Initial Experience with 3D XPoint Main Memory

Jihang Liu
State Key Laboratory of Computer Architecture
Institute of Computing Technology, CAS
University of Chinese Academy of Sciences
liujihang@ict.ac.cn

Shimin Chen*
State Key Laboratory of Computer Architecture
Institute of Computing Technology, CAS
University of Chinese Academy of Sciences
chensm@ict.ac.cn

***Abstract*—3D XPoint will likely be the first commercially available main memory NVM solution targeting mainstream computer systems. Previous database studies on NVM memory evaluate their proposed techniques mainly on simulated or emulated NVM hardware. In this paper, we report our initial experience experimenting with the real 3D XPoint main memory hardware.**

I. Introduction

A new generation of Non-Volatile Memory (NVM) technologies, including PCM [1], STT-RAM [2], and Memristor [3], are expected to address the DRAM scaling problem and become the next generation main memory technology in future computer systems [4]–[7]. NVM has become a hot topic in database research in recent years. However, previous studies evaluate their proposed techniques mainly on simulated or emulated hardware. In this paper, we report our initial experience with real NVM main memory hardware.

3D XPoint is an NVM technology jointly developed by Intel and Micron [8]. Intel first announced the technology in 2015. 3D XPoint based Optane SSD products are already available on the market. Intel expects to ship 3D XPoint based persistent memory DIMM products (a.k.a. Apache Pass, or AEP) in 2019. As a result, 3D XPoint will likely be the first commercially available main memory NVM solution targeting mainstream computer systems.

A number of companies have obtained AEP sample machines from Intel for testing purposes. We get access to an AEP machine through a collaboration project with Alibaba. We are interested in understanding the system architecture of the AEP machine, and the performance characteristics of the 3D XPoint main memory.

The remainder of the paper is organized as follows. Section II describes the interesting features of the AEP machine. Section III discusses our experimental methodology. Section IV reports the experimental results. Finally, Section V concludes the paper.

II. AEP Machine

The system architecture of the AEP machine is shown in Figure 1. We describe its interesting features that are different from those in existing x86-64 servers.

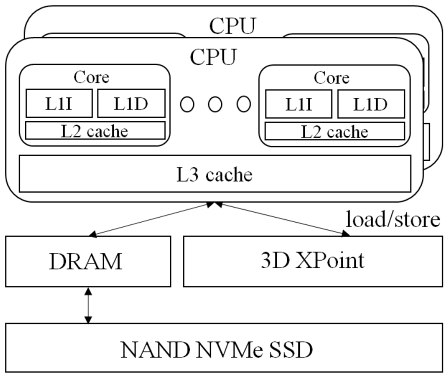

Fig. 1. AEP machine.

First, there are two CPUs in the machine. From the `/proc/cpuinfo`, the CPU model is listed as "Genuine Intel(R) CPU 0000%@". Our guess is that the CPU is a test model and/or not yet recognized by the Linux kernel. The CPU supports the new `clwb` instruction [9]. Given a memory address, `clwb` writes back to memory the associated cache line if it is modified. `clwb` and `sfence` can be combined to persist modified data to the NVM main memory. Compared to the existing `clflush` instruction, `clwb` is expected to be more efficient because it does not invalidate the associated line from the CPU cache.

Second, the AEP machine is equipped with two 3D XPoint main memory modules. Each module is 256GB large, and is connected to a CPU. For a CPU, one 3D XPoint module is local, the other module is remote. In other words, there are NUMA effects for NVM memory accesses.

Third, the machine is running a 4.9.75 version Linux kernel. Under the `/dev` directory, there are two special devices that represent the two 3D XPoint modules. They are named as `/dev/pmemX`, where X is a number.

Finally, there are two ways to exploit the 3D XPoint modules. File systems are installed on the devices using the fsdax mode. fsdax supports direct accesses to the NVM device without going through the OS page cache. More interestingly, we can use PMDK (Persistent Memory Development Kit)[1] to mmap an NVM file into the virtual address space of the running process, and then access the NVM memory using normal load and store instructions, essentially as main memory. We focus on the latter approach in this paper.

*Corresponding Author

[1]http://pmem.io/pmdk/libpmem/

III. Experimental Methodology

In the literature, NVM is expected to have read/write performance similar to but slower than DRAM. The performance of reads and writes may be asymmetric. Persisting modified contents to NVM is considered an expensive operation. Are these true with 3D XPoint?

We would like to understand the performance characteristics of 3D XPoint main memory through systematic performance tests. We consider the following aspects in our experiments:

- **Impact of write contents on performance**: Various techniques, including data comparison writes [10], have been proposed to improve NVM write performance and reduce write energy consumption in the literature. These techniques exploit the fact that a portion of the bits written are unchanged compared to the original contents. Therefore, it is possible to compare the new contents with the original contents and perform the NVM write operations only for the changed bits, thereby avoiding unnecessary overwrites. We would like to check if such effects exist in the real 3D XPoint memory hardware.
- **Total amount of data accessed**: SSDs and HDDs often employ caching in the devices to improve performance. If the working data set fits into the in-device caches, applications will see much better performance. In addition to caching, SSDs often reserve additional flash space (e.g. as much as 20% more than the reported capacity) in order to better organize new writes for wear-leveling purposes. While NVM is significantly different from SSDs and HDDs, techniques in the similar vein may be exploited. Therefore, it is necessary to observe the performance of 3D XPoint under different amount of data accessed.
- **The randomness of the memory accesses**: Database operations display different memory access patterns. For example, table scans perform sequential accesses. Hash or B+-Tree index accesses often visit memory locations in a random fashion. Random accesses can be either dependent or independent. A typical scenario that performs dependent random accesses is following a linked list. The memory address of the next node on the linked list is stored in the previous node. Therefore, the access to the next node depends on the access to the previous node. In contrast, retrieving a set of records given their record IDs result in independent random accesses, which may be served by parallel memory accesses. It is important to study different memory access patterns.
- **Whether to persist data using clwb+sfence**: A store instruction (e.g., `mov`) completes when it reaches the CPU cache. After that, when and in what order modified lines are written back to memory are controlled by the CPU cache hardware. Unfortunately, CPU cache is volatile. Therefore, to guarantee consistent NVM data structures after crash recovery, special instructions (e.g., `clwb` and `sfence`) have to be used to ensure modified contents are written back to the NVM memory. We would like to quantify the overhead of such special instructions.
- **NUMA effect**: In the AEP machine, two 3D XPoint modules are attached to two CPUs, respectively. Therefore, we study 3D XPoint's NUMA effect in Section IV-D. We control the CPU affinity and memory binding for the tests. (Please note that in the experiments of Section IV-A–IV-C, we avoid the NUMA effect by performing only local memory and NVM accesses.)

We run a set of memory tests. Every test run allocates $BlockNum \times BlockSize$ memory either from DRAM or from 3D XPoint (using PMDK). Then, it initializes the allocated memory by writing every bytes. After that, it tests a specified memory access pattern. `gettimeofday` is used to measure the elapsed time of performing all the memory accesses. Finally, it computes the access latency and bandwidth by combining the elapsed time, $BlockNum$, and $BlockSize$. We test the following memory access patterns:

- **Sequential read (seqread)**: The test sequentially reads the $BlockNum \times BlockSize$ memory.
- **Random read (randread)**: The memory is divided into $BlockNum$ blocks, each with $BlockSize$ bytes. The test allocates a pointer array with $BlockNum$ entries. It populates the array with the memory addresses of the blocks, then randomly shuffles the array entries. It reads the blocks by following the pointers in the array, thereby performing independent random accesses. We vary $BlockSize$ in the experiments. As $BlockSize$ increases, the memory access patterns become closer to the sequential access pattern.
- **Dependent read (depread)**: The test obtains a random permutation of the blocks using the random pointer array as in randread. It constructs a linked list of the blocks in the order of the random permutation. Then, the test reads the memory blocks by following the linked list. In this way, it performs dependent memory reads.
- **Sequential / random / dependent write (seqwrite / randwrite / depwrite)**: The write tests use similar patterns as the read tests. There is one major difference. A write test can be specified to persist modified data. If this feature is enabled, then a `clwb` is issued after the modified contents of a cache line are fully written, and a `sfence` is issued after an entire block is written. If this feature is disabled, then writes are performed without the special instructions.

IV. Experimental Results

A. Impact of Write Contents on Performance

The first set of experiments check whether 3D XPoint memory modules take advantage of the data patterns of the modified data and the original data during the write operations. If the write performance is dependent on the data patterns, then we need to pay special attention to choose appropriate initial and final values for the writes in all subsequent experiments. If write values do not affect the write performance, then we can use any arbitrary values.

We test three initial values in the experiments: `0xf0f0f0f0f0f0f0f0` (every byte consists of four "1"s

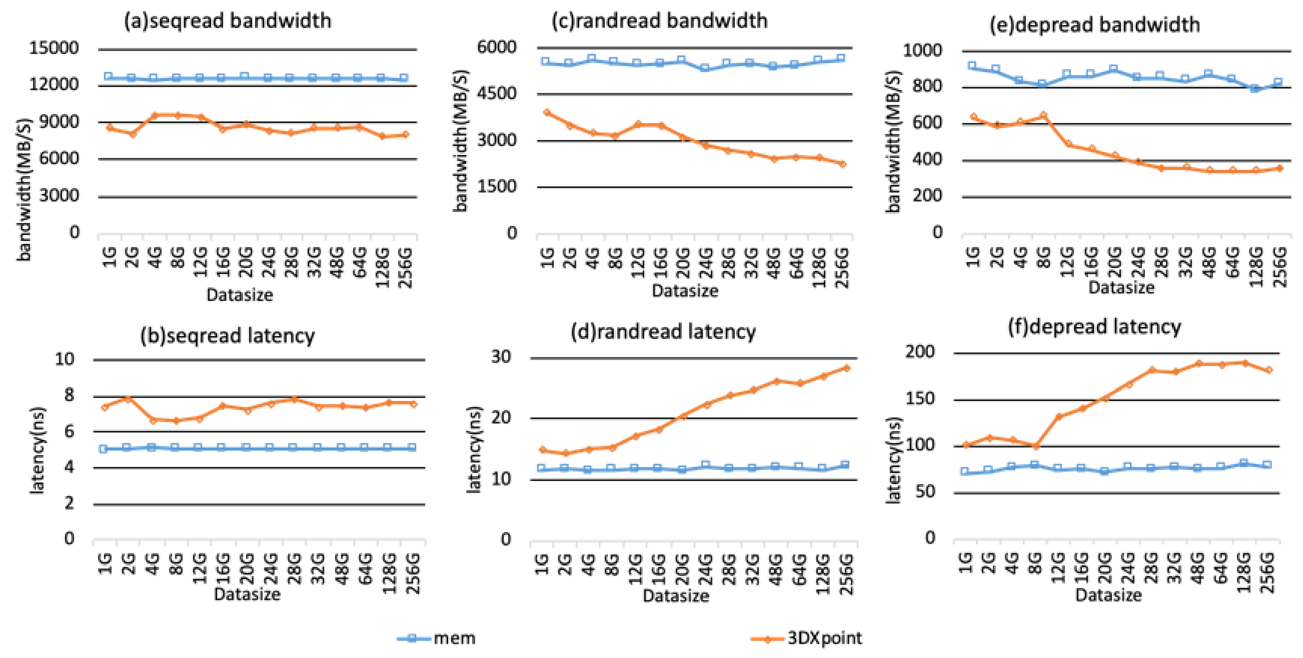

Fig. 3. Read performance varying the amount of data accessed.

followed by four "0"s), `0xaaaaaaaaaaaaaaaa` (alternating between "1"s and "0"s), and `0x0000000012345678` (an arbitrary value). For every initial value, we construct eight values to write by performing bitwise not to 1 byte, 2 bytes, ..., 8 bytes of the initial value. We choose the total amount of data accessed to be 8GB and 32GB. Then we run seqwrite and randwrite experiments.

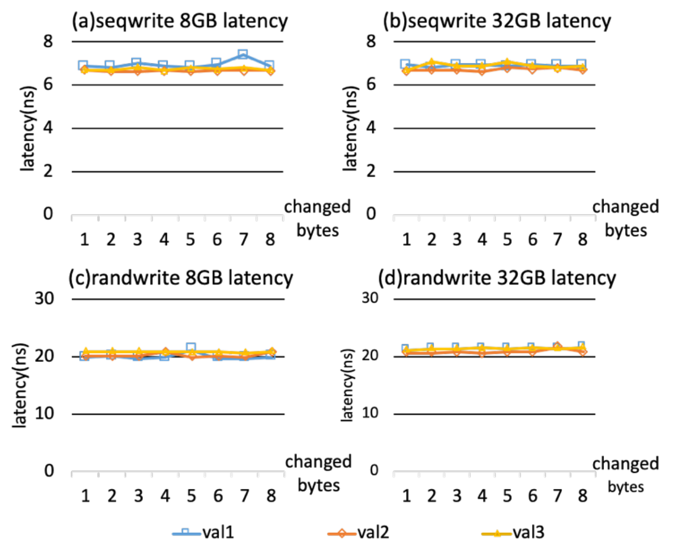

Fig. 2. Varying the initial values and the write values.

Figure 2(a)-(d) show the performance of sequential writes and random writes of 8GB and 32GB data while varying the number of changed bytes in every 8-byte word. The Y-axis shows the average latency for writing a 64B cache line. From the figures, it is clear that the write performance of 3D XPoint does not change significantly while varying write data patterns. Therefore, we conclude that the write performance of 3D XPoint in the sample AEP machine does not depend on the data patterns. In the following experiments, we can choose the initial values and write values arbitrarily.

B. *Varying Total Amount of Data Accessed*

In this set of experiments, we perform sequential, random, and dependent read and write tests while varying the total amount of data accessed. We avoid the NUMA effect by testing only the local DRAM module and the local NVM module. We vary the total amount of data accessed from 1GB to 256GB. The smallest size, 1GB, is well beyond the CPU cache size. The largest size, 256GB, is the capacity of a single 3D XPoint module in this machine. The block size in the experiments is equal to the cache line size (i.e. 64B).

Figure 3 (a)-(f) show the bandwidth and latency for seqread, randread, and depread on DRAM (mem) and 3D XPoint memory. Figure 4 (a)-(f) show the bandwidth and latency for seqwrite, randwrite, and depwrite on DRAM (mem), 3D XPoint memory, and 3D XPoint with `clwb+sfence` instructions (persist). Please note that the Y-axes in the write performance figures (Figure 4 (a)-(f)) are in the logarithmic scale. From the figures, we have the following observations.

Performance varying data size. As the total amount of data accessed increases from 1GB to 256GB, the DRAM curves remain relatively flat in all figures, while the random and dependent read and write curves of 3D XPoint change significantly. We compute the slope of changes by dividing the delta latency by the delta data sizes of adjacent points. We see that the slopes become relatively small after 32GB for all cases. Therefore, we suspect that techniques such as caching are used inside the 3D XPoint module to improve performance. This effect becomes insignificant when the amount of data accessed is 32GB or larger.

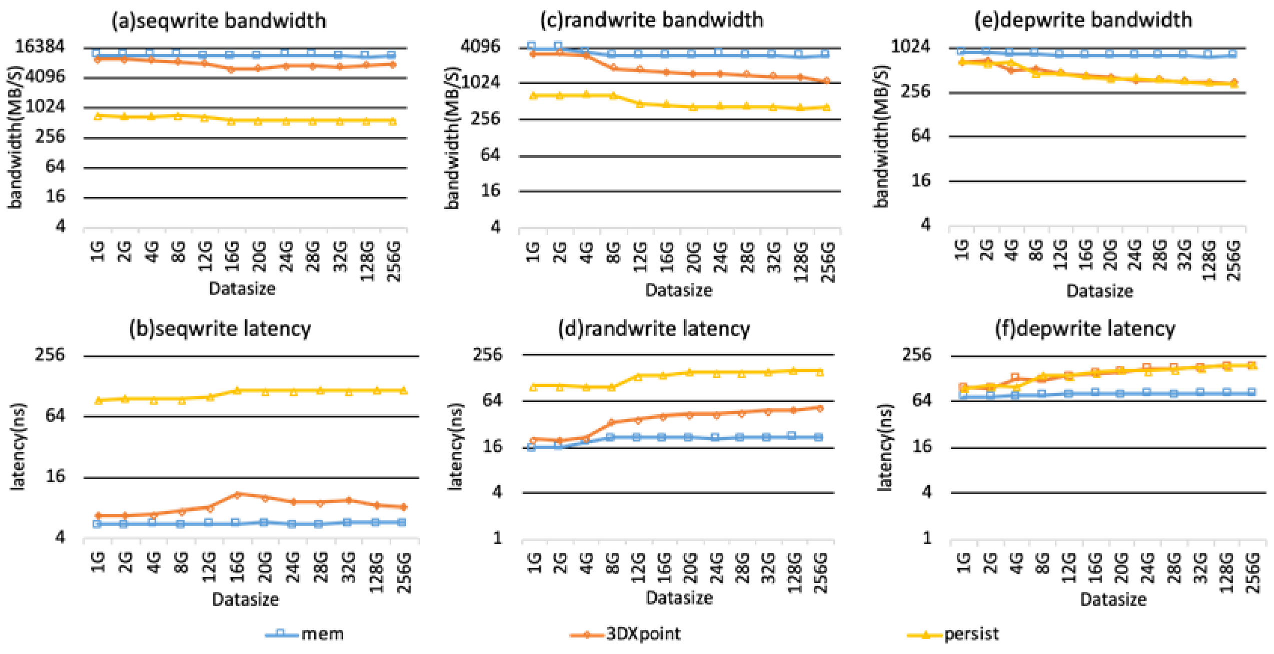

Fig. 4. Write performance varying the amount of data accessed.

Read performance: 3D XPoint vs. DRAM. Compared to DRAM, 3D XPoint is 1.3–1.6x slower for seqread, 1.2–2.3x slower for randread, and 1.3–2.5x slower for depread. If we consider only data sizes greater than or equal to 32GB, then compared to DRAM, 3D XPoint is 1.5x slower for seqread, 2.1–2.3x slower for randread, and 2.3–2.5x slower for depread.

depread ensures that only a single read can be serviced at a time. On the other hand, in randread and seqread, the CPU can issue multiple reads in parallel. The performance of randread and seqread also reflects the device's capability of supporting concurrent memory accesses. Therefore, if we consider the latency of a memory read, 3D XPoint is 2.3–2.5x slower than DRAM.

Write performance: 3D XPoint vs. DRAM. Compared to DRAM, 3D XPoint is 1.2–1.9x slower for seqwrite, 1.1–2.5x slower for randwrite, and 1.3–2.3x slower for depwrite. If we consider only data sizes greater than or equal to 32GB, then compared to DRAM, 3D XPoint is 1.4–1.7x slower for seqwrite, 2.2–2.5x slower for randwrite, and 2.2–2.3x slower for depwrite.

Write performance: persist vs. normal 3D XPoint write. From Figure 4, we see that persist is drastically worse than normal 3D XPoint writes for seqwrite and randwrite. This is because normal 3D XPoint writes are cached in the CPU cache. The actual writes to the 3D XPoint can be reordered and performed in parallel in the background. In contrast, persist uses `clwb` to enforce the writes to immediately go to the 3D XPoint memory. `sfence` further ensures that the next block of data is written after the previous writes complete. Since the block size is 64B in this set of experiments, there will be no concurrent cache line writes in the persist experiments.

As a result, compared to normal 3D XPoint writes, persist is 10.7–14.2x slower for seqwrite, and 2.9–5.1x slower for randwrite. If we consider only data sizes greater than or equal to 32GB, then compared to normal 3D XPoint writes, persist is 12.1–14.1x slower for seqwrite, and 3.0–3.4x slower for randwrite. Interestingly, the largest slowdowns occur when the data sizes are small, where the caching mechanism works best.

For depwrite, the persist curve and the 3D XPoint curve almost overlap with each other. For depwrite, there is no temporal or spatial locality among the writes and there is no concurrent writes in the memory system. Therefore, the CPU cache has limited benefits to depwrite. Consequently, depwrite and persist have similar performance.

3D XPoint: Read vs. Write. Compared to reads, 3D XPoint writes are up to 1.4x slower for sequential accesses, up to 2.3x slower for random accesses, and up to 1.2x slower for dependent accesses. Overall, the asymmetric effect of NVM reads and writes are modest: 3D XPoint writes are up to 2.3x than reads.

C. Varying Block Size

In the previous experiments, the block size is fixed to 64B. In this set of experiments, we vary the block size from 16B to 4096B and run sequential, random, and dependent memory tests. For the persist write experiments, we issue a `clwb` for every line in the block, and issue a single `sfence` instruction after the entire block is written.

Figure 5(a)-(f) show seqread, randread, depread, seqwrite, randwrite, and depwrite performance while varying the block size from 16B to 4096B. The Y-axes of Figure 5(a) and (b) are in the linear scale, while the Y-axes of Figure 5 (c)-(f) are in the logarithmic scale. From the figures, we have the following observations.

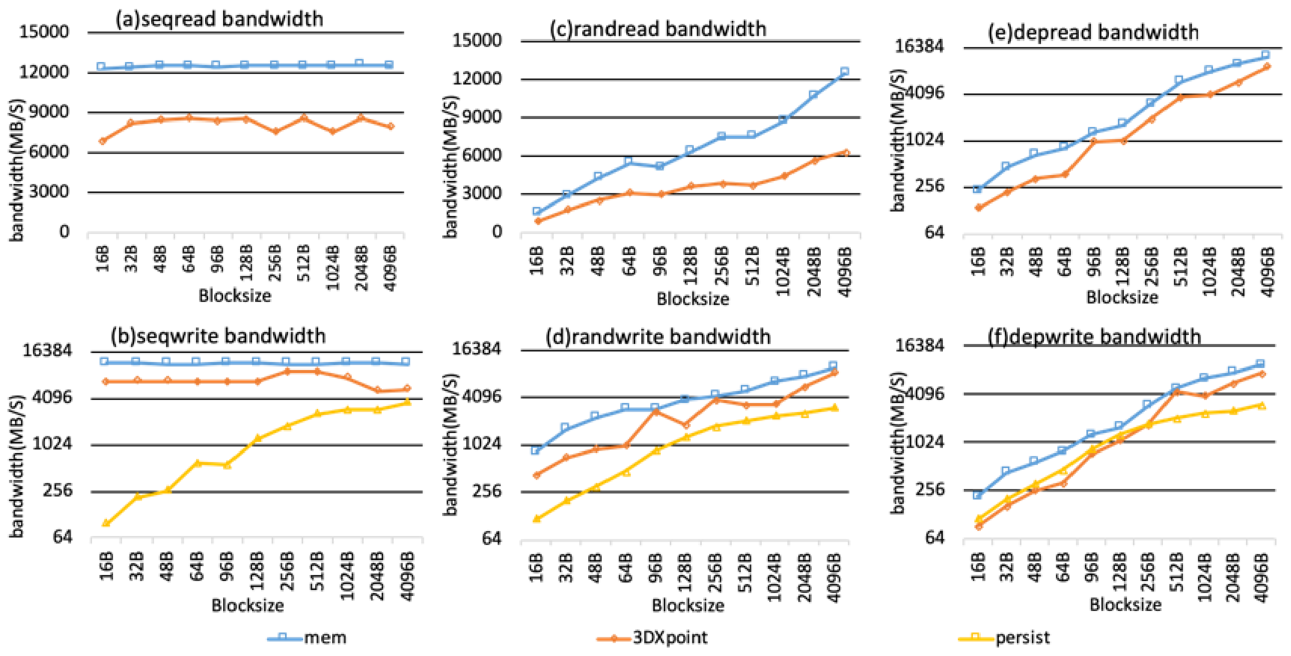

Fig. 5. Read and write performance varying block size.

First, in the cases of seqread and seqwrite, the performance of DRAM and 3D XPoint is roughly stable. In the test program, the inner loop processes the data in a block, and the outer loop processes every block. The test program performs sequential memory accesses for all the data accessed. The block size in the inner loop does not make much difference.

Second, in the cases of randread and randwrite, the bandwidth of DRAM and 3D XPoint increases as the block size grows larger. This is because the larger the block size, the less random and the more sequential the accesses. Therefore, the randread (randwrite) performance become closer and closer to the seqread (seqwrite) performance.

Finally, the performance of persist writes also increases as the block size increases. However, interestingly, in the case of depwrite, the curve of persist and the curve of normal 3D XPoint writes diverge when the block size is beyond 512B. As the access pattern becomes more and more similar to the sequential accesses, the cost of `clwb` shows up.

D. NUMA Effect

In this set of experiments, we study the NUMA effect of the AEP machine. The tests use the `numactl`, `cpubind`, and `membind` commands to bind the desired CPU and memory nodes to the test program. Both DRAM and 3D XPoint have NUMA effect. That is, remote memory accesses have longer latency and lower bandwidth than local memory accesses. What's interesting is the impact of 3D XPoint vs. DRAM on the NUMA effect.

Table I compares the performance of remote and local memory accesses for DRAM, 3D XPoint, and persist. The table lists the latency differences (remote latency - local latency) and the latency ratios (remote latency / local latency) for seqread, randread, depread, seqwrite, randwrite, and depwrite. From the table, we have the following observations:

1) As expected, the performance of remote memory access is lower than the performance of local memory accesses for both DRAM and 3D XPoint. The two CPUs in the system are connected through the QPI link. A remote memory access has to go through the QPI link between the two CPUs and then use the memory controller on the remote CPU to perform the memory access. This extends the memory access latency and reduces the memory bandwidth.
2) For seqread, seqwrite, randread and randwrite, the latency difference of 3D XPoint is larger than that of DRAM. However, the latency ratios are all similar. In these cases, the CPU can issue multiple concurrent memory accesses. The bandwidth supported by the QPI are the determinant factor. This is a feature of the QPI. Therefore, the latency ratios are similar for both DRAM and 3D XPoint.
3) For depread, the latency differences of DRAM and 3D XPoint are similar, but the latency ratio of DRAM is larger than that of 3D XPoint. In this case, there are no concurrent memory accesses. The dominant factor is the extra latency caused by the QPI. Therefore, the latency differences of DRAM and 3D XPoint are similar. Latency ratio = (local latency + difference) / local latency = 1 + difference/local latency. Since the local latency of 3D XPoint is larger than that of DRAM, its latency ratio is smaller.
4) For 3D XPoint persist writes, the latency differences and the latency ratios are all larger than those of DRAM and 3D XPoint normal writes. Remote persist is much more costly than local persist. `clwb`+`sfence` have greater impact on remote accesses than on local accesses. Our

TABLE I
REMOTE VS. LOCAL DRAM AND 3D XPOINT PERFORMANCE.

	remote/local DRAM		remote/local 3D Xpoint		remote/local persist	
	latency difference	latency ratio	latency difference	latency ratio	latency difference	latency ratio
seqread	2.4ns	1.5	3.4ns	1.5	–	–
randread	6.2ns	1.5	13.4ns	1.5	–	–
depread	58.3ns	1.8	47.3ns	1.3	–	–
seqwrite	2.7ns	1.5	4.2ns	1.5	112ns	2.1
randwrite	10.4ns	1.5	22.0ns	1.5	186ns	2.2
depwrite	56.3ns	1.7	145ns	1.8	147ns	1.8

guess is that in the remote case, the special instructions need the two CPUs to cooperate, perform certain extra operations, which lead to the additional overhead.

V. CONCLUSION

In conclusion, we summarize our main observations in the experiments.

- The memory access performance of 3D XPoint is modestly lower than that of DRAM. Compared to DRAM, 3D XPoint's read performance can be up to 2.5x slower, and its write performance can be up to 2.3x slower.
- The write performance of 3D XPoint is quite similar to its read performance. Writes are at most 2.3x slower than reads. The asymmetric effect of NVM reads and writes are modest.
- Persisting writes to 3D XPoint incur drastic cost for sequential and random writes. The slowdowns can be at least 10.7x and 2.9x for sequential writes and random writes, respectively.
- The performance of 3D XPoint degrades as an application accesses more data. We suspect that techniques such as caching are exploited in the 3D Xpoint modules to improve performance. This effect is less significant when the size of data visited is 32GB or larger for the 256GB 3D XPoint module.
- There is 3D XPoint NUMA effect in the AEP system. Remote 3D XPoint accesses have significantly longer latency and lower bandwidth than local 3D XPoint accesses. We would like to point out that the performance of persisting writes to 3D XPoint changes the most drastically among the various access patterns tested.

In the literature, previous studies often assume that NVM writes perform much worse than DRAM writes, and also much worse than NVM reads. This is not the case for the 3D XPoint main memory tested in our experiments. On the other hand, previous studies assume that the special instructions to persist NVM writes incur significant overhead. This is confirmed by the 3D XPoint experiments. The overhead caused by `clwb+sfence` can be quite drastic for local writes, and even more drastic for remote writes. Given these observations, it would be interesting to re-evaluate the techniques for reducing NVM writes in the literature.

Given the impact of data size on performance, we need to carefully set up the experiments for NVM database studies on 3D XPoint main memory. We expect that database systems will typically utilize large fractions of the 3D XPoint memory space. Therefore, the data size in the experiments should be at least (32GB/256GB) 12.5% of the 3D XPoint memory capacity in order to model the typical database use scenarios.

Finally, we would like to note that we tested an AEP *sample* machine. Since this is not yet a product on the market, there could potentially be modifications to its designs. The mature product may have slightly different performance characteristics. However, we believe our initial experience with 3D XPoint main memory is still interesting. It sheds light on the first NVM main memory technology expected to be commercially available in the near future.

VI. ACKNOWLEDGMENTS

This work is partially supported by NSFC project No. 61572468, by Alibaba collaboration project No. XT622018000648, and by K.C.Wong Education Foundation.

REFERENCES

[1] S. Raoux, G. W. Burr, M. J. Breitwisch, C. T. Rettner, Y. Chen, R. M. Shelby, M. Salinga, D. Krebs, S. Chen, H. Lung, and C. H. Lam, "Phase-change random access memory: A scalable technology," *IBM Journal of Research and Development*, vol. 52, no. 4-5, pp. 465–480, 2008.

[2] D. Apalkov, A. Khvalkovskiy, S. Watts, V. Nikitin, X. Tang, D. Lottis, K. Moon, X. Luo, E. Chen, A. Ong, A. Driskill-Smith, and M. Krounbi, "Spin-transfer torque magnetic random access memory (STT-MRAM)," *JETC*, vol. 9, no. 2, pp. 13:1–13:35, 2013.

[3] J. J. Yang and R. S. Williams, "Memristive devices in computing system: Promises and challenges," *JETC*, vol. 9, no. 2, pp. 11:1–11:20, 2013.

[4] S. Chen, P. B. Gibbons, and S. Nath, "Rethinking database algorithms for phase change memory," in *CIDR 2011, Fifth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 9-12, 2011, Online Proceedings*, 2011, pp. 21–31.

[5] S. Chen and Q. Jin, "Persistent b+-trees in non-volatile main memory," *PVLDB*, vol. 8, no. 7, pp. 786–797, 2015.

[6] J. Arulraj, A. Pavlo, and S. Dulloor, "Let's talk about storage & recovery methods for non-volatile memory database systems," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, 2015, pp. 707–722.

[7] I. Oukid, J. Lasperas, A. Nica, T. Willhalm, and W. Lehner, "Fptree: A hybrid SCM-DRAM persistent and concurrent b-tree for storage class memory," in *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, 2016, pp. 371–386.

[8] D. H. Graham, "Intel optane technology products - what's available and what's coming soon," https://software.intel.com/en-us/articles/3d-xpoint-technology-products.

[9] Intel Corp., "Intel 64 and ia-32 architectures software developer's manual," Order Number: 325383-060US, 2016.

[10] B. Yang, J. Lee, J. Kim, J. Cho, S. Lee, and B. Yu, "A low power phase-change random access memory using a data-comparison write scheme," in *International Symposium on Circuits and Systems (ISCAS 2007), 27-20 May 2007, New Orleans, Louisiana, USA*, 2007, pp. 3014–3017.