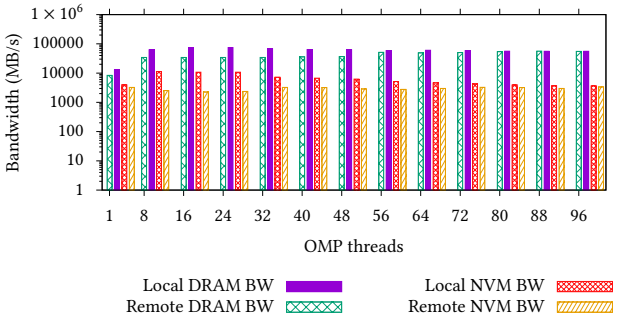
6 Results

6.1 Performance evaluation of different streams on Optane DC

We evaluate the performance of various streams executed on Optane DC compared to DRAM. We perform strong scaling on the streams by increasing the OpenMP threads from 1 to 96 for different runs. We pinned the threads using `numactl -C` to specific processing units and then allocated the streams on every NUMA node to evaluate the effect of NUMA distances on memory bandwidth. We

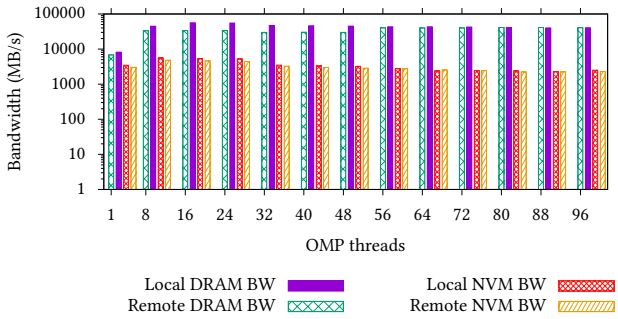


(a) Write-only stream bandwidth on the Optane node

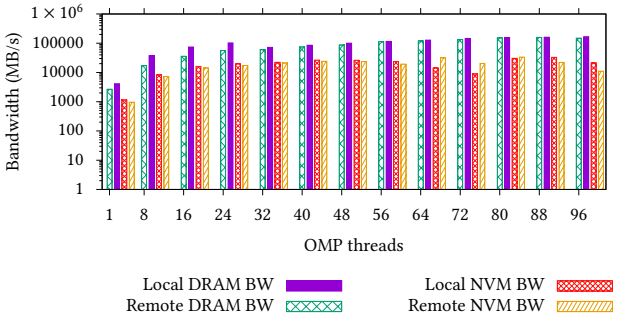


(b) 1 write+4 read stream bandwidth on the Optane node

Figure 3. Bandwidth measurement of Write-only and 1-write+4-read sequential access streams

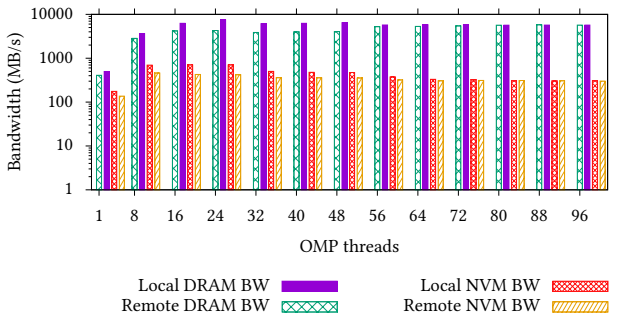


(a) Multiple stream kernel bandwidth on the Optane node

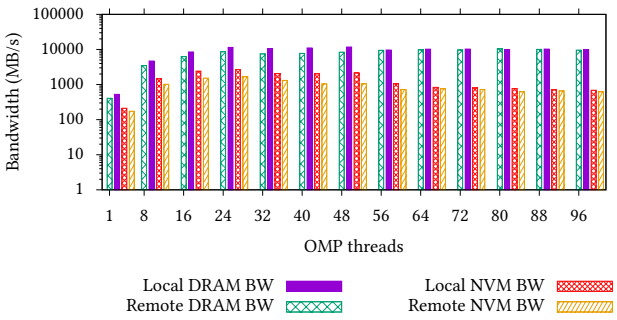


(b) Strided stream bandwidth on the Optane node

Figure 4. Bandwidth measurement of Multiple 1-write+4-read sequential access and Fixed stride access stream

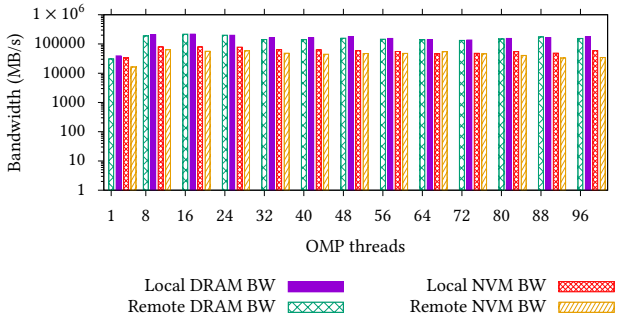


(a) Random stream bandwidth on the Optane node

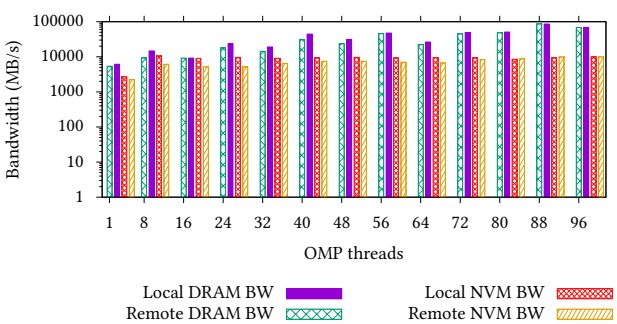


(b) LLC bypass stream bandwidth on the Optane node

Figure 5. Bandwidth measurement of Random and LLC bypass stride access stream



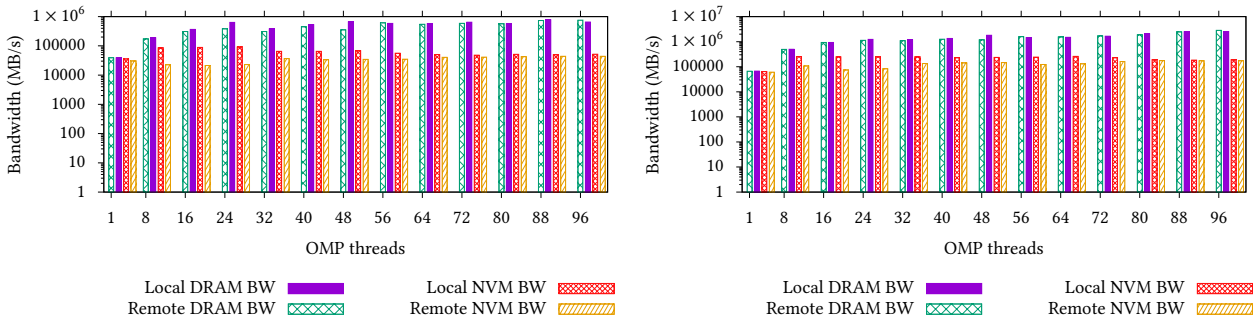
(a) Row major matrix stream bandwidth on the Optane node



(b) Column major matrix stream bandwidth on the Optane node

Figure 6. Bandwidth measurement of Row-major and Column-major matrix access stream

collected the effective bandwidth for all processing units and their local and remote NUMA node combinations and averaged the bandwidth results over 10 runs with a standard deviation of 7%. Every stream or data structure used in the experiment was 1 GB in size. All the plots have bandwidth on the Y axis which is depicted on a log scale and the number of threads on X axis.



(a) Row+column major matrix stream bandwidth on Optane node **(b)** 9-cell stencil stream bandwidth on the Optane node

Figure 7. Bandwidth measurement of Row-Column matrix and 9 point stencil stream

Fig. 3a depicts results for strong scaling of the write-only stream. The average bandwidth for local NUMA nodes peaks at 48 threads and then it plateaus. The bandwidth of remote NUMA nodes also increases with the number of OpenMP threads and peaks at 24 threads after which it plateaus as well. This effect is caused by oversubscribing of resources beyond 48 threads which causes the memory controller queues to overflow. This can lead to serialization of loads and stores due to back pressure and nullifies the benefit of bank parallelism. We observe that the NUMA distances affect bandwidth of NVM up to 22% and up to 16% for DRAM in case of strong scaling for the write only stream. Fig. 3b shows results for the kernel with a single write and four read stream. The same effect is observed as for the write-only stream but the relative bandwidth for each run is more than 3x the bandwidth of the write-only stream. This effect is observed for both DRAM and NVM, but DRAM achieves 8x higher bandwidth than NVM before oversubscribing. The difference between DRAM and NVM bandwidth worsens beyond 48 threads where they start to plateau. Such sequentially accessed streams are used mostly during initialization or problem generation phases of HPC applications. The above results indicate that utilization of local DRAM nodes in this phase is critical while not oversubscribing to compute resources.

Fig. 4a provides results for a kernel with multiple single write plus 4 read streams. The performance is similar to what is observed for the single write and four read stream but the effective bandwidth is slightly lower. The bandwidth for DRAM is 30% lesser than the single write and four read stream and 50% lesser for NVM. Such streams don't need a lot of parallelism to achieve maximum bandwidth however the access latency of the memory device will affect the performance. Fig. 4b depicts results for a kernel with a fixed stride access for increasing OpenMP threads. The stride is bigger than the cache line size. For this stream the NUMA distances have no effect on the memory bandwidth, except for fewer than 24 threads for DRAM. NVM scales similarly to the previous streams but achieves higher memory bandwidth. The memory bandwidth of DRAM keeps increasing with the number of threads until it peaks at 48 threads for NVM. This indicates that NVM bandwidth may be constrained by the core count irrespective of the access pattern. This stream achieves up to 8x the bandwidth of write-only stream for DRAM and 10x the bandwidth of the same for NVM. Fig. 5a shows results from a randomly accessed single write plus 4 read stream, which achieves the lowest effective bandwidth of all streams. The random accesses are determined by an indirection array that is initialized by the rand() function that generates the order of indices to access. The memory bandwidth for local DRAM and NVM nodes is up to 40% higher till 48 threads after which it plateaus and is unaffected by NUMA distances. This effect is observed because of not allowing the HW prefetcher to take advantage of any temporal locality. Hence, effective bandwidth is so low. Fig. 5b illustrates results for a single write plus 4 read stream forced to bypass the L3 cache for every access. In this stream, we observe that the bandwidth achieved by DRAM is up to 5x higher than NVM till 48 threads. The bandwidth remains plateaus for all NUMA nodes beyond that. This result indicates that if we take the caches out of the picture, the effective bandwidth achieved by Optane DC is not affected as much as DRAM despite the higher access latencies. Such streams with varying access patterns are common in the computation phase of HPC applications. It becomes essential to identify the access patterns of each linear stream at every stage of the program and place the stream in the most effective memory node given the amount of compute resources subscribed to.

Fig. 6a depicts results from a stream that accesses a single write and two read matrices in row-major order. The scaling pattern is similar to the write-only stream but the effective bandwidth is almost 10x the bandwidth for DRAM and up to 40x the bandwidth for NVM when compared to the same. This high bandwidth may be observed due to the large cache size and prefetching which take advantage of the spatial locality. Again, such a stream is a common occurence during the initialization phase of HPC applications and it can benefit from being placed in local DRAM memory. Fig. 6b assesses a stream accessing a single write and two read matrices in column-major order. Here,

the NVM bandwidth remains steady with the increase in threads with a slight advantage for local NVM nodes. However, for DRAM, NUMA distances do not make a big difference and the bandwidth keeps increasing with the increasing number of threads. The bandwidth is 3x lower for DRAM and 8x lower for NVM than the row-major stream for most threads, except for the 2 highest thread counts, where DRAM bandwidth jumps up to 86 GB/s. Fig. 7a shows results for a kernel that accesses a single write plus two read matrices stream, all in row-major except for the last read stream in column-major order. It achieves 4x higher memory bandwidth than the row major stream for both DRAM and NVM. It achieves such high bandwidth due to spatial locality and prefetching in the cache. The scaling pattern is similar to the row major access stream but achieves substantially higher bandwidth for both memories. Lower thread counts give advantage to local NUMA node but beyond 48 threads, there is no difference. Fig. 7b depicts results for a 9-point stencil kernel, which scales similarly Fig. 7a but with approximately 40x the bandwidth for all thread counts and memories. This stream achieves the highest bandwidth of all the streams due to a lot of spatial and temporal locality in the cache. THe bandwidth observed is effectively the bandwidth of the cache. Such matrix streams occur during the computation phase of a HPC application. Although NUMA distances do influence the effective bandwidth of these streams, the memory device used affects the bandwidth significantly with increasing number of threads. Also the effective use of cache locality can help in achieving higher performance for both memory devices.

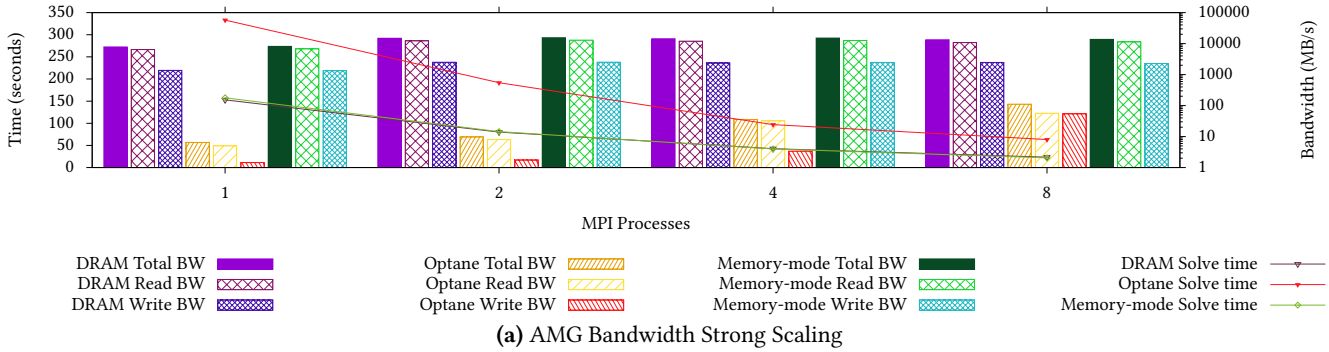
Taking all the results into account, we can infer that the higher latency of Optane DC and the lack of optimal cache support causes it to not perform as well as DRAM. We observe that with effective caching and prefetching, Optane DC can deliver much better performance than what is observed in our evaluation. However, these results give a fair idea of which workloads can benefit from NVM and gives a quantification of the performance impact by using NVM in place of DRAM.

6.2 HPC Benchmark evaluation

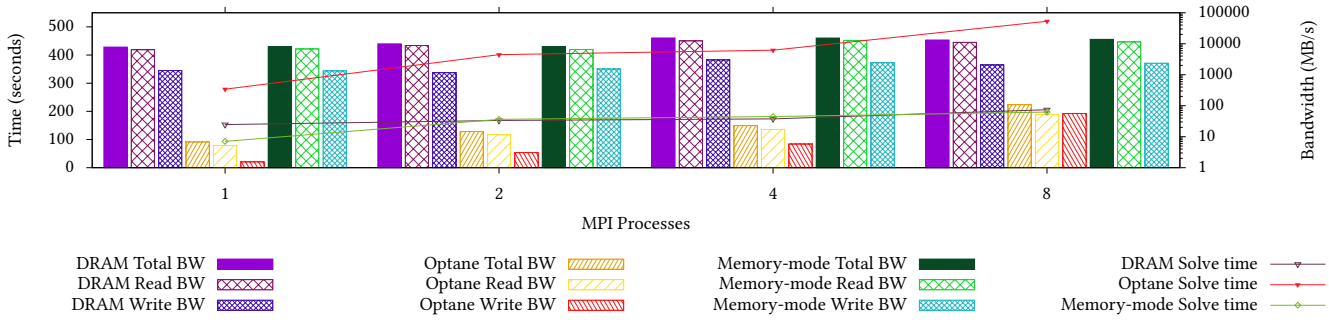
For benchmarks, we plot the application bandwidth and execution time measurements together in a single graph to observe their correlation. Similarly, we plot the energy consumption and execution time measurements together. We also plot the cycles/instruction (CPI) and L3 miss ratio together. We plot these graphs for both strong and weak scaling experiments. For bandwidth and execution time graphs, we plot execution time on the left-hand side y-axis in seconds and depicted as lines. The bandwidth is plotted on the right-hand side y-axis in megabytes/seconds (MB/s) and depicted as a bar chart. For energy consumption and execution time graphs, we again plot our execution time on the left-hand side y-axis as lines. The energy is plotted on the right-hand side y-axis in Joules (J) as a bar chart. For CPIs and L3 miss ratio graphs, we plot the CPIs on the left-hand side y-axis as lines. The L3 miss ratio is plotted on the right-hand side y-axis as a bar chart. Both CPIs and L3 miss ratio have no unit. Bandwidth is plotted on a logarithmic scale whereas all other measurements are plotted on a linear scale. The x-axis depicts the number of MPI processes for a given execution.

6.2.1 AMG results

Fig. 8a and Fig. 8b depict the graphs for strong and weak scaling of AMG, respectively. For strong scaling, we scale the processes from 1 to 8 using MPI and keep the data size constant by reducing the size per processor from 256 to 128. For weak scaling, we again scale the processes from 1 to 8 and keep the size per processor same at 256 as we scale the data size proportionally. We observe that memory bandwidth for Optane-only execution is 2 to 3 orders of magnitude lower than DRAM-only and Memory-mode executions. This results in more than 2x higher execution times for Optane-only execution. This result is observed for both strong and weak scaling cases. The observed bandwidth remains fairly constant for both strong and weak scaling across all number of processes for DRAM-only and Memory-mode executions but rises for Optane-only execution. The lower bandwidth for Optane-only execution is a result of the higher access latency of Optane DC. The Memory-mode execution matches the performance of DRAM-only execution because it uses DRAM as a cache. The problem sizes for this experiment are small enough to fit into DRAM. Hence, there is minimal difference between the performance of DRAM-only and Memory-mode executions. Fig. 9a and Fig. 9b depict the energy consumption and execution time of all three executions of AMG for strong and weak scaling. We observe that the energy consumption of Optane-only execution is 2x higher than DRAM-only and Memory-mode executions. This is due to its higher execution time even though the power consumed by Optane-only execution is lower than the other executions. Fig. 10a and Fig. 10b depict the L3 cache misses and Cycles/Instruction(CPI) for strong and weak scaling of AMG for all 3 executions. In strong scaling, we observe that the CPIs for Optane-only execution are higher for low number of processes. For higher number of processes they are almost equal to the other 2 executions. However, the L3 cache misses increase rapidly with the number of processes. This also explains the difference in execution times of Optane-only execution and other 2 executions. The increase in L3 cache misses in Optane-only execution is observed under weak scaling as well but the CPIs are consistently higher than the CPIs for the other 2 executions. AMG is a memory bound application that is heavily affected by memory access speeds. Hence, the Optane-only executions

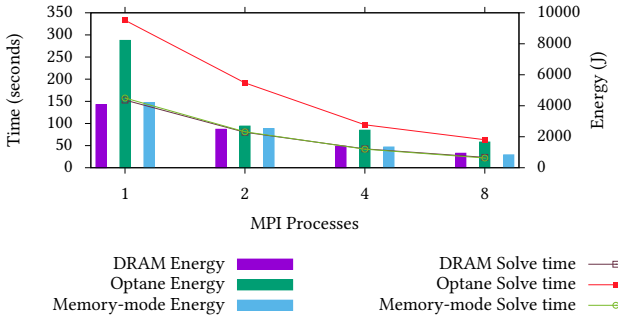


(a) AMG Bandwidth Strong Scaling

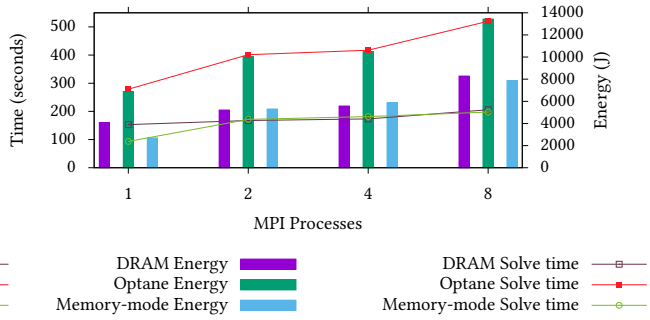


(b) AMG Bandwidth Weak Scaling

Figure 8. Bandwidth measurement for AMG

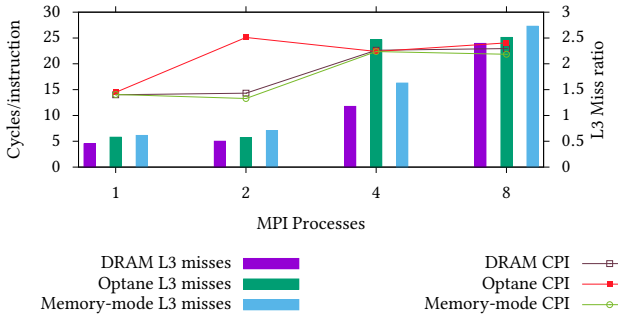


(a) AMG Memory Energy Consumption Strong Scaling

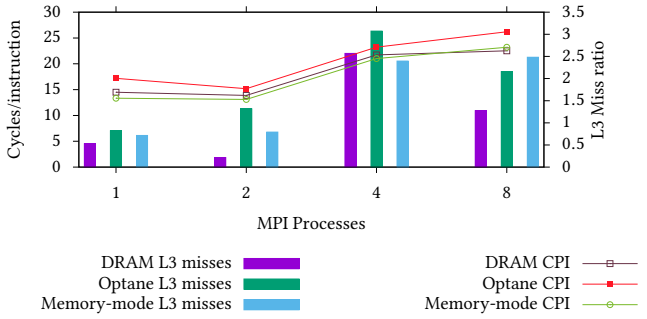


(b) AMG Memory Energy Consumption Weak Scaling

Figure 9. Memory energy consumption for AMG



(a) AMG L3 Miss Ratio and CPI Strong Scaling



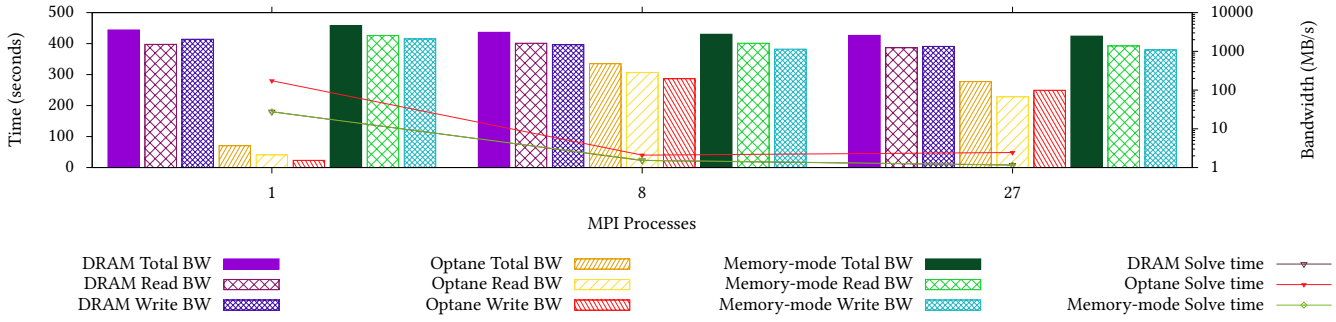
(b) AMG L3 Miss Ratio and CPI Weak Scaling

Figure 10. L3 miss ratio and CPI for AMG

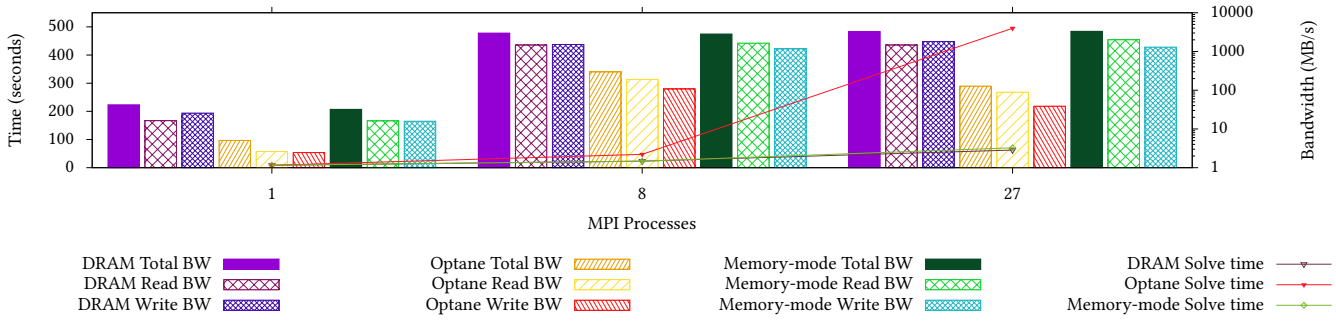
suffer from heavy performance degradation in terms of execution time and energy consumption. Such applications that require faster access speeds will suffer from a NVM-only approach.

6.2.2 LULESH results

Fig. 11a and Fig. 11b depict the graphs for strong and weak scaling of LULESH, respectively. We increase the number of processors from 1 to 27 using MPI, as LULESH accepts only cubes of natural numbers as a valid configuration. For

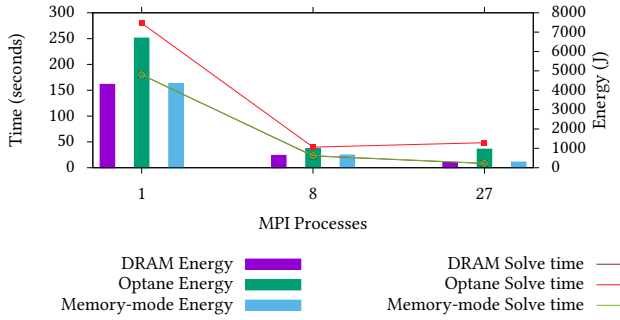


(a) LULESH Bandwidth Strong Scaling

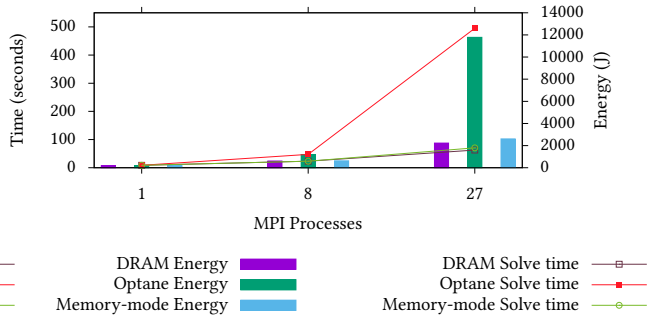


(b) LULESH Bandwidth Weak Scaling

Figure 11. Bandwidth measurement for LULESH

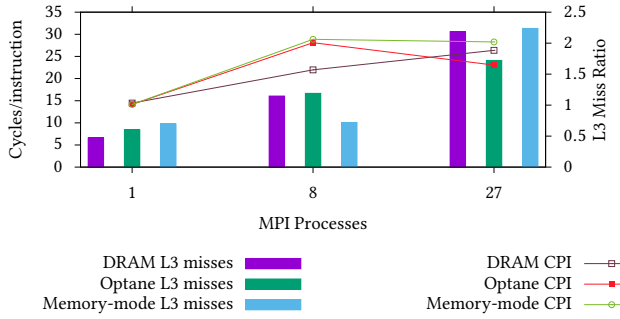


(a) LULESH Memory Energy Consumption Strong Scaling

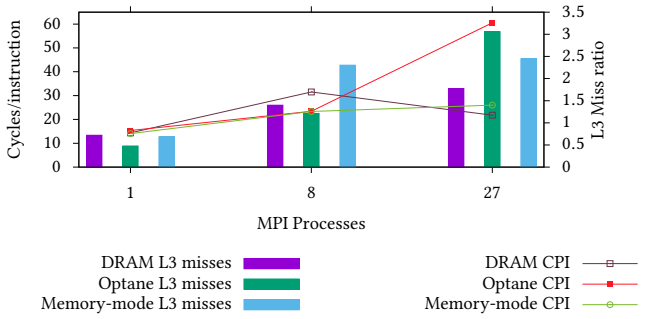


(b) LULESH Memory Energy Consumption Weak Scaling

Figure 12. Memory energy consumption for LULESH



(a) LULESH L3 Miss Ratio and CPI Strong Scaling



(b) LULESH L3 Miss Ratio and CPI Weak Scaling

Figure 13. L3 miss ratio and CPI for LULESH

strong scaling, we keep the problem size constant at 125000 data points and increase the number of processors. We observe that when LULESH is running only on Optane DC in flat mode, it has a 50% higher execution time than DRAM-only and Memory-mode configurations for a single process and 8 processes. For 27 processes, the execution time is approximately 6x higher. This effect is observed because the memory bandwidth is almost an order of magnitude lower than DRAM-only and Memory-mode executions. The difference in execution time is more

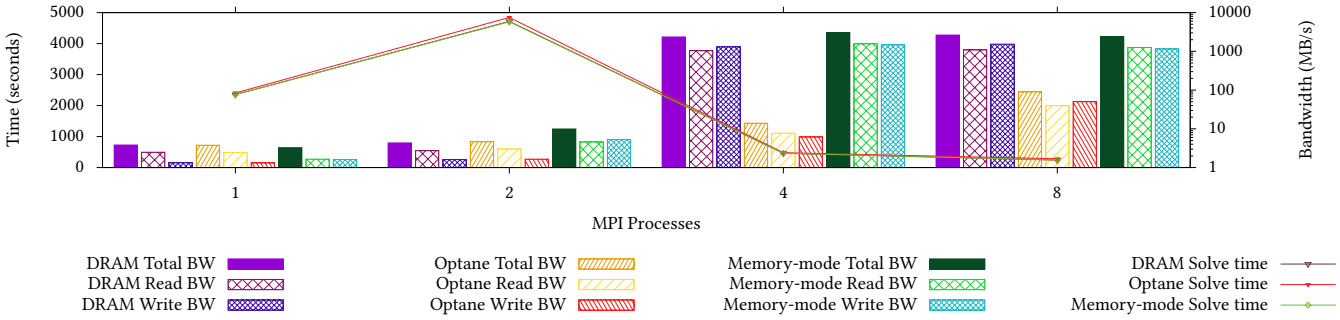
amplified in weak scaling for 27 processes due to lower bandwidth. In weak scaling, we keep the number of data points per process constant at 15,625. The low bandwidth on Optane-only execution is due to the high access latency of Optane DC. We do not observe this effect in Memory-mode because of the DRAM caching in Memory-mode. Hence, there is no difference in execution time for DRAM-only and Memory-mode execution as the access latency would be the same. In flat mode, Intel's ADR is at work to guarantee persistence of data, which can be hampering performance and the write amplification might increase access latencies, too. The memory bandwidth in weak scaling is similar to strong scaling except for single process execution in strong scaling, where Optane DC has 2 orders of magnitude lower bandwidth. Fig. 12a and 12b depict the memory energy consumption of LULESH in each mode for strong and weak scaling, respectively. We observe that the energy consumption for Optane DC is up to 60% higher than DRAM-only and Memory-mode executions. This is in direct correlation to the execution time because the power consumption for Optane DC is up to 30% less than DRAM. For smaller problem sizes under weak scaling with fewer threads the energy consumption of Optane DC is similar to the other executions. A trade-off between capacity, problem size and performance needs to be achieved to keep the application execution within desired energy budgets. Fig. 13a and Fig. 13b depict the L3 cache miss ratio and CPIs for LULESH strong and weak scaling, respectively. For strong scaling, the L3 cache misses increase with the number of processes but they are lower for Optane-only execution on 27 nodes. In weak scaling, the CPIs are 3x higher compared to DRAM-only and the L3 cache misses are significantly higher. These add to the execution time, explaining the difference in execution time for weak scaling of LULESH. Applications like LULESH are dependent on memory bandwidth for performance. These applications can reduce their energy consumption with NVM when they are running smaller problem sizes and fewer threads.

6.2.3 VPIC results

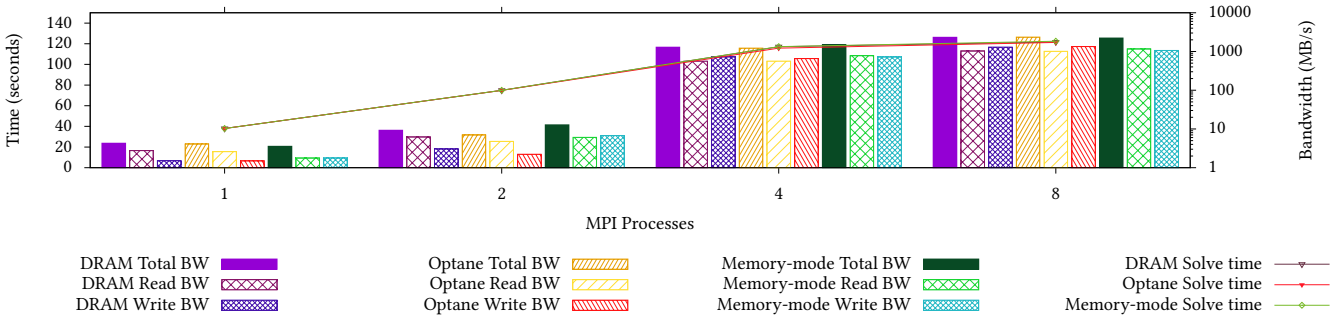
Fig. 14a and Fig. 14b depict the execution time and memory bandwidth for strong and weak scaling of VPIC. We use the 'lpi' input deck provided by the authors of the benchmark for our experiments. For strong scaling, we increase the number of processes from 1 to 8 using MPI and keep the problem size the same by changing the 'nppc' value from 2048 to 256. The 'nppc' variable in the input deck determines the number of particles/cell for each species in the plasma. We observe that NVM-only execution of VPIC is 2 to 16% slower than DRAM-only and Memory-mode executions. For weak scaling, we keep the problem size per process same by keeping the 'nppc' value at 512. This slowdown is caused by the lower bandwidth observed for Optane-only execution. Optane-only memory bandwidth is at least an order of magnitude lower than DRAM-only and Memory-mode bandwidth for larger number of processes in case of strong scaling. For weak-scaling, the memory bandwidth of Optane-only execution is similar to DRAM-only execution and Memory-mode execution and hence there is no difference in execution times either. Fig. 15a and Fig. 15b depict that the memory energy consumption for VPIC strong and weak scaling, respectively. The Optane-only execution's energy consumption remains constant with strong scaling of VPIC similar to the DRAM energy consumption. However under weak scaling, Optane-only execution's energy consumption rises at slower rate than the other 2 executions. As the execution times are similar for all execution under weak scaling, the energy consumed by all three executions is also similar. Fig. 16a and Fig. 16b depict the L3 cache miss ratio and CPIs for strong scaling and weak scaling of VPIC. Even though the cache misses increase with strong scaling for Optane DC execution, the CPIs remain lower than DRAM-only execution. The cache miss ratio for Memory-mode execution rises at a slower rate than NVM-only and DRAM-only execution. In weak scaling, Optane-only execution results in fewer L3 cache misses than DRAM-only and Memory-mode execution but higher CPIs. This keeps the execution times of Optane-only execution low for weak scaling. VPIC optimizes its cache hits, as seen in the results, to achieve higher performance and hence there is minimal difference in execution times of all three executions even though there is a significant difference in bandwidth. Such applications can benefit from Optane DC by reducing energy consumption while not compromising on performance.

6.2.4 SNAP results

Fig. 17a and Fig. 17b depict the memory bandwidth and execution time for strong and weak scaling of SNAP in Optane-only, DRAM-only, and Memory-mode executions. We use MPI to scale the number of processes from 1 to 8. We use the C-version of SNAP that is compiled with mpicc. Here, we observe that the execution times in strong scaling vary only slightly for all the three executions. Optane DC memory bandwidth increases with increasing number of processes and is the highest out of all three executions for 8 processes. This is reflected in the execution times of all 3 executions. However, for weak scaling the execution time increases for Optane-only execution when we scale up to 8 processes. We also observe that the memory bandwidth for Optane-only execution does not increase with weak scaling. This explains the 2x higher execution time for Optane-only execution compared to DRAM-only and Memory-mode execution. Fig. 18a and Fig. 18b depict the energy consumption of SNAP execution in Optane-only, DRAM-only, and Memory-mode executions. The energy

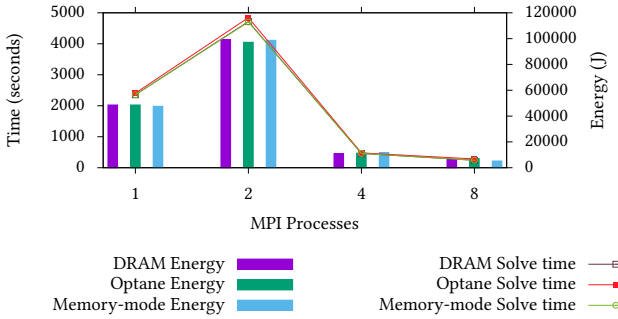


(a) VPIC Bandwidth Strong Scaling

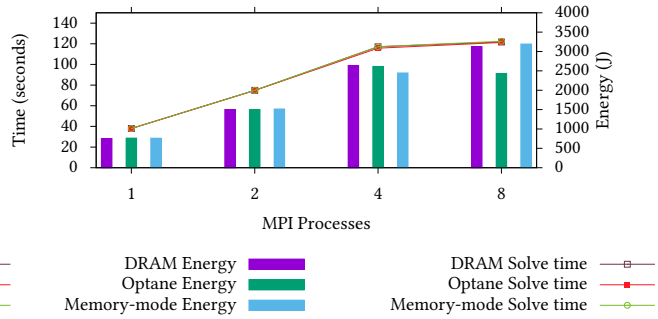


(b) VPIC Bandwidth Weak Scaling

Figure 14. Bandwidth measurement for VPIC

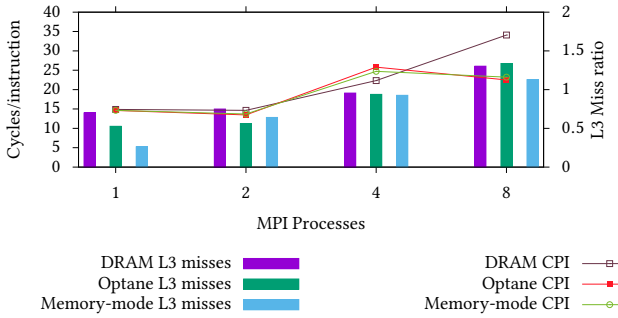


(a) VPIC Memory Energy Consumption Strong Scaling

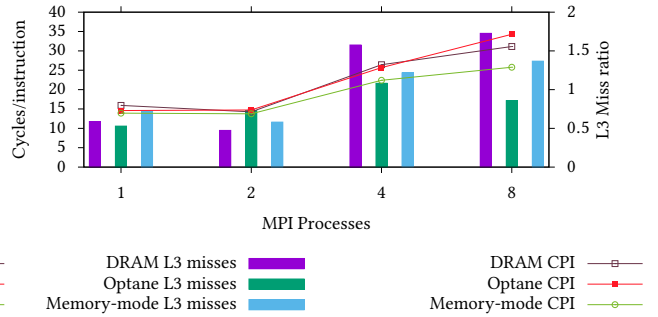


(b) VPIC Memory Energy Consumption Weak Scaling

Figure 15. Memory energy consumption for VPIC



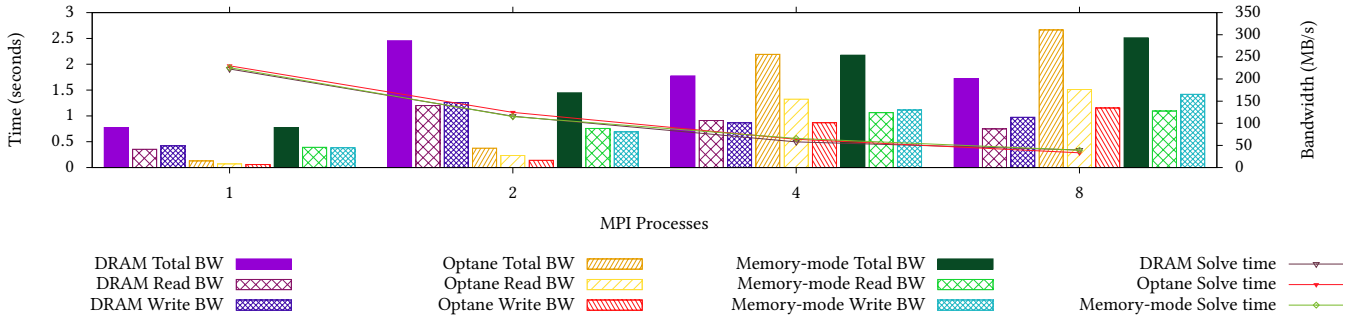
(a) VPIC L3 Miss Ratio and CPI Strong Scaling



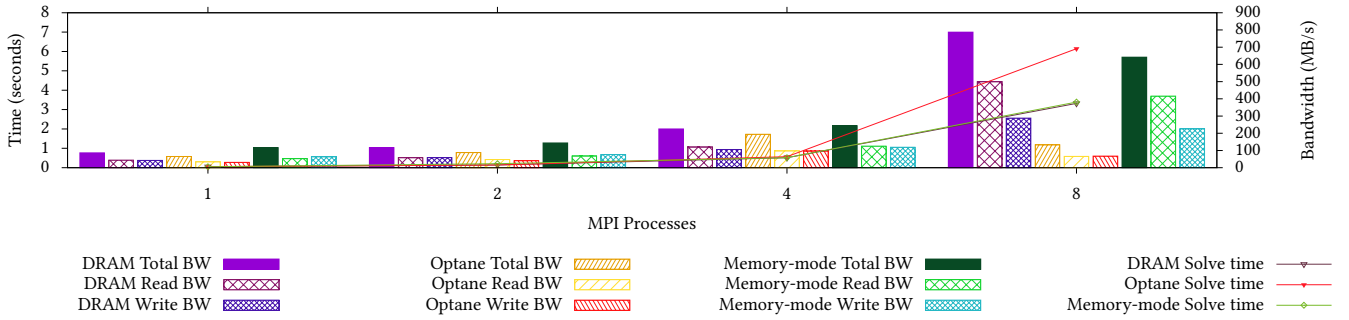
(b) VPIC L3 Miss Ratio and CPI Weak Scaling

Figure 16. L3 miss ratio and CPI for VPIC

consumption remain fairly constant across all three executions for both strong and weak scaling. As the execution times are similar for these executions the energy consumption of Optane-only execution is the least amongst the 3 executions, when the number of processes are 4. Under weak scaling, Memory-mode consumes the least power overall but due to larger execution times it consumes the maximum energy. Fig. 19a and Fig. 19b depict the plot of L3 cache miss ratios and CPIs for strong and weak scaling of SNAP. We observe that CPIs differ significantly for

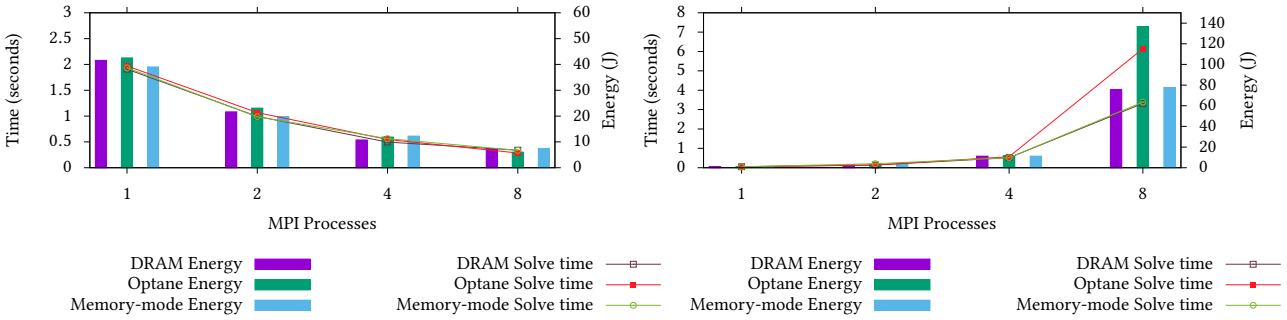


(a) SNAP Bandwidth Strong Scaling



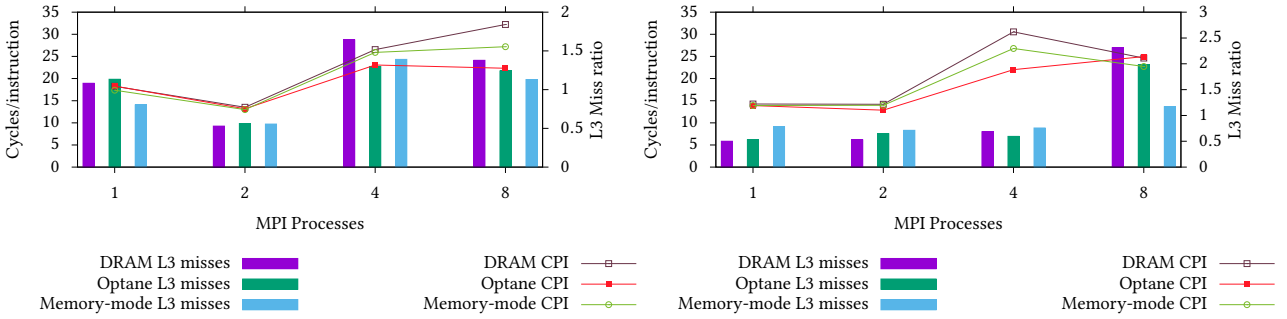
(b) SNAP Bandwidth Weak Scaling

Figure 17. Bandwidth measurement for SNAP



(a) SNAP Memory Energy Consumption Strong Scaling **(b)** SNAP Memory Energy Consumption Weak Scaling

Figure 18. Memory energy consumption for SNAP



(a) SNAP L3 Miss Ratio and CPI Strong Scaling **(b)** SNAP L3 Miss Ratio and CPI Weak Scaling

Figure 19. L3 miss ratio and CPI for SNAP

strong scaling at higher number of processes, where Optane-only execution experiences the least CPIs. The L3 cache misses actually rise with increasing number of processors for all three executions. The L3 cache misses scale with increasing number of processes for weak scaling, too. However, the changes in CPIs for higher number processes are erratic. The C-version of SNAP was created to take advantage of the vector operations in the Intel microarchitecture and is highly optimized to take advantage of the cache hierarchy and prefetching methods. Due to this,

the performance degradation is minimal with SNAP, which results in reduction in energy consumption on the low power Optane DC memory.

7 Future Work

Intel's Optane DC PMM opens up a range of possibilities for use of NVM in various applications. We plan to explore the use of NVDIMMs in optimizing HPC applications. The different modes in which you can operate Optane DC PMMs have potential to optimize many HPC applications. With our performance characterizations, we plan to develop allocation policies for NVM for different configurations. This will help accommodate large problems with fewer compute nodes and operate under a required compute and energy budget. There are many variable components to Optane DC that need to be characterized as well to take full advantage of the technology, for example, its variable latency and power consumption. We plan to study the longevity and varying latencies of NVDIMMs when used with HPC workloads and their susceptibility to faults and failures. This will help improve the resiliency of the supercomputers that would use NVDIMMs.

In addition to the massive memory capacity, NVDIMMs have data persistence, which can help develop novel resiliency techniques. They can be used to store lightweight checkpoints and restart processes that fail. We plan to explore the possibility of building a fast and lightweight checkpoint/restart mechanism for exascale supercomputers [22]. It can also be used to maintain metadata of large-scale systems and help in lookup operations. The data stored on NVDIMMs can be used to detect and correct soft errors by using checksums for increasing reliability. We will explore the use of NVDIMMs to increase the reliability of computations. We also plan to investigate the kernel and user-level support required for efficient use of NVDIMMs. Compiler-based analysis and profiling information can help optimize the use NVDIMMs for various applications. We will also assess support for other memory technologies that can be incorporated into the DRAM-NVM hybrid memory hierarchy.

8 Timeline

In this section, we have laid out a plan for the remaining time till the completion of Ph.D. Our work is going to focus on developing and providing software support for hybrid memory architectures.

- We will check the feasibility of DRAM-NVM hybrid memories to support large problems on fewer nodes. For this task, we will set up an experiment where we will compare a DRAM-NVM hybrid memory system with a small set of nodes equipped with a high bandwidth interconnect and an equivalent amount of DRAM memory. We will run large problem sizes for HPC benchmarks on these systems and then compare their performance in terms execution time and energy along with other metrics. We expect this research to result into a conference paper. (1-2 months)

- We want to explore the effects of different caching and prefetching methods on HPC applications while using a hybrid memory architecture. We will evaluate how the current caching and prefetching methods affect the performance of HPC applications while using NVM and then develop solutions that can improve the performance of HPC applications. We expect this work to also result into a conference publication. (2-3 months)

- We also want to explore and evaluate allocation policies for hybrid memory architectures that optimize HPC performance. Based on the performance characterization of DRAM and NVM in a hybrid memory system, we want to develop data allocation and movement policies that can achieve close to peak performance of HPC applications. (2-3 months)

- We want to develop compiler support for hybrid memory architectures where we can automate the data allocation and data movement policies statically and optimize HPC applications in order to relieve the programmer from changing applications for hybrid memory systems. We expect that the combination of this work and above mentioned work to result into a conference publication with the possibility of a journal extension. (2-3 months)

Here is also a list of publications that I have till date apart from this paper.

- (Technical poster) Exploring Use-cases for Non-Volatile Memories in support of HPC Resilience (SC 2017) *O. Patil, S. Hukerikar, F. Mueller, C. Englemann*

- (Workshop paper) Persistent Regions that Survive NVM Media Failure (NVM 2017) *O. Patil, M. Kuscu, T. Tran, C. Johnson, J. Tucek, H. Kuno*

- (Conference paper) Efficient & Predictable Group Communication Messaging over Manycore NoCs (ISC 2016) *K. Yagna, O. Patil, F. Mueller*

- (Conference paper) End-to-end Resilience for HPC Applications (ISC 2019) *A. Rezai, H. Khetawat, O. Patil, F. Mueller, P. Hargrove, E. Roman*

9 Conclusion

In this paper, we performed characterization of a hybrid memory system comprising of a slower NVM device and a

faster DRAM device. We conclude that using a slower byte-addressable memory device hampers the performance of memory-bound HPC applications due to higher access latencies and lower memory bandwidth. However, using the DRAM as a cache for the slower NVM device maintains the performance of HPC applications observed on DRAM-only memory systems while increasing the memory capacity of the system, which needs to be further verified on large problem sizes. Although using NVM as main memory directly hampers the performance, it has the potential to reduce the energy consumption of HPC applications with reasonable trade-offs. Optane DC PMMs enables us to close the gap between core count and memory capacity scaling.

Acknowledgments

This material is based upon work supported by United States Department of Energy National Nuclear Security Administration prime contract #89233218CNA000001 subcontract #508854 dated November 1, 2018 for Los Alamos National Laboratory, NSF grants 1217748 and 1525609 and by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.

References

[1] Dmytro Apalkov, Alexey Khvalkovskiy, Steven Watts, Vladimir Nikitin, Xueti Tang, Daniel Lottis, Kiseok Moon, Xiao Luo, Eugene Chen, Adrian Ong, Alexander Driskill-Smith, and Mohamad Krounbi. 2013. Spin-transfer Torque Magnetic Random Access Memory (STT-MRAM). *J. Emerg. Technol. Comput. Syst.* 9, 2, Article 13 (May 2013), 35 pages. https://doi.org/10.1145/2463585.2463589

[2] K J Bowers, B J Albright, L Yin, W Daughton, V Roytershteyn, B Bergen, and T J T Kwan. 2009. Advances in petascale kinetic plasma simulation with VPIC and Roadrunner. *Journal of Physics: Conference Series* 180 (jul 2009), 012055. https://doi.org/10.1088/1742-6596/180/1/012055

[3] Dhruva R Chakrabarti, Hans-J Boehm, and Kumud Bhandari. 2014. Atlas: Leveraging locks for non-volatile memory consistency. In *ACM SIGPLAN Notices*, Vol. 49. ACM, 433–452.

[4] Joel Coburn, Adrian M Caulfield, Ameen Akel, Laura M Grupp, Rajesh K Gupta, Ranjit Jhala, and Steven Swanson. 2012. NV-Heaps: making persistent objects fast and safe with next-generation, non-volatile memories. *ACM Sigplan Notices* 47, 4 (2012), 105–118.

[5] Leonardo Dagum and Ramesh Menon. 1998. OpenMP: An Industry-Standard API for Shared-Memory Programming. *IEEE Comput. Sci. Eng.* 5, 1 (Jan. 1998), 46–55. https://doi.org/10.1109/99.660313

[6] Subramanya R Dulloor, Sanjay Kumar, Anil Keshavamurthy, Philip Lantz, Dheeraj Reddy, Rajesh Sankaran, and Jeff Jackson. 2014. System software for persistent memory. In *Proceedings of the Ninth European Conference on Computer Systems*. ACM, 15.

[7] Gurbinder Gill, Roshan Dathathri, Loc Hoang, Ramesh Peri, and Keshav Pingali. 2019. Single Machine Graph Analytics on Massive Datasets Using Intel Optane DC Persistent Memory. *CoRR* abs/1904.07162 (2019). arXiv:1904.07162 http://arxiv.org/abs/1904.07162

[8] Saurabh Gupta, Tirthak Patel, Christian Engelmann, and Devesh Tiwari. 2017. Failures in large scale systems: long-term measurement, analysis, and implications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 44.

[9] Hideto Hidaka, Yoshio Matsuda, Mikio Asakura, and Kazuyasu Fujishima. 1990. The cache DRAM architecture: A DRAM with an on-chip cache memory. *IEEE Micro* 10, 2 (1990), 14–25.

[10] Joseph Izraelevitz, Jian Yang, Lu Zhang, Juno Kim, Xiao Liu, Amirsaman Memaripour, Yun Joon Soh, Zixuan Wang, Yi Xu, Subramanya R. Dulloor, Jishen Zhao, and Steven Swanson. 2019. Basic Performance Measurements of the Intel Optane DC Persistent Memory Module. *CoRR* abs/1903.05714 (2019). arXiv:1903.05714 http://arxiv.org/abs/1903.05714

[11] JEDEC (2017). JEDEC DDR4 SDRAM standards. https://www.jedec.org/standards-documents/docs/jesd79-4a

[12] Sudarsun Kannan, Ada Gavrilovska, Karsten Schwan, and Dejan Milojicic. 2013. Optimizing checkpoints using nvm as virtual memory. In *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*. IEEE, 29–40.

[13] Ian Karlin, Jeff Keasler, and Rob Neely. 2013. *LULESH 2.0 Updates and Changes.* Technical Report LLNL-TR-641973. 1–9 pages.

[14] Rob Latham, N Miller, Robert Ross, P Carns, and Clemson Univ. 2004. A next-generation parallel file system for Linux cluster. *LinuxWorld Mag.* 2 (01 2004).

[15] Benjamin C Lee, Engin Ipek, Onur Mutlu, and Doug Burger. 2009. Architecting phase change memory as a scalable dram alternative. *ACM SIGARCH Computer Architecture News* 37, 3 (2009), 2–13.

[16] Xu Li, Kai Lu, Xiaoping Wang, and Xu Zhou. 2012. NV-process: a fault-tolerance process model based on non-volatile memory. In *Proceedings of the Asia-Pacific Workshop on Systems*. ACM, 1.

[17] Kevin Lim, Jichuan Chang, Trevor Mudge, Parthasarathy Ranganathan, Steven K Reinhardt, and Thomas F Wenisch. 2009. Disaggregated memory for expansion and sharing in blade servers. In *ACM SIGARCH computer architecture news*, Vol. 37. ACM, 267–278.

[18] John D. McCalpin. 1991-2007. *STREAM: Sustainable Memory Bandwidth in High Performance Computers.* Technical Report. University of Virginia, Charlottesville, Virginia. http://www.cs.virginia.edu/stream/ A continually updated technical report. http://www.cs.virginia.edu/stream/.

[19] John D. McCalpin. 1995. Memory Bandwidth and Machine Balance in Current High Performance Computers. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter* (Dec. 1995), 19–25.

[20] Onur Mutlu. 2013. Memory scaling: A systems architecture perspective. In *2013 5th IEEE International Memory Workshop*. IEEE, 21–25.

[21] Ravi Nair. 2015. Evolution of memory architecture. *Proc. IEEE* 103, 8 (2015), 1331–1345.

[22] Onkar Patil, Saurabh Hukerikar, Frank Mueller, and Christian Engelmann. 2017. Exploring use-cases for non-volatile memories in support of hpc resilience. *SC Poster Session* (2017).

[23] Onkar Patil, Charles Johnson, Mesut Kuscu, Joseph Tucek, Tuan Tran, and Harumi Kuno. 2009. Persistent Regions that Survive NVM Media Failure. (2009).

[24] Georgios Psaropoulos, Ismail Oukid, Thomas Legler, Norman May, and Anastasia Ailamaki. 2019. Bridging the latency gap between NVM and DRAM for latency-bound operations. In *Proceedings of the 15th International Workshop on Data Management on New Hardware*. ACM.

[25] Simone Raoux, Feng Xiong, Matthias Wuttig, and Eric Pop. 2014. Phase change materials and phase change memory. *MRS Bulletin* 39, 8 (2014), 703âĂŞ710. https://doi.org/10.1557/mrs.2014.139

[26] Brian M Rogers, Anil Krishna, Gordon B Bell, Ken Vu, Xiaowei Jiang, and Yan Solihin. 2009. Scaling the bandwidth wall: challenges in and avenues for CMP scaling. *ACM SIGARCH Computer Architecture News* 37, 3 (2009), 371–382.

[27] Thomas Shull, Jian Huang, and Josep Torrellas. 2019. Designing a User-Friendly Java NVM Framework. (2019).

[28] SICM 2018. *Proceedings of the Workshop on Memory Centric High Performance Computing, MCHPC@SC 2018, Dallas, TX, USA, November 11, 2018*. ACM. http://dl.acm.org/citation.cfm?id=3286475

[29] SNAP [n. d.]. SNAP: SN (Discrete Ordinates) Application Proxy. https://github.com/lanl/SNAP

[30] Titan (2019). TITAN. https://www.olcf.ornl.gov/olcf-resources/compute-systems/titan/

[31] TOP500 List - November 2018 (2018). TOP500 List - November 2018. https://www.top500.org/list/2018/11/

[32] Jan Treibig, Georg Hager, and Gerhard Wellein. 2010. Likwid: A lightweight performance-oriented tool suite for x86 multicore environments. In *2010 39th International Conference on Parallel Processing Workshops*. IEEE, 207–216.

[33] Alexander van Renen, Lukas Vogel, Viktor Leis, Thomas Neumann, and Alfons Kemper. 2019. Persistent Memory I/O Primitives. *arXiv preprint arXiv:1904.01614* (2019).

[34] Jeffrey S Vetter and Sparsh Mittal. 2015. Opportunities for nonvolatile memory systems in extreme-scale high-performance computing. *Computing in Science & Engineering* 17, 2 (2015), 73–82.

[35] Haris Volos, Andres Jaan Tack, and Michael M Swift. 2011. Mnemosyne: Lightweight persistent memory. In *ACM SIGARCH Computer Architecture News*, Vol. 39. ACM, 91–104.

[36] Chao Wang, Sudharshan S Vazhkudai, Xiaosong Ma, Fei Meng, Youngjae Kim, and Christian Engelmann. 2012. NVMalloc: Exposing an aggregate SSD store as a memory partition in extreme-scale machines. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium*. IEEE, 957–968.

[37] Kai Wu, Frank Ober, Shari Hamlin, and Dong Li. 2017. Early Evaluation of Intel Optane Non-Volatile Memory with HPC I/O Workloads. *CoRR* abs/1708.02199 (2017). arXiv:1708.02199 http://arxiv.org/abs/1708.02199

[38] Jun Yang, Qingsong Wei, Cheng Chen, Chundong Wang, Khai Leong Yong, and Bingsheng He. 2015. NV-Tree: reducing consistency cost for NVM-based single level systems. In *13th {USENIX} Conference on File and Storage Technologies ({FAST} 15)*. 167–181.

[39] Ulrike Meier Yang et al. 2002. BoomerAMG: a parallel algebraic multigrid solver and preconditioner. *Applied Numerical Mathematics* 41, 1 (2002), 155–177.