Persistent Memory Objects on the Cheap

International Conference on Supercomputing 2025.
Salt Lake City, Utah, United States. June 11.

Derrick Greenspan, Naveed Ul Mustafa,
Jongouk Choi, Mark Heinrich, Yan Solihin

CompArch & ARPERS research groups
Cyber Security and Privacy Research Cluster

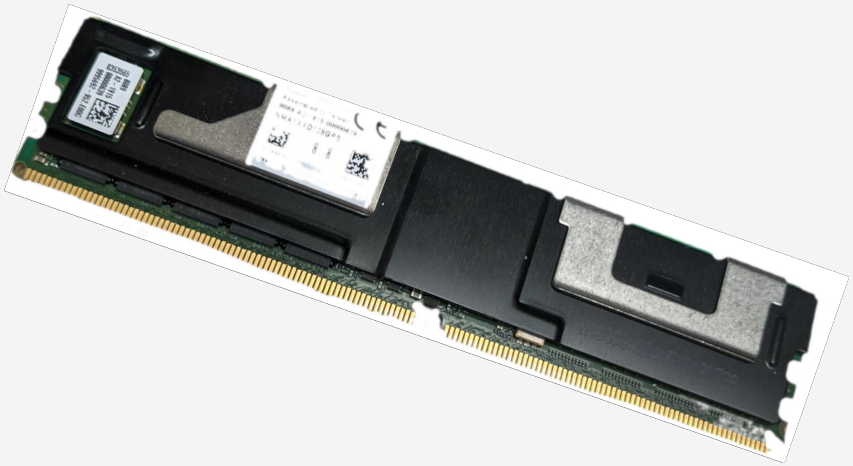
UCF | College of Engineering and Computer Science

UNIVERSITY OF CENTRAL FLORIDA

Overview

Persistent Memory (PM)

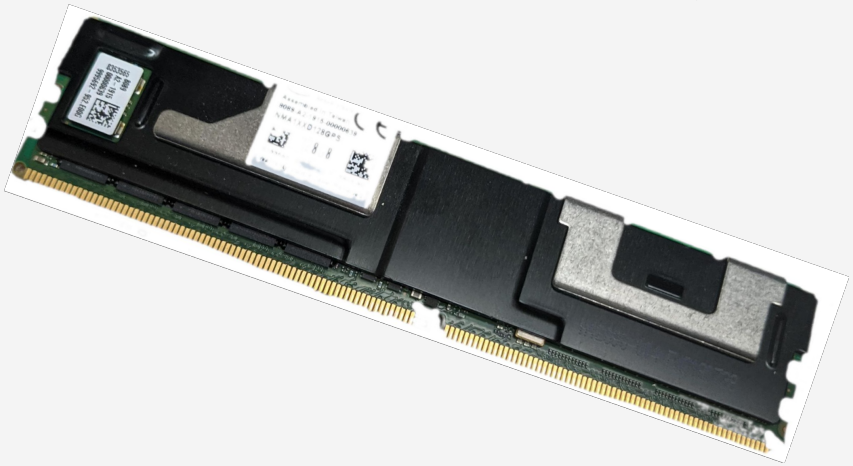
Example: Intel Optane, CXL Memory-Semantic SSDs



Characteristics

Persistent Memory (PM)

Example: Intel Optane, CXL Memory-Semantic SSDs

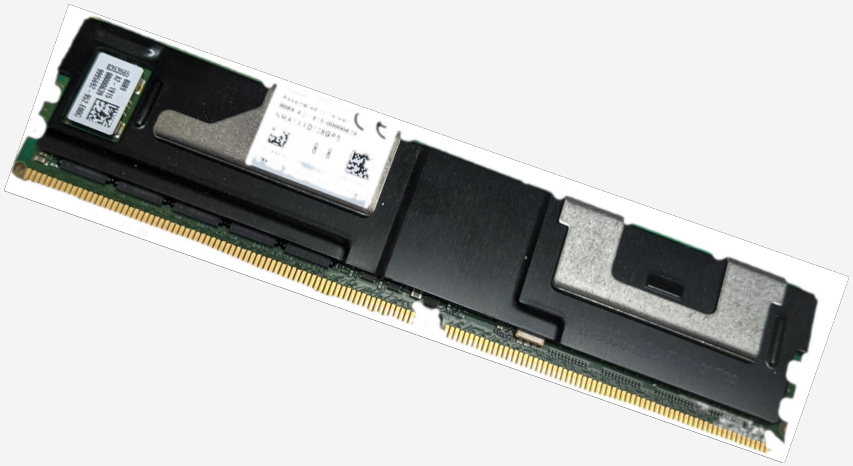


Characteristics

- Slow write performance/Decent read performance

Persistent Memory (PM)

Example: Intel Optane, CXL Memory-Semantic SSDs

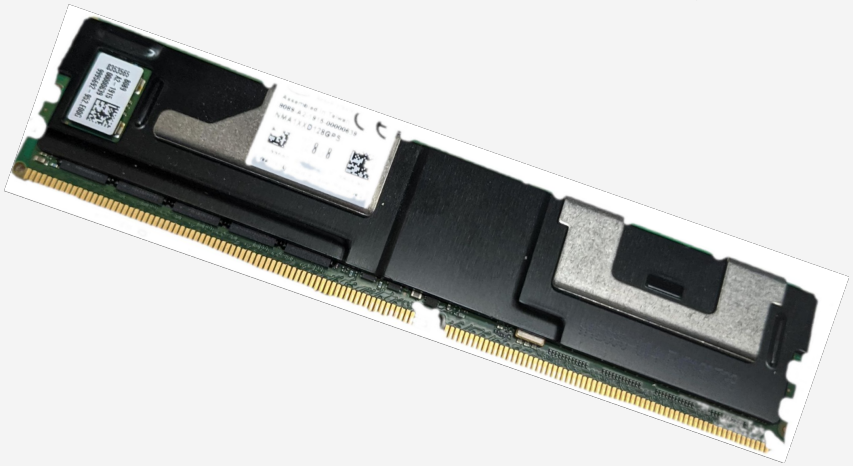


Characteristics

- Slow write performance/Decent read performance
- Poor multithreaded performance

Persistent Memory (PM)

Example: Intel Optane, CXL Memory-Semantic SSDs

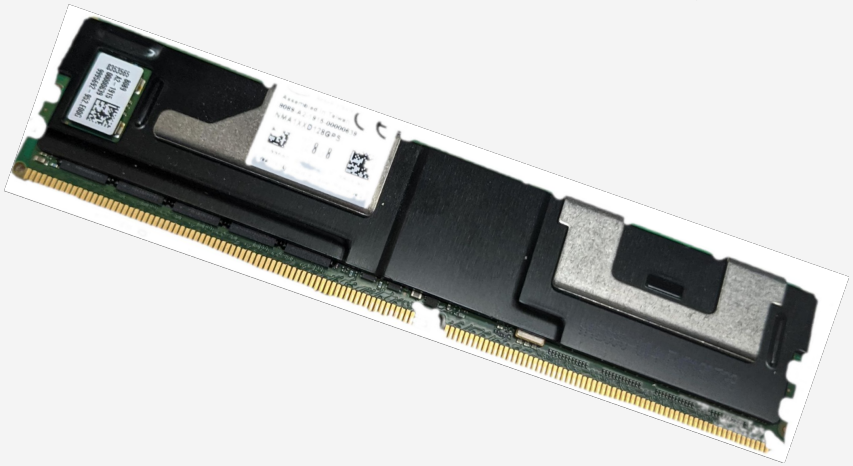


Characteristics

- Slow write performance/Decent read performance
- Poor multithreaded performance (more on this later)

Persistent Memory (PM)

Example: Intel Optane, CXL Memory-Semantic SSDs



Characteristics

- Slow write performance/Decent read performance
- Poor multithreaded performance (more on this later)

Note: Devices with PM usually have DRAM as well

Persistent Memory Objects (PMOs)

Primitives: pcreate()

Persistent Memory Objects (PMOs)

Primitives: pcreate() attach()

Persistent Memory Objects (PMOs)

Primitives: pcreate() attach() detach()

Persistent Memory Objects (PMOs)

Primitives: pcreate() attach() detach() psync()

Persistent Memory Objects (PMOs)

Primitives: pcreate() attach() detach() psync()

Properties

- File-less
- Potentially pointer-rich
- Accessed via load-store instructions
- Metadata managed by kernel

Persistent Memory Objects (PMOs)

Primitives: pcreate() attach() detach() psync()

Properties

- File-less
- Potentially pointer-rich
- Accessed via load-store instructions
- Metadata managed by kernel

Features

Persistent Memory Objects (PMOs)

Primitives: pcreate() attach() detach() psync()

Properties

- File-less
- Potentially pointer-rich
- Accessed via load-store instructions
- Metadata managed by kernel

Features

- Fast with minimal metadata

Persistent Memory Objects (PMOs)

Primitives: pcreate() attach() detach() psync()

Properties

- File-less
- Potentially pointer-rich
- Accessed via load-store instructions
- Metadata managed by kernel

Features

- Fast with minimal metadata
- Crash-consistency

Persistent Memory Objects (PMOs)

Primitives: pcreate() attach() detach() psync()

Properties

- File-less
- Potentially pointer-rich
- Accessed via load-store instructions
- Metadata managed by kernel

Features

- Fast with minimal metadata
- Crash-consistency
- Security at rest

Persistent Memory Objects (PMOs)

Primitives: pcreate() attach() detach() psync()

Properties

- File-less
- Potentially pointer-rich
- Accessed via load-store instructions
- Metadata managed by kernel

Features

- Fast with minimal metadata
- Crash-consistency
- Security at rest
- Integrity verification at rest

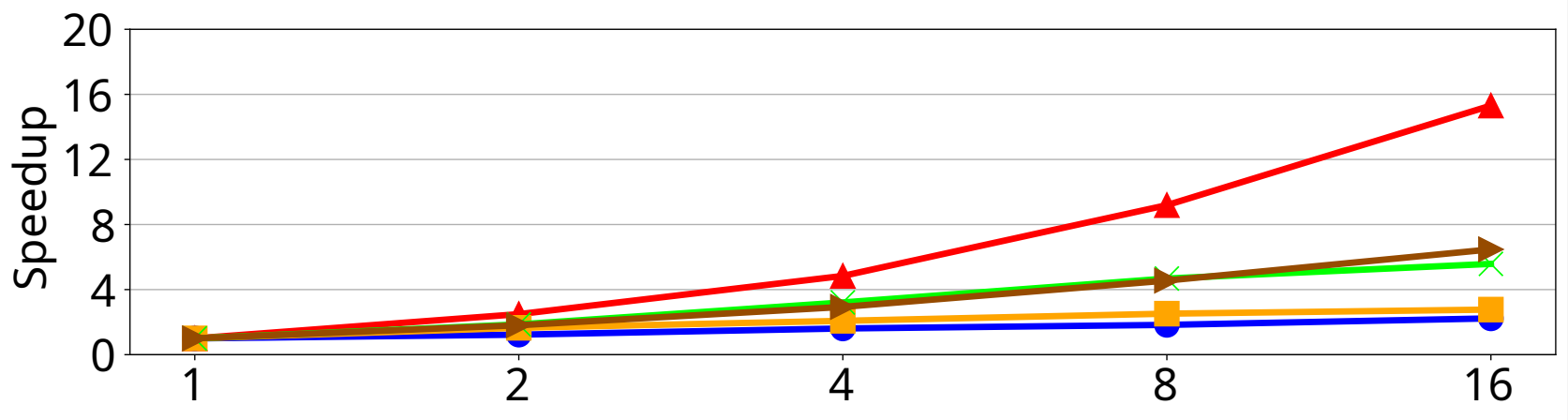
Thread Scaling

Prior work exhibited poor thread scaling

Thread Scaling

Prior work exhibited poor thread scaling

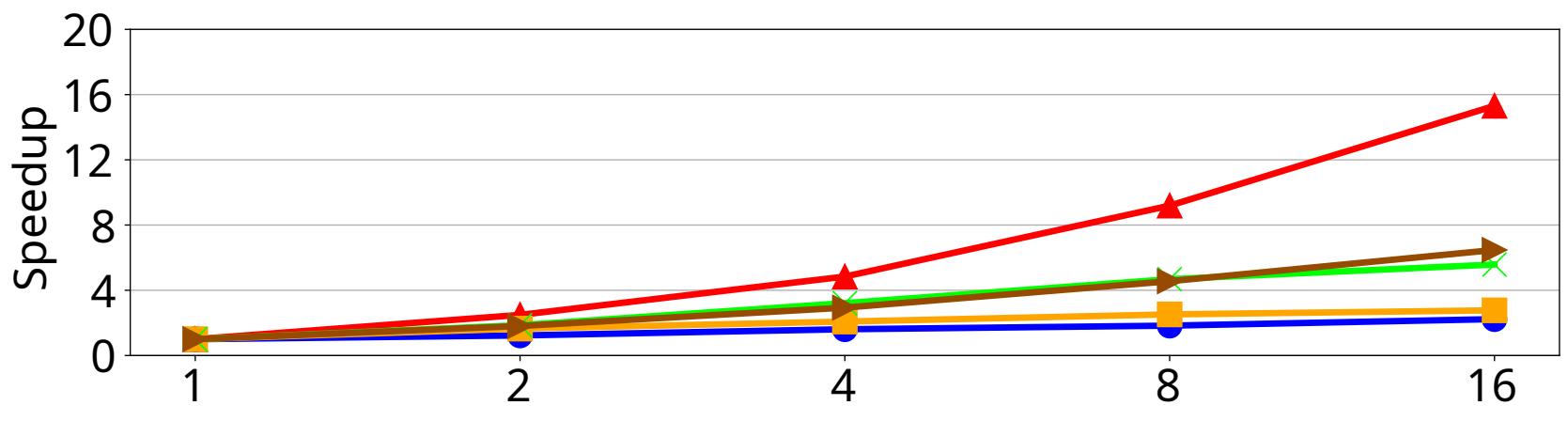
△ 2dConv, ○ Gauss, □ LU, × TMM, ▷ AVG



Thread Scaling

Prior work exhibited poor thread scaling

△ 2dConv, ○ Gauss, □ LU, × TMM, ▷ AVG



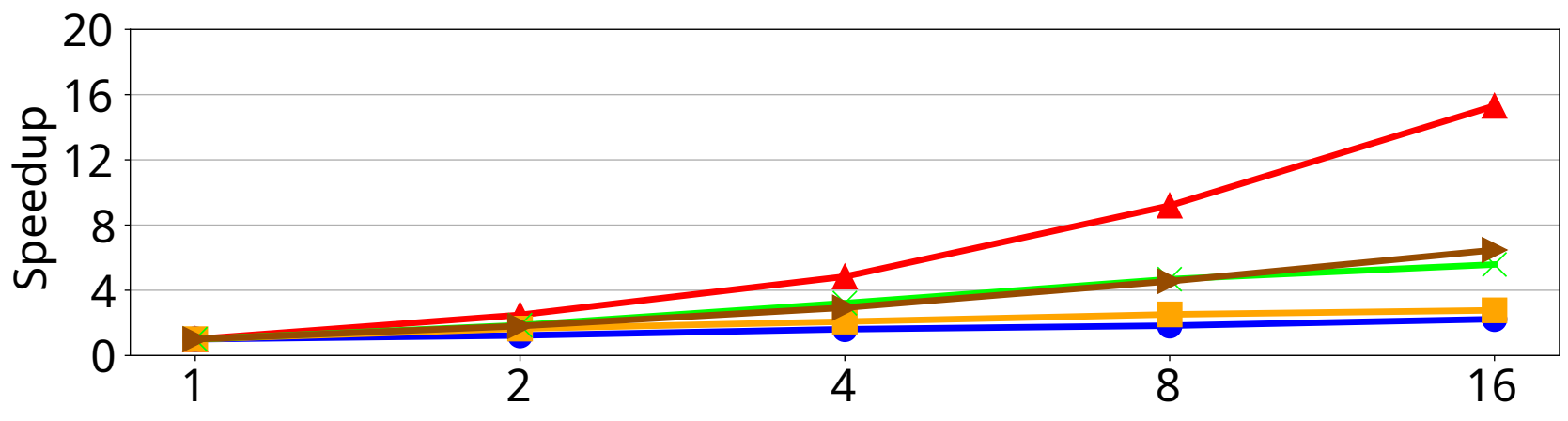
Reasons

- PMOs are hosted entirely in PM

Thread Scaling

Prior work exhibited poor thread scaling

△ 2dConv, ○ Gauss, □ LU, × TMM, ▷ AVG



Reasons

- PMOs are hosted entirely in PM
- Encryption and integrity verification on the critical path

CXL Devices

CXL: SOTA for PM

CXL Devices

CXL: SOTA for PM

Compute Express Link

- Utilizes PCIe interface
- Direct access from CPU to memory
- Heterogeneous memory pools
- Can use PM or Volatile Memory

CXL Devices

CXL: SOTA for PM

Compute Express Link

- Utilizes PCIe interface
- Direct access from CPU to memory
- Heterogeneous memory pools
- Can use PM or Volatile Memory

Additional latency

- From controller
- Comparable to NUMA

CXL Devices

CXL: SOTA for PM

Compute Express Link

- Utilizes PCIe interface
- Direct access from CPU to memory
- Heterogeneous memory pools
- Can use PM or Volatile Memory

Additional latency

- From controller
- Comparable to NUMA

Goal: High-Performance PMOs

CXL at-rest encryption

There is a way to do this...

CXL at-rest encryption

There is a way to do this...

CXL 3.1 Specification

- Trusted Execution Environments (TEE) Security Protocol (TSP)

CXL at-rest encryption

There is a way to do this...

CXL 3.1 Specification

- Trusted Execution Environments (TEE) Security Protocol (TSP)
- Range-based memory encryption

CXL at-rest encryption

There is a way to do this...

CXL 3.1 Specification

- Trusted Execution Environments (TEE) Security Protocol (TSP)
- Range-based memory encryption

I.e., Transparent hardware encryption

CXL at-rest encryption

There is a way to do this...

CXL 3.1 Specification

- Trusted Execution Environments (TEE) Security Protocol (TSP)
- Range-based memory encryption

I.e., Transparent hardware encryption

...more on this later.

Overview

Threat Model

Goal: Protect at-rest data from disclosure/corruption

Threat Model

Goal: Protect at-rest data from disclosure/corruption

Out of Scope

- Side-channel attacks
- Data-remanence attacks (DRAM)

Avoiding PM Pathologies

Prior work: PMO entirely in PM

Avoiding PM Pathologies

Prior work: PMO entirely in PM

- Crash consistency simple

Avoiding PM Pathologies

Prior work: PMO entirely in PM

- Crash consistency simple
- High latency, low write bandwidth

Avoiding PM Pathologies

Prior work: PMO entirely in PM

- Crash consistency simple
- High latency, low write bandwidth

LPMO can exploit DRAM as cache **without** hardware support

Avoiding PM Pathologies

Prior work: PMO entirely in PM

- Crash consistency simple
- High latency, low write bandwidth

LPMO can exploit DRAM as cache **without** hardware support
DRAM as cache = Reconfigurable Memory

Reconfigurable Memory Hierarchy

Challenges

Reconfigurable Memory Hierarchy

Challenges

Which data should be placed in DRAM?

Reconfigurable Memory Hierarchy

Challenges

Which data should be placed in DRAM?

- All data in DRAM

Reconfigurable Memory Hierarchy

Challenges

Which data should be placed in DRAM?

- All data in DRAM
- **Shadow in DRAM, primary in PM**
  Psync: Temporary Shadow Page (TSC) in PM, copy to Primary

Reconfigurable Memory Hierarchy

Challenges

Which data should be placed in DRAM?

- All data in DRAM
- **Shadow in DRAM, primary in PM**
  Psync: Temporary Shadow Page (TSC) in PM, copy to Primary

Reconfigurable Memory Hierarchy

Challenges

Which data should be placed in DRAM?

- All data in DRAM
- **Shadow in DRAM, primary in PM**
  Psync: Temporary Shadow Page (TSC) in PM, copy to Primary

What data should be encrypted?

Reconfigurable Memory Hierarchy

Challenges

Which data should be placed in DRAM?

- All data in DRAM
- **Shadow in DRAM, primary in PM**
  Psync: Temporary Shadow Page (TSC) in PM, copy to Primary

What data should be encrypted?

- Primary and shadow page encrypted

Reconfigurable Memory Hierarchy

Challenges

Which data should be placed in DRAM?

- All data in DRAM
- **Shadow in DRAM, primary in PM**
  Psync: Temporary Shadow Page (TSC) in PM, copy to Primary

What data should be encrypted?

- Primary and shadow page encrypted
- Primary and shadow in plaintext

Reconfigurable Memory Hierarchy

Challenges

Which data should be placed in DRAM?

- All data in DRAM
- **Shadow in DRAM, primary in PM**
  Psync: Temporary Shadow Page (TSC) in PM, copy to Primary

What data should be encrypted?

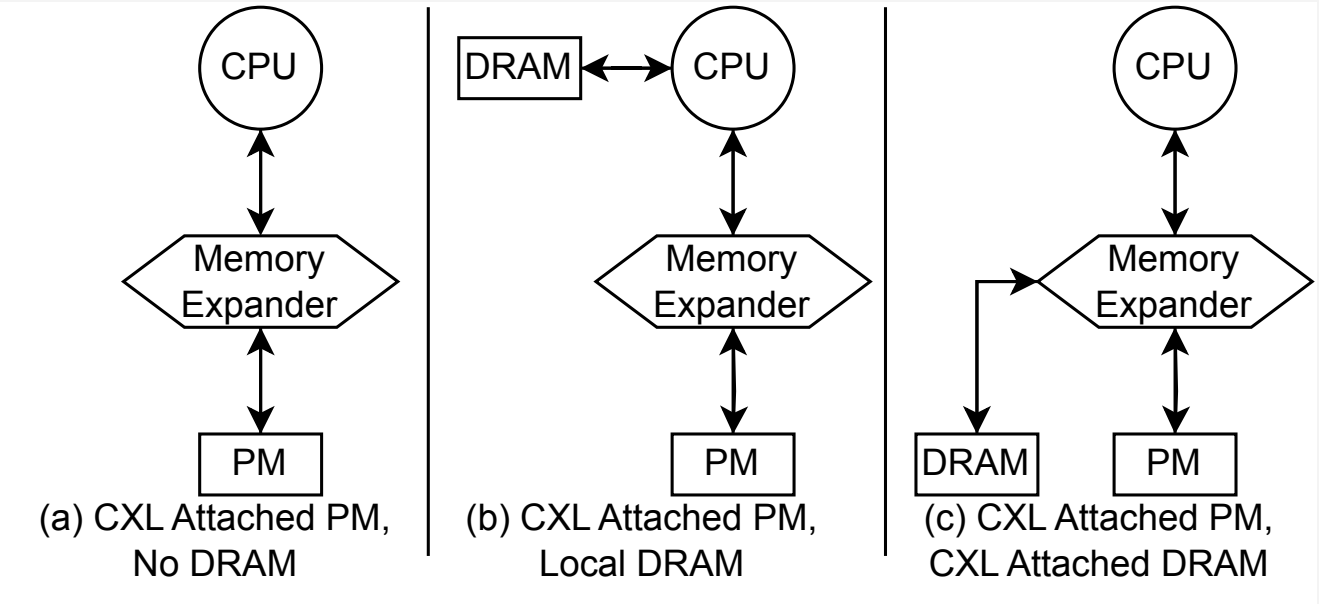
- Primary and shadow page encrypted
- Primary and shadow in plaintext
- **Shadow in plaintext, primary in ciphertext**

Reconfigurable Memory Hierarchy (Part 2)

Optane PM uses local DRAM

Reconfigurable Memory Hierarchy (Part 2)

Optane PM uses local DRAM



CXL can place system on either side of memory expander

Reduce page fault latency

Prior work: Demand Faulting

Why not **predict** when pages are needed?

Reduce page fault latency

Prior work: Demand Faulting

Why not **predict** when pages are needed?

Example Solution: Stream Buffer

- On fault, predict next X sequential pages (**depth**)
- Works well for access patterns amenable to prediction

Overview

Evaluation

Evaluated Benchmarks

- Microbenchmarks
  - 2d Convolution (2dConv)
  - Gaussian Elimination (Gauss)
  - LU Decomposition (LU)
  - Tiled Matrix Matrix Multiplication (TMM)
- Filebench (Fileserver, VarMail, WebProxy, WebServer)
- LMDB

Evaluation

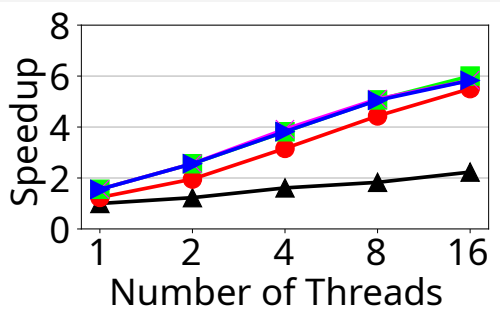
Evaluated Benchmarks

- Microbenchmarks
  - 2d Convolution (2dConv)
  - Gaussian Elimination (Gauss)
  - LU Decomposition (LU)
  - Tiled Matrix Matrix Multiplication (TMM)
- Filebench (Fileserver, VarMail, WebProxy, WebServer)
- LMDB

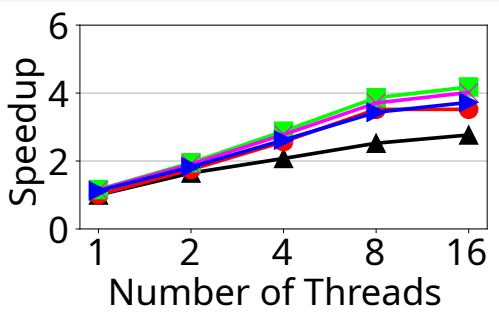
Component	Specifications
MB	Supermicro X11DPi-NT
CPU	2×Intel Xeon Gold 6230 (20 cores)
DRAM	4 × 32GiB DDR4 @ 2666MHz
PM	4 × 128GiB Intel Optane DIMM
OS	AlmaLinux 9.0; Linux 5.15.157

LPMO Performance

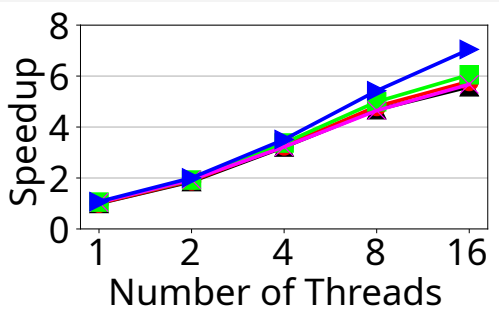
△ O, ○ D, □ D2, × D4, ▷ D8



(a) Gauss



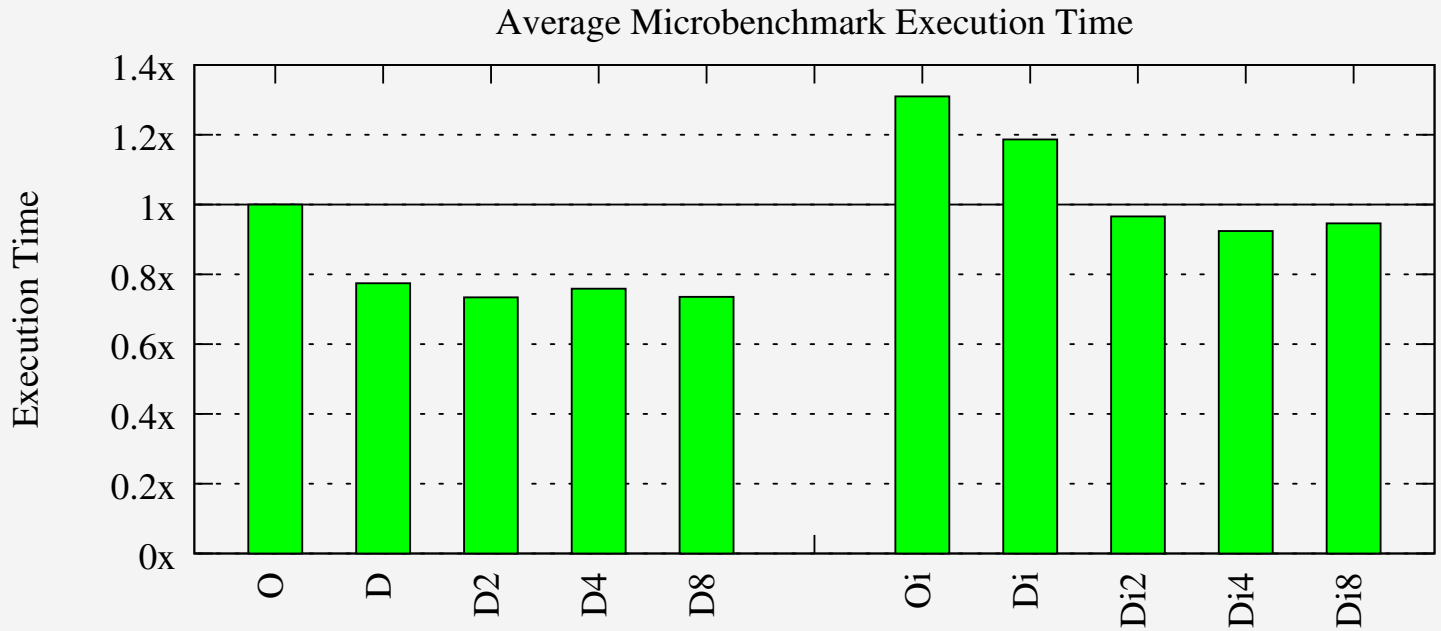
(b) LU



(c) TMM

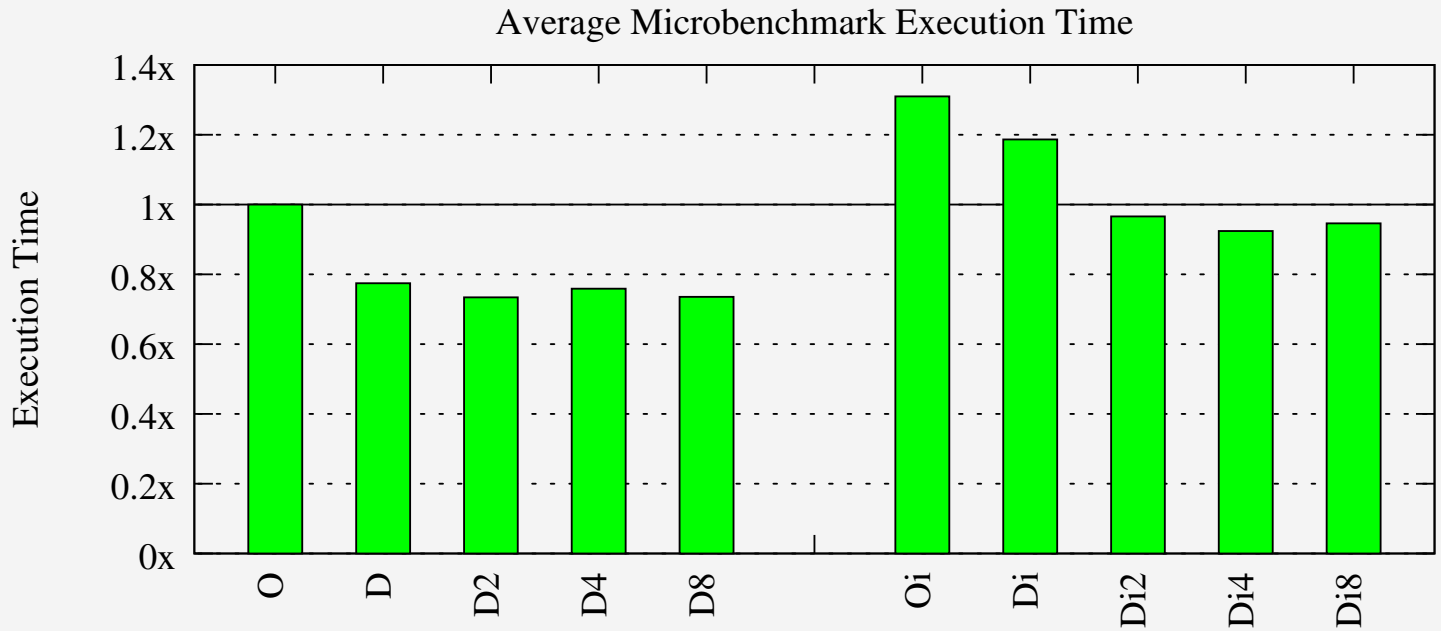
All benchmarks have better thread scaling!

LPMO Performance



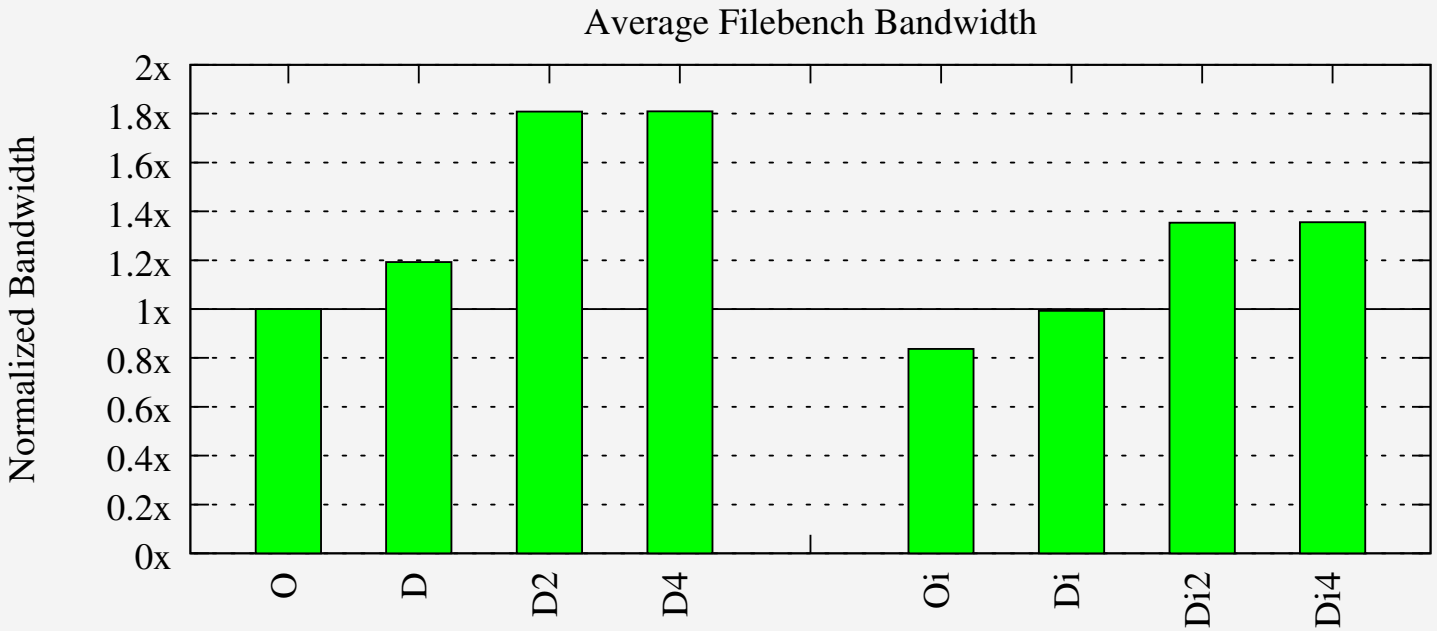
- DRAM reduces execution time by $\approx 21\%$

LPMO Performance



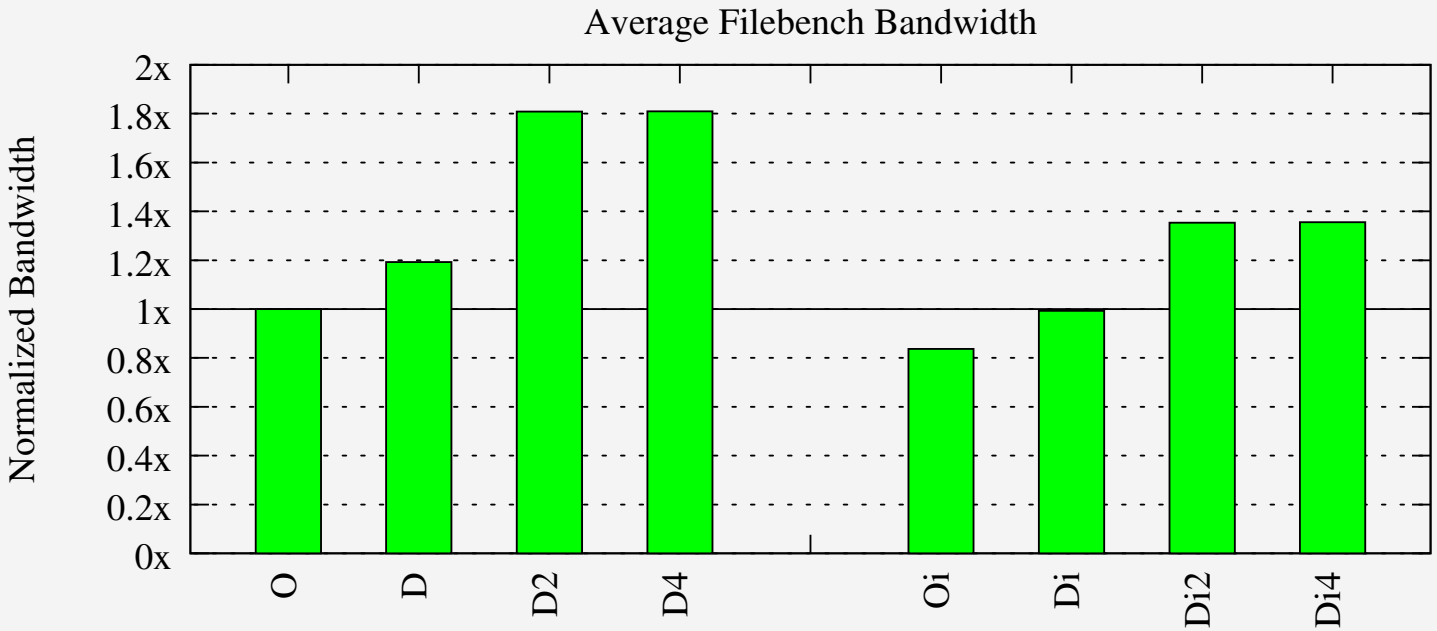
- DRAM reduces execution time by $\approx 21\%$
- IV + Prediction faster than original GPMO design w/o IV

LPMO Performance - Filebench



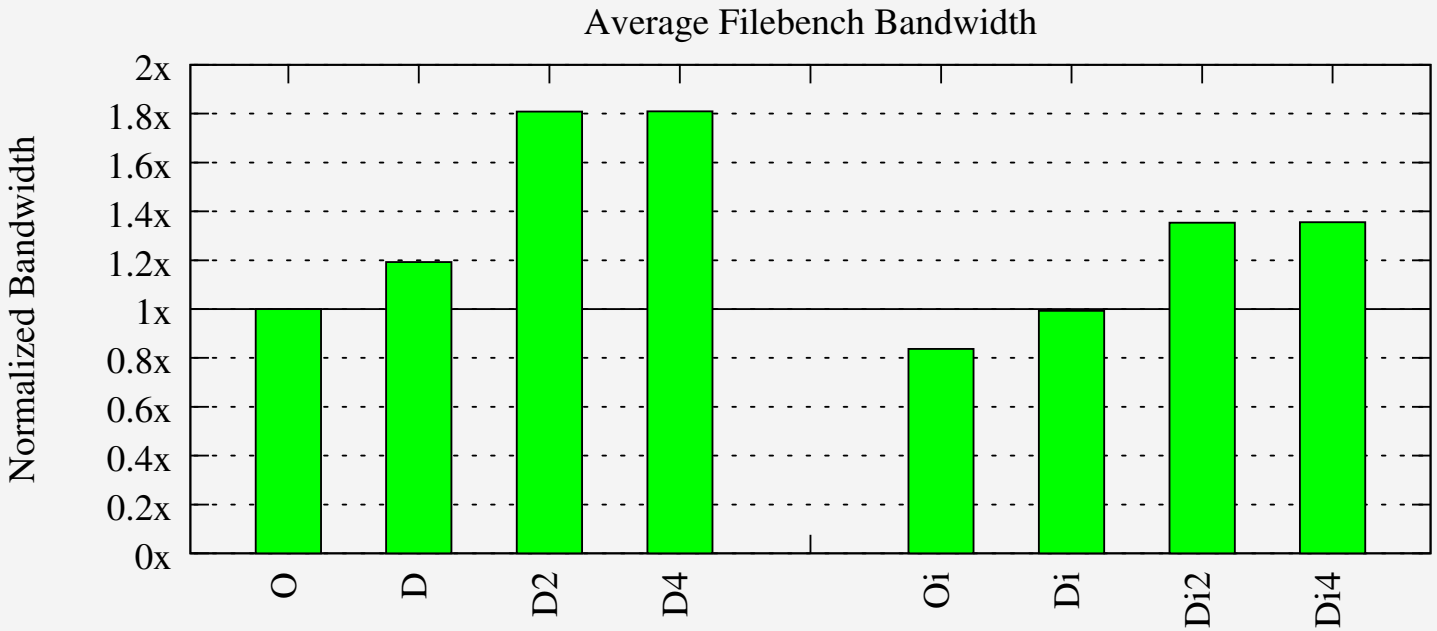
- Only 1.19× faster with DRAM

LPMO Performance - Filebench



- Only 1.19× faster with DRAM
- 1.81× **faster with page prediction**

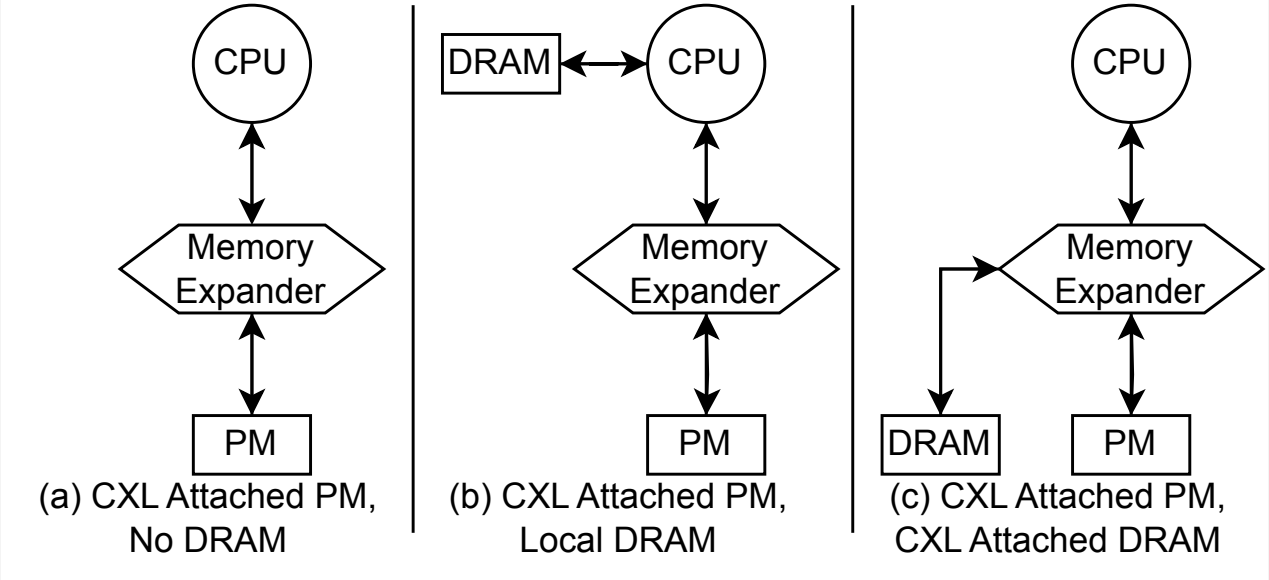
LPMO Performance - Filebench



- Only 1.19× faster with DRAM
- 1.81× **faster with page prediction**
- 1.37× faster with page prediction & **IV**

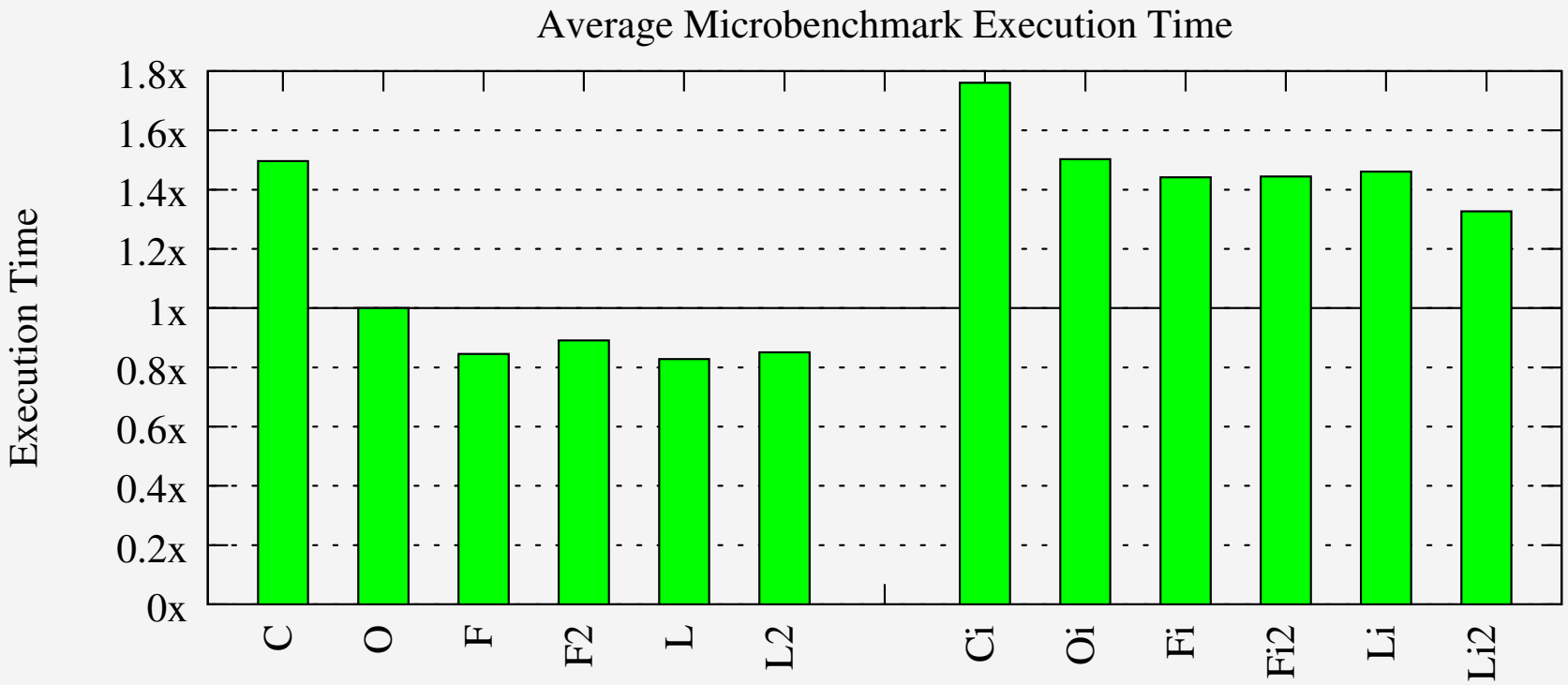
CXL Performance

Perform same tests, but emulate CXL latency



- Use opposite-node Optane
- Near configuration: cache allocated from local node
- Far configuration: cache allocated from opposite node

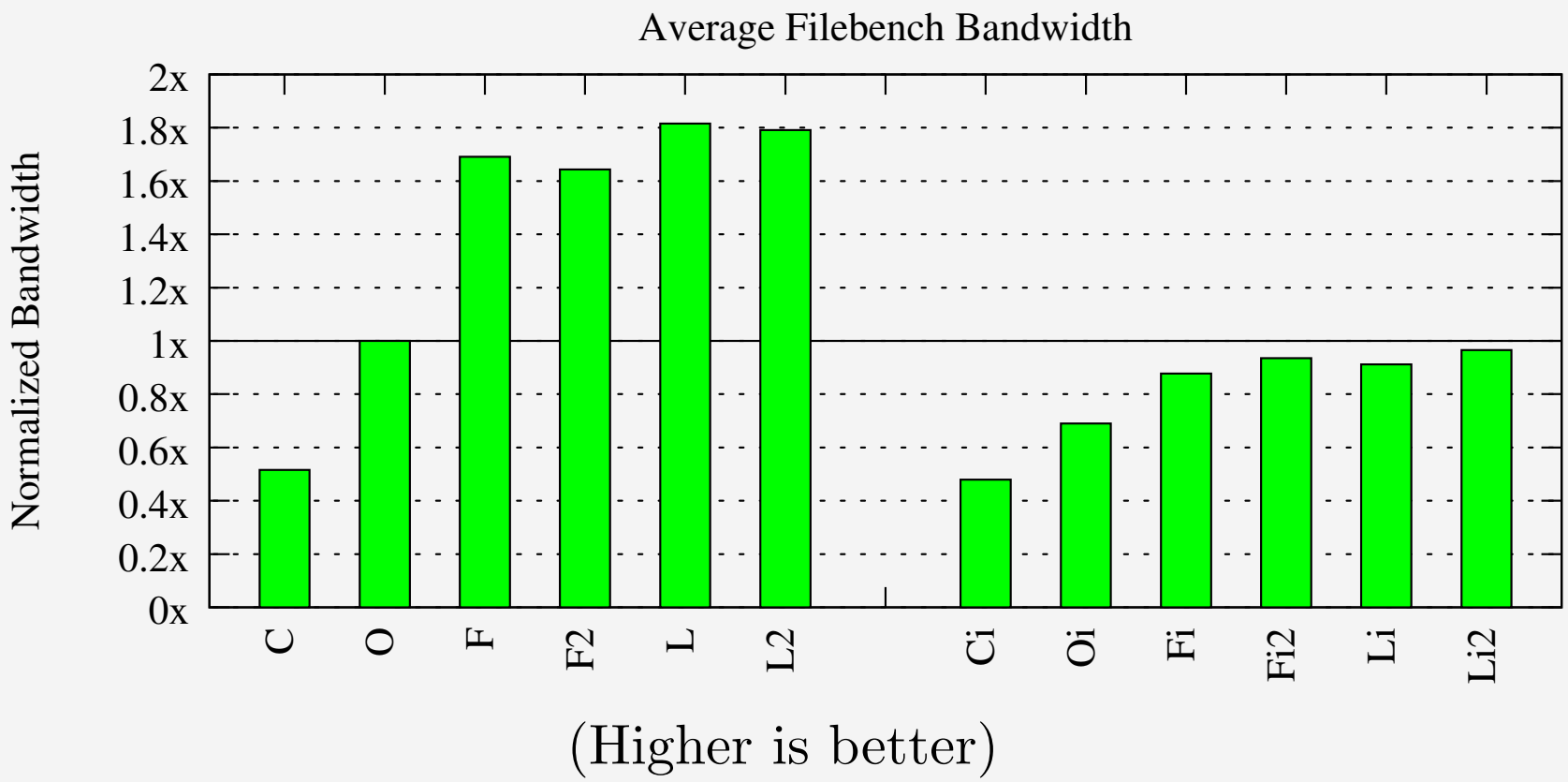
CXL Performance



(Lower is better)

- With CXL alone: 50% slower than original
- With DRAM: 20% faster (despite CXL latency)

CXL Performance - Filebench



Overview

Conclusion

LPMO

- Software-based DRAM caching
  - Up to 1.25× faster

Conclusion

LPMO

- Software-based DRAM caching
  - Up to 1.25× faster
- Predictive Decryption
  - Up to 1.81× faster

Conclusion

LPMO

- Software-based DRAM caching
  - Up to 1.25× faster
- Predictive Decryption
  - Up to 1.81× faster

CXL

- Introduced Reconfigurable Memory Hierarchy
- CXL latency can be masked by LPMO optimizations

Q & A

Thank You!

Any questions?