Characterizing the Performance of Intel Optane Persistent Memory

-- A Close Look at its On-DIMM Buffering

Lingfeng Xiang,
Xingsheng Zhao, Jia Rao, Song Jiang, Hong Jiang

The University of Texas at Arlington

Code: https://github.com/lingfenghsiang/Persistent-Memory-Study

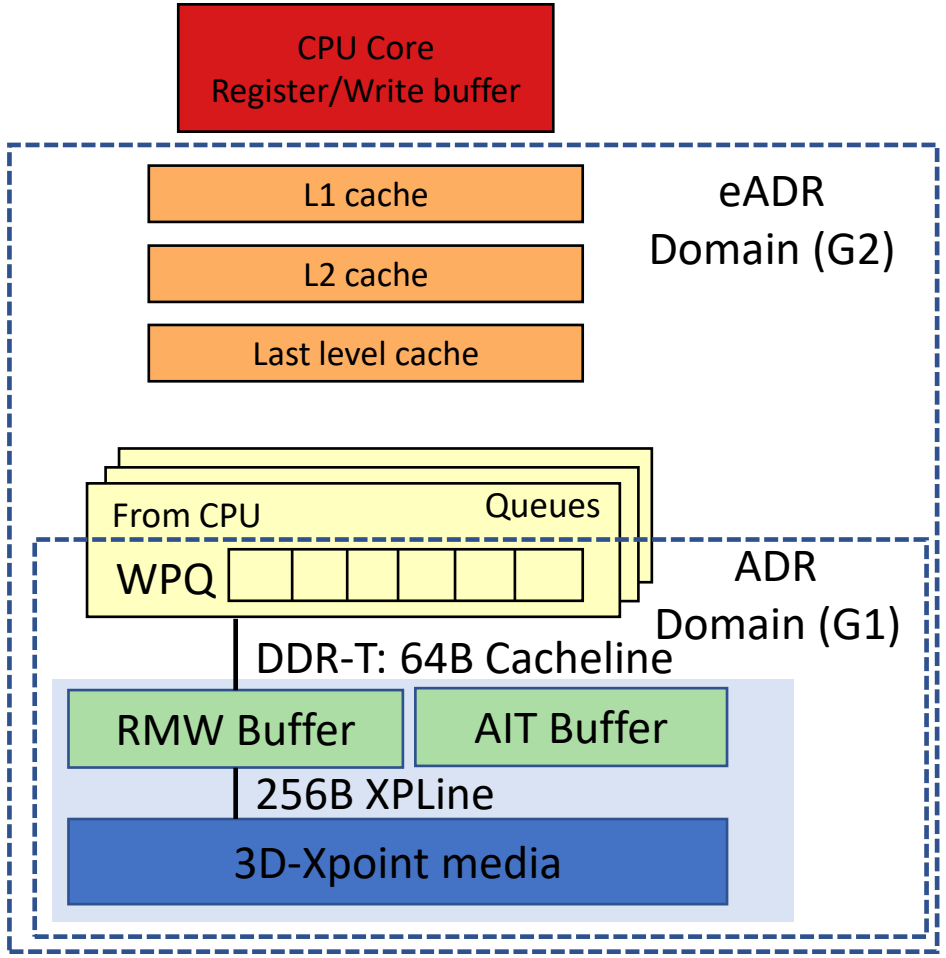
UNIVERSITY OF TEXAS ARLINGTON

Intel Optane Persistent Memory (DCPMM)

- The first commercially available NVM DIMM
- Features
  - Affordable large capacity, 3X cheaper than DRAM ($495 vs. $1,750 for 128 GB)
  - Support for persistence
  - Byte addressable, accessible via `load`/`store` instructions
  - On memory bus, directly connected to the iMC via the DDR-T protocol
  - 2-3X slower than DRAM

DCPMM Architecture

- Writes reaching the ADR domain are persistent
  - ADR includes the WPQ and on-DIMM buffers
  - eADR (G2 Optane) includes CPU caches
- On-DIMM buffers
  - Read-modify-write (RMW) buffer
    - Bridging the gap in access granularity (64B vs. 256B)
  - Address indirection table (AIT) buffer
    - Wear-leveling and bad block management



Persistent Programming

- Persistence
  - Writes reach the persistence (ADR) domain
  - `clflush`, `clwb`, non-temporal store
- Write ordering
  - Writes are persisted in program order
  - `mfence`, `sfence`
- Update atomicity
  - 8-byte write are guaranteed atomic by processor
  - Use undo/redo logging or shadowing to guarantee atomicity for larger writes

DCPMM vs. DRAM

- Similarities
  - Byte-addressable
  - Allowing CPU caching
  - Sharing the same memory consistency model
- Differences
  - Access granularity (256B vs. 64B)
  - DDR-T (asynchronous) vs. DDR4 (synchronous)
  - Proprietary yet complex on-DIMM read-write buffering

VS.

Known Characteristics

- Write amplification
  - Stores < 256B become RMW operations
  - The RMW buffer merges adjacent writes and is 16KB in size
- Asymmetric read-write performance
  - Read bandwidth is ~3x higher than write bandwidth
  - Read latency is ~2x higher than write latency
- Performance strongly dependent on access pattern
  - Sequential access is much faster than random access

[Yang, FAST'20], [Wang, MICRO'21], [Gugnani, VLDB'21]

Many Unknowns

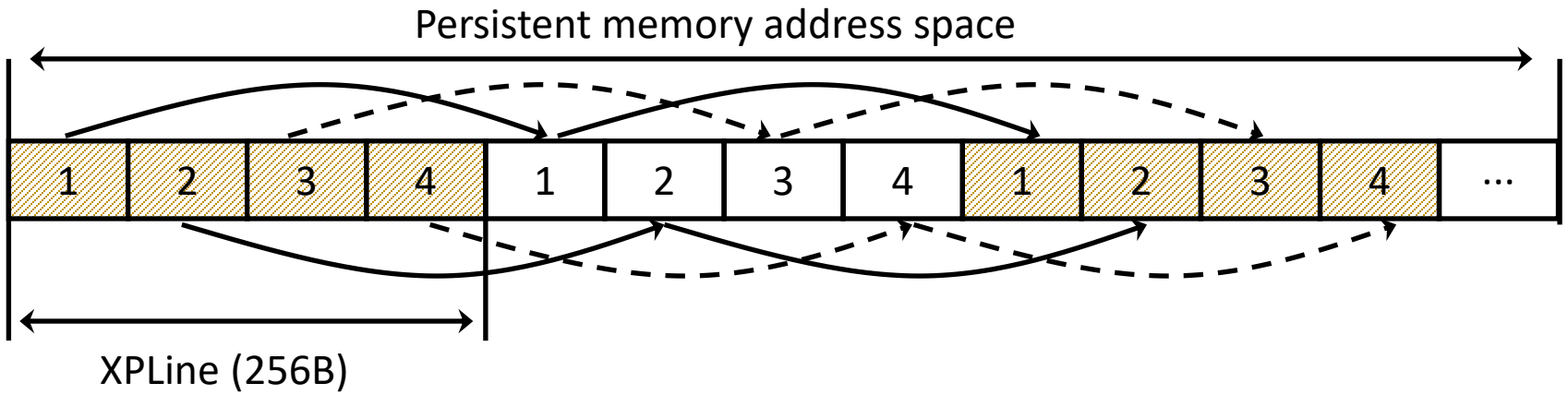
- Read buffering vs. writing buffering
  - Write buffering -- coalescing small writes
  - Read buffering – potential data reuse
- On-DIMM buffering vs. CPU caching
- Performance implications of persistence barriers
- The evolution from G1 to G2 Optane

Methodology

- Metrics
  - Write amplification (WA) = $\frac{data\ written\ to\ media}{data\ issued\ by\ iMC}$
  - Read amplification (RA) = $\frac{data\ read\ from\ media}{data\ demanded\ by\ iMC}$
- Platform
  - G1 Optane: Intel Xeon Gold 6320 + 6* 128GB DCPMM + 192GB DRAM
  - G2 Optane: Intel Xeon Gold 5317 + 6* 128GB DCPMM + 192GB DRAM
  - NVM mounted to program address space using `DAX` mode in `ext4`
  - Use `ipmwatch` to monitor data access to the media and iMC

Read Buffering

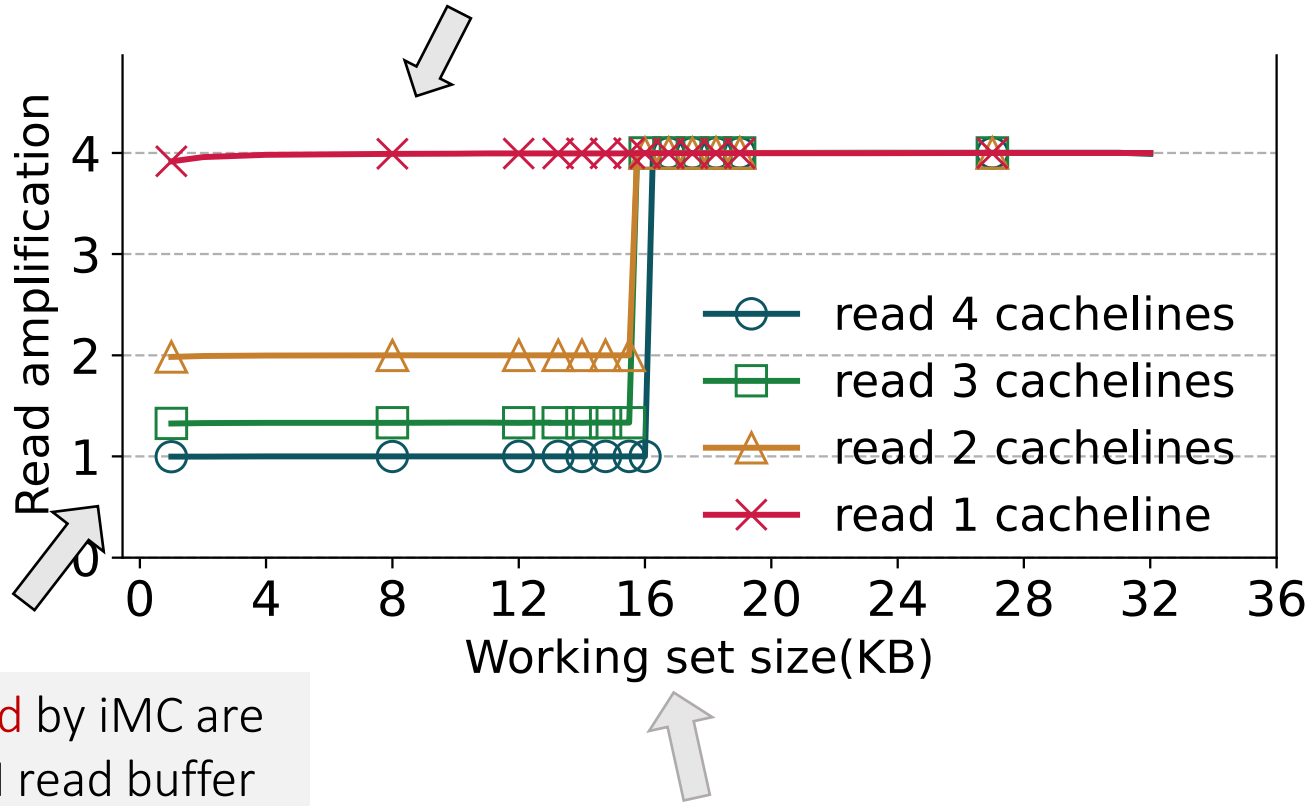
Monitor RA as the benchmark repeatedly access a cacheline and its sibling cachelines in the same XPLine to infer the size of the read buffer and its management scheme



- Read one cacheline each time with a stride of 256B
- Cachelines are immediately flushed from CPU caches after each access
- Two parameters: WSS and # of cachelines read per XPLine

Read Buffering – cont'd

1. Read buffer is exclusive to the CPU cache

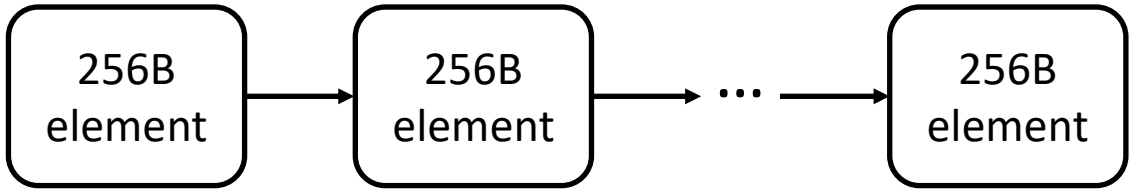


2. Cachelines not loaded by iMC are cached in the on-DIMM read buffer

3. Read buffer size is 16KB in G1 Optane

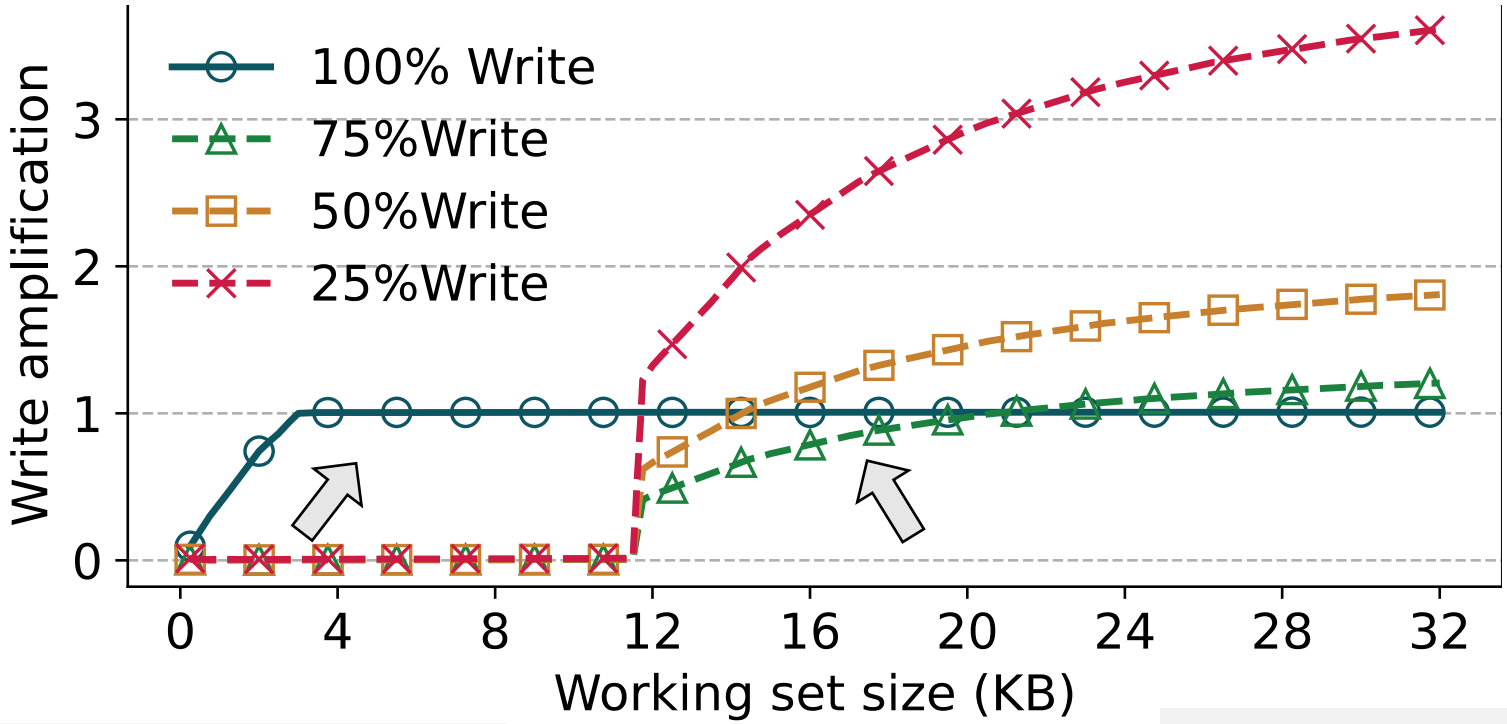
Write Buffering

Monitor WA to infer the size of the write buffer and its write-back policy



- Sequentially or randomly ordered elements
- Partial or full writes to each element (XPLine)
- Writes are non-temporal stores that bypass the CPU caches

Write Buffering – cont'd

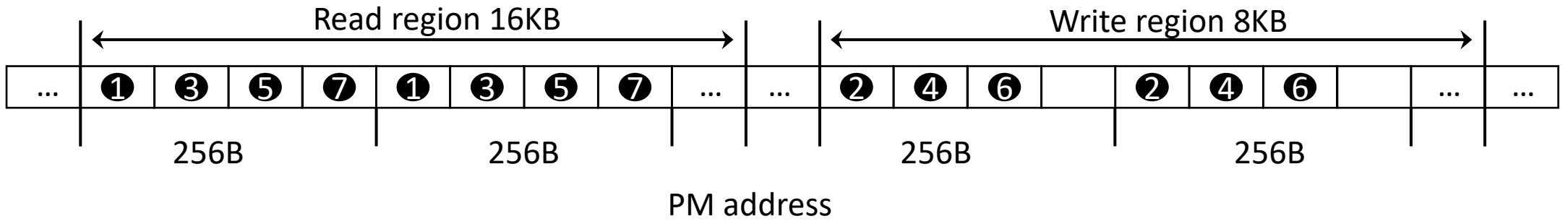


1. Full writes are periodically written back to media

2. Partial writes are retained in the write buffer until evicted

3. Periodic writeback is disabled in G2 Optane

The Relationship between Read and Write Buffers



- Interleaved reads and write to two non-overlapping regions
- Cachelines invalidated after read and using `nt-stores` for writes
- Two baseline programs only accessing the read and write regions, respectively

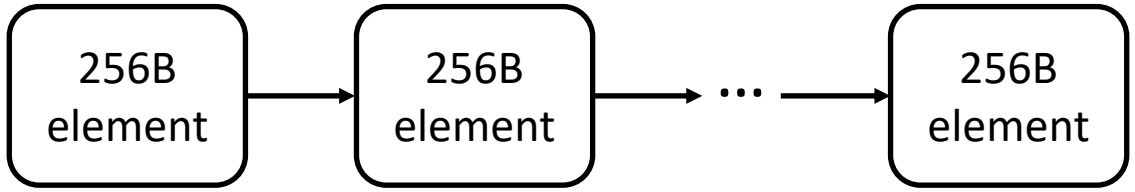
Compare RA and WA with that in the baseline programs to infer if the two buffers are a shared space or separate

The Relationship between Read and Write Buffers – cont'd

- Findings
  - The benchmark with interleaved read and write has the same RA and WA as the baseline programs do
  - The WSS of the read and write region fits in the read and write buffer, respectively, but their aggregate size overflows either buffer

- The read and write buffers are separate
- Reads can hit the write buffer and writes can directly update XPLines in the read buffer

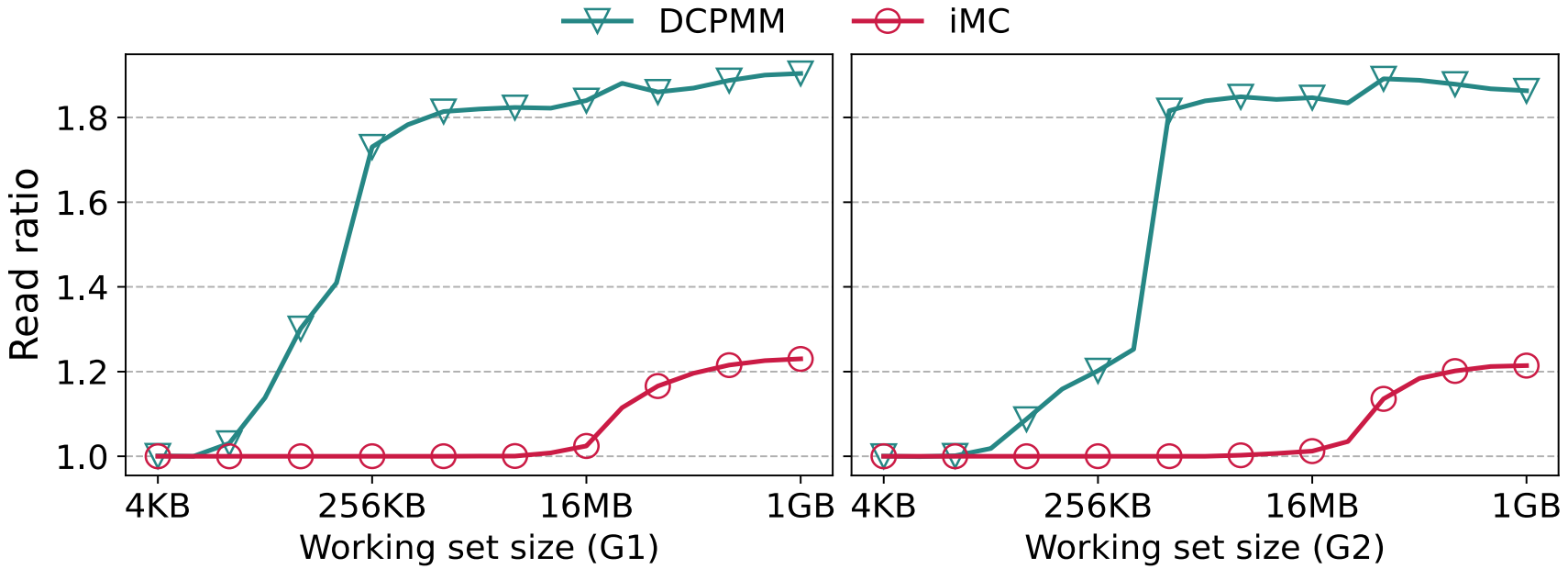
Prefetching



- Randomly ordered elements
- Sequentially read within each element (XPLine) to trigger prefetching

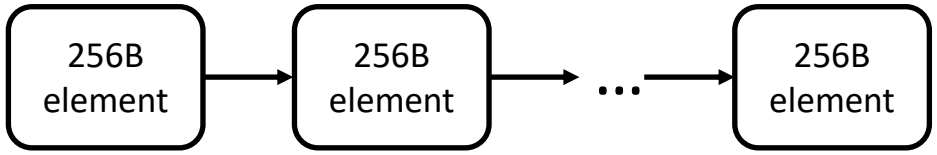
Measure PM and iMC read ratio to infer CPU cache and DCPMM prefetching activities

Prefetching – cont'd



- Most data prefetched to on-DIMM buffers are due to CPU prefetching
- Cost of misprefetching is much higher in DCPMM due to the mismatch in access granularity

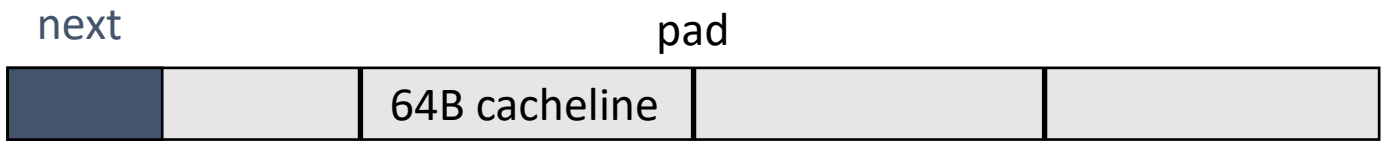
CPU Caching vs. On-DIMM Buffering



Sequentially or randomly linked list

```
typedef struct working_set_unit
{
        struct working_set_unit *next;
        uint64_t pad[NPAD ];
} working_set_unit_t;
```

XPLine-size element



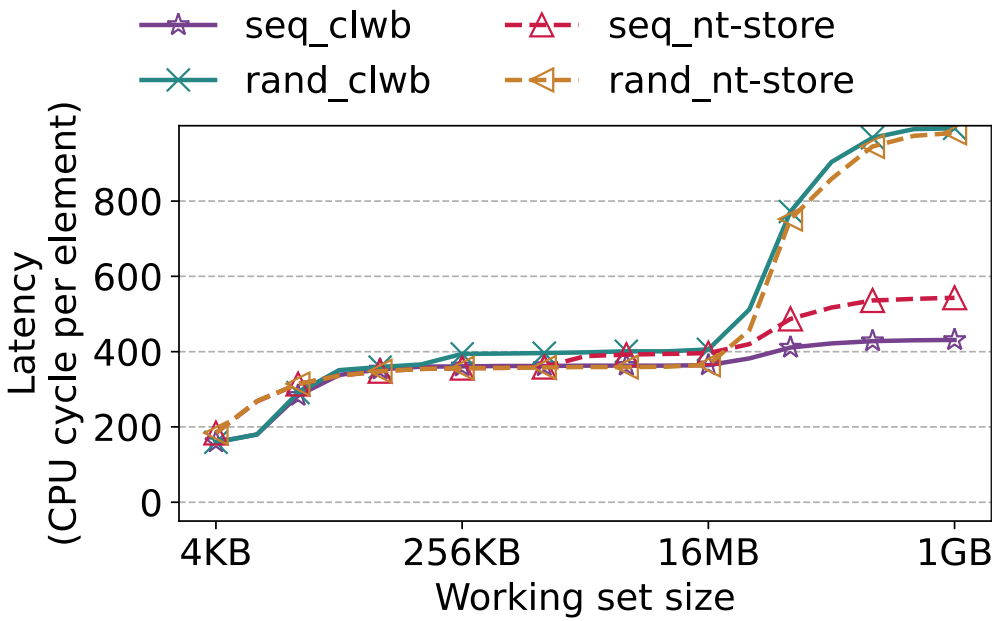
Read (pointer chasing)

Write (`clwb` or `nt-store`)

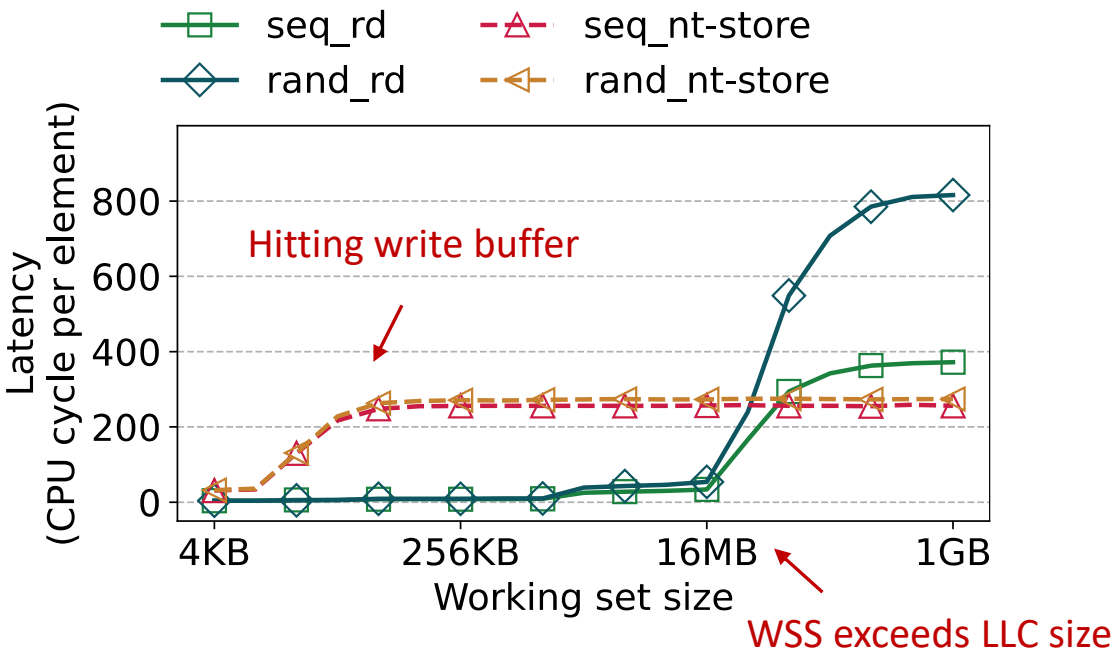
To decouple read and write, store element addresses in an array in DRAM

- Write-dominant workload
- Dereferencing is the only read to an element
- Read and write occur on different cachelines in an XPLine

Breaking Down Latency



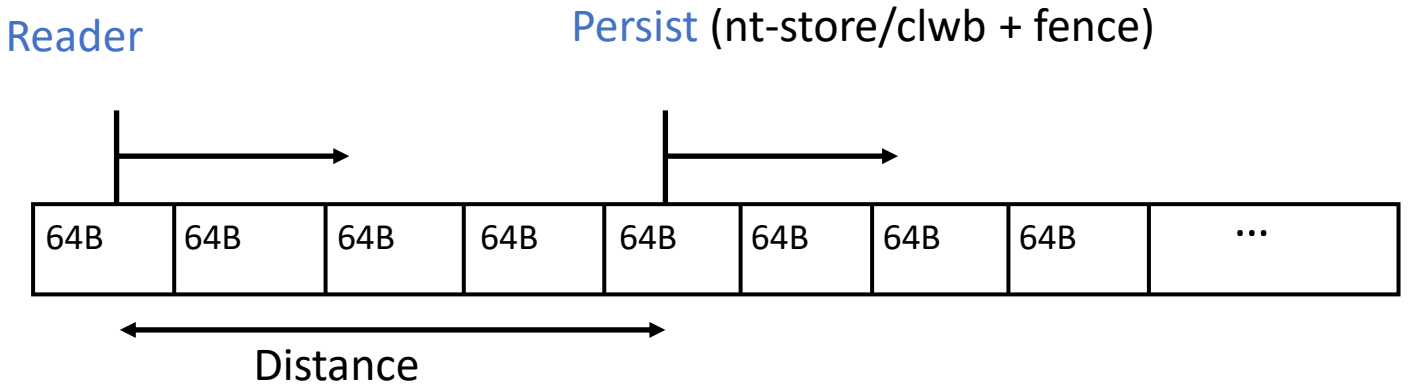
List traversal via pointer chasing



Decouple the latency of pointer chasing (read) and the actual write

Random read could dominate performance !

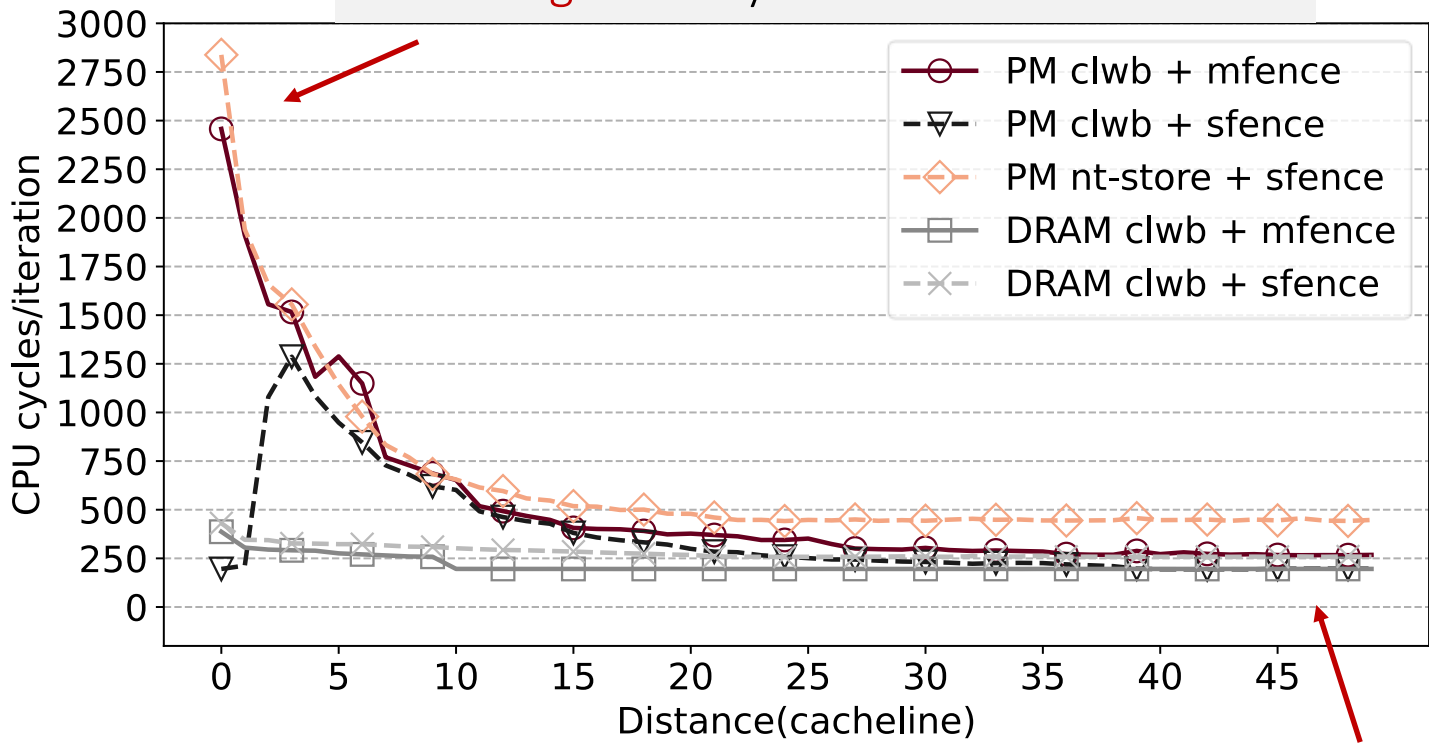
Read After Persist (RAP)



- Measure the latency of read to a recently persisted address
- Control the distance between the reader and persist

RAP Latency

1. Reading a recently persisted cacheline suffers high latency



Memory fences only ensure persists are globally visible (i.e., reaching WPQ) but not necessarily completed

2. Read latency approaches to on-DIMM buffer latency if sufficiently distant from persist

Takeaways

- Separate on-DIMM read and write buffers with distinct purposes
  - Read buffer: aids prefetching and improves sequential performance
  - Write buffer: hides media latency

  Decouple read and write in performance analysis and optimizations
- Various implications due to the mismatched cacheline and media access granularity and the asynchronous DDR-T protocol
  - Write amplification
  - Increased misprefetching penalty
  - Long RAP latency

  Rethinking the use of XPLine-aligned data blocks

Case Studies

- **Cacheline-Conscious Extendible Hashing (CCEH) [Nam, FAST'19]**
  - **Problem:** 3 random reads (pointer dereference) to locate a hash table bucket before an update, a performance bottleneck
  - **Optimization:** A helper thread to speculatively prefetch pointer addresses
- **FAST & FAIR B+-Tree [Hwang, FAST'18]**
  - **Problem:** Repeated read and write to the same cacheline during in-place key update in internal nodes, causing long RAP latency
  - **Optimization:** Redo logging to turn in-place updates to out-of-place updates
- **XPLine-aligned workloads**
  - **Problem:** Misprefetching at the boundary of XPLines is especially expensive
  - **Optimization:** Loading XPLines via SIMD instructions to avoid CPU prefetching

Conclusions

- A suite of carefully-designed microbenchmarks
  - Infer the design of on-DIMM read and write buffers
  - Discover interesting issues
    - Random read dominating performance, RAP latency, misprefetching penalty
- Three case studies showing substantial performance improvements
- Discussions on
  - G1 to G2 evolution, ADR vs. eADR, and programming guide

Questions?

lingfeng.xiang@mavs.uta.edu

https://github.com/lingfenghsiang/Persistent-Memory-Study