Bandwidth Expansion via CXL: A Pathway to Accelerating In-Memory Analytical Processing

Wentao Huang*, Mo Sha†, Mian Lu‡, Yuqiang Chen‡, Bingsheng He*, Kian-Lee Tan*

*National University of Singapore

†Alibaba Cloud

‡4Paradigm Inc.

Growing Memory Demands

Growing Memory Demands

- Data driven application

Growing Memory Demands

- Data driven application
  - high performance computing

Growing Memory Demands

- Data driven application
  - high performance computing
  - video processing

Growing Memory Demands

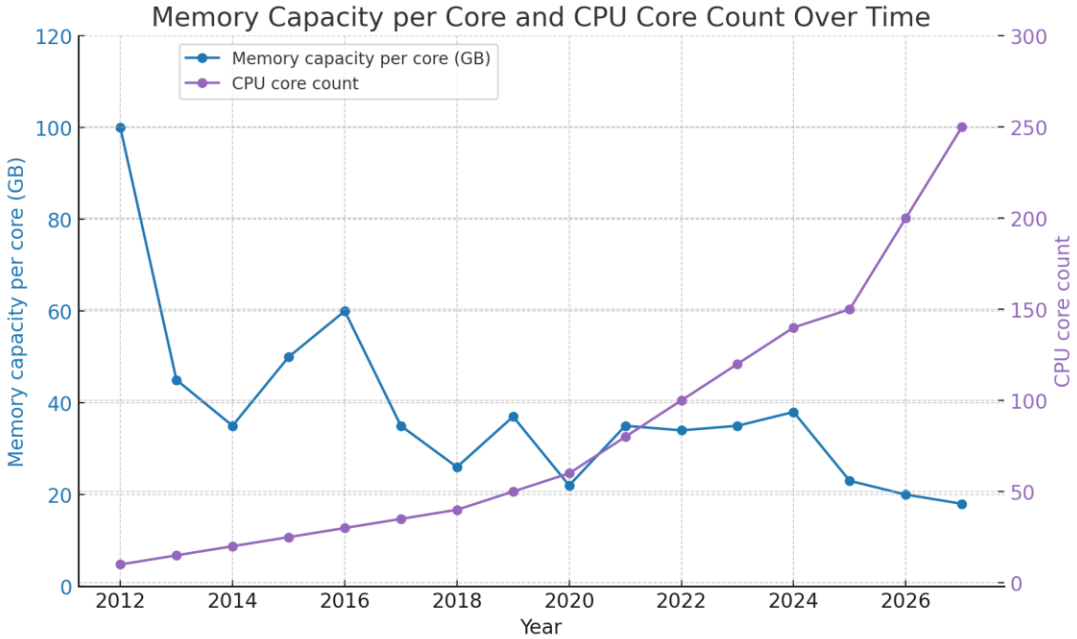
- Data driven application
  - high performance computing
  - video processing
  - large language model
  - ……

Growing Memory Demands

- Data driven application
  - high performance computing
  - video processing
  - large language model
  - ……
- More data → more memory
  - larger capacity
  - faster memory access
  - lower TCO

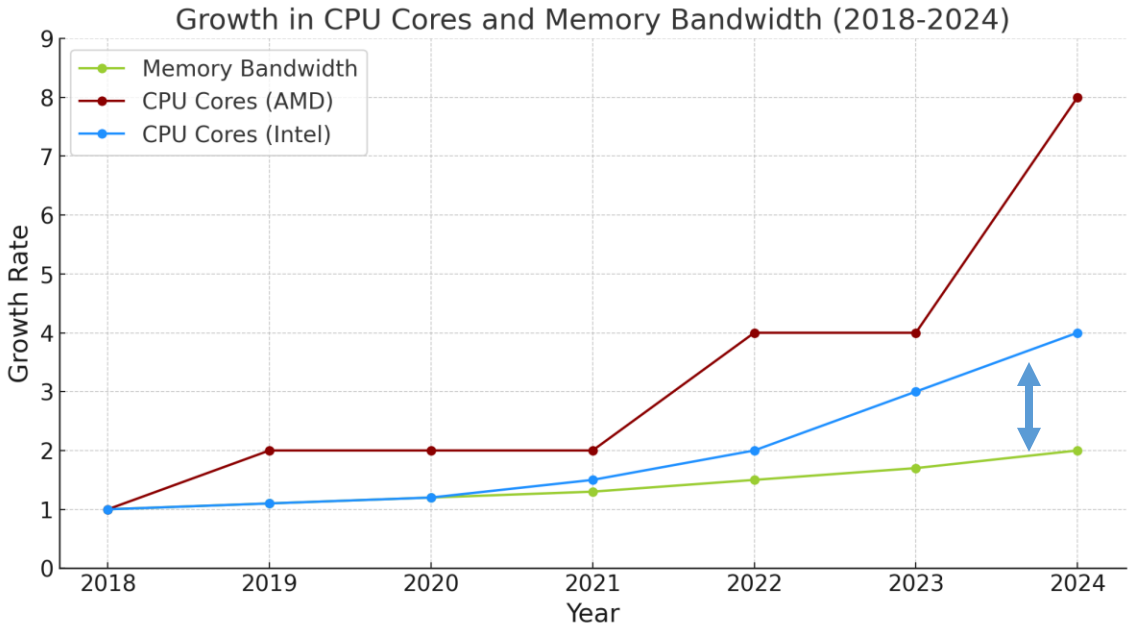
Hitting the Memory Wall

- Memory capacity



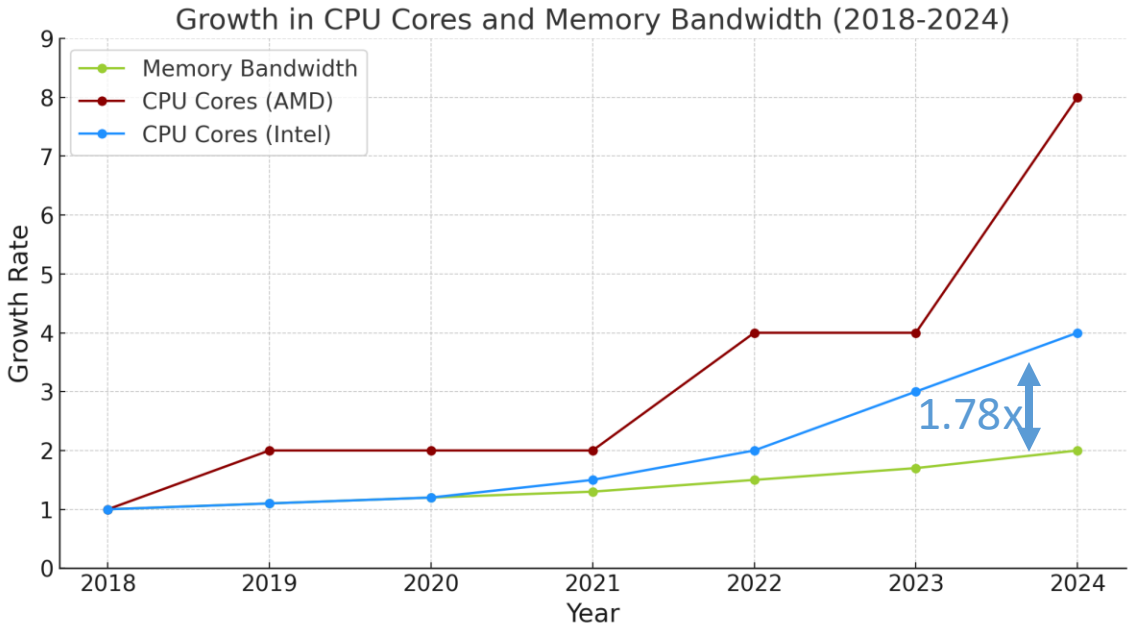
Hitting the Memory Wall

- Memory bandwidth



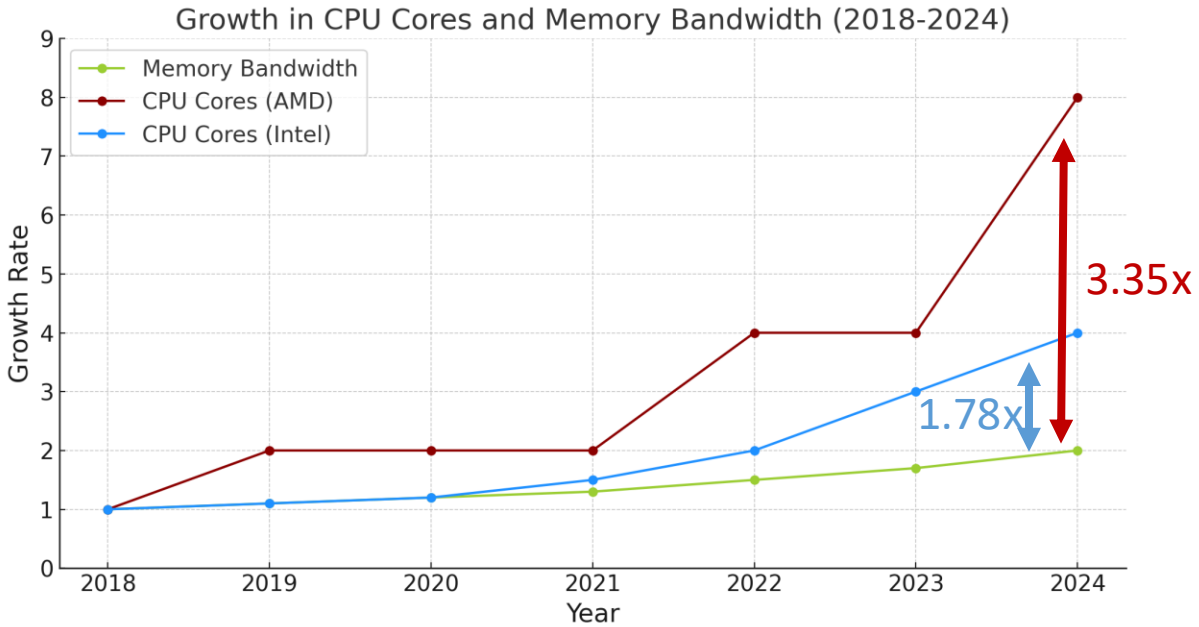
Hitting the Memory Wall

- Memory bandwidth



Hitting the Memory Wall

- Memory bandwidth



Memory Scaling Challenges

Memory Scaling Challenges

- More channels per socket

Memory Scaling Challenges

- More channels per socket
  - increasing pin count and routing complexity

Memory Scaling Challenges

- More channels per socket
  - increasing pin count and routing complexity
- More memory DIMMs per channel

Memory Scaling Challenges

- More channels per socket
  - increasing pin count and routing complexity
- More memory DIMMs per channel
  - bandwidth reduction due to signal integrity issues

Memory Scaling Challenges

- More channels per socket
  - increasing pin count and routing complexity
- More memory DIMMs per channel
  - bandwidth reduction due to signal integrity issues
- Memory packing technologies, e.g., 3D stacking

Memory Scaling Challenges

- More channels per socket
  - increasing pin count and routing complexity
- More memory DIMMs per channel
  - bandwidth reduction due to signal integrity issues
- Memory packing technologies, e.g., 3D stacking
  - escalating TCO

Memory Scaling Challenges

- More channels per socket
  - increasing pin count and routing complexity
- More memory DIMMs per channel
  - bandwidth reduction due to signal integrity issues
- Memory packing technologies, e.g., 3D stacking
  - escalating TCO
- ……

CXL: Compute Express Link

CXL: Compute Express Link

- New protocols based on PCIe
  - CXL.cache (cache coherence)
  - CXL.mem (memory expansion)
  - CXL.io (peripheral configuration)

CXL: Compute Express Link

- New protocols based on PCIe
  - CXL.cache (cache coherence)
  - CXL.mem (memory expansion)
  - CXL.io (peripheral configuration)

- CXL specifications
  - CXL 1.1 — single machine
  - CXL 2.0 — 2-16 machines (single switch)
  - CXL 3.0 — 100+ machines (multiple switches)

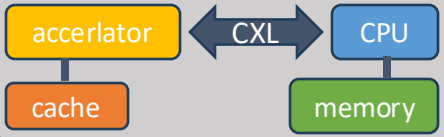
CXL End-Point Devices

CXL End-Point Devices

Type 1: accerlators w/o memory.

Usage:
- PGAS NIC
- NIC atomics

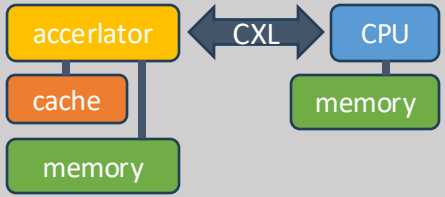
Protocol:
- CXL.cache
- CXL.io



Type 2: accerlators w/ memory.

Usage:
- GPU
- FPGA

Protocol:
- CXL.cache
- CXL.mem
- CXL.io

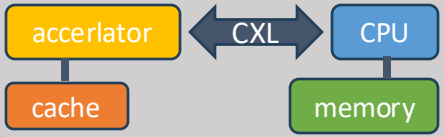


CXL End-Point Devices

Type 1: accerlators w/o memory.

Usage:
- PGAS NIC
- NIC atomics

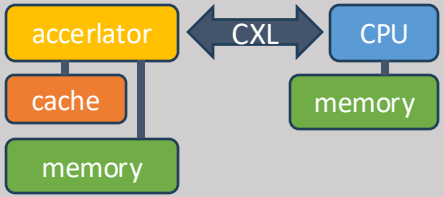
Protocol:
- CXL.cache
- CXL.io



Type 2: accerlators w/ memory.

Usage:
- GPU
- FPGA

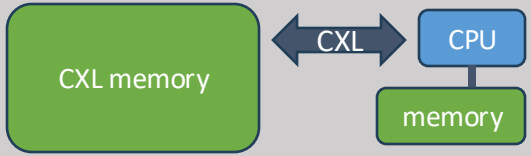
Protocol:
- CXL.cache
- CXL.mem
- CXL.io



Type 3: memory buffer.

Usage:
- capacity extension
- tiered memory system

Protocol:
- CXL.io
- CXL.mem



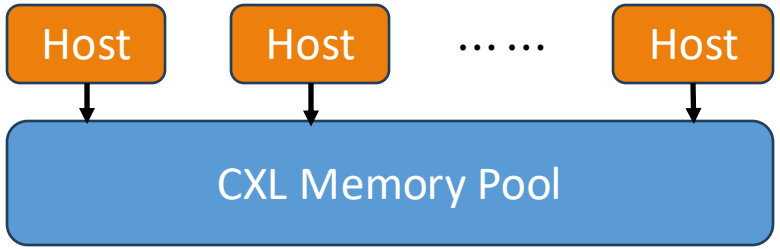
Scale-Out vs. Scale-Up

Scale-Out vs. Scale-Up

- Scale-out
  - CXL 2.0/3.0
  - resource pooling
  - economical cost

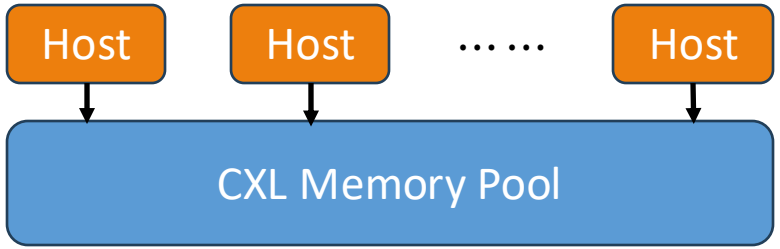
Scale-Out vs. Scale-Up

- Scale-out
  - CXL 2.0/3.0
  - resource pooling
  - economical cost



Scale-Out vs. Scale-Up

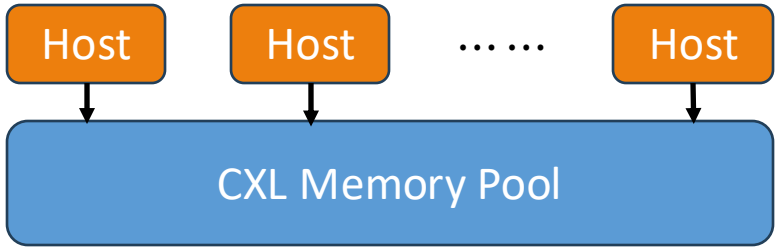
- Scale-out
  - CXL 2.0/3.0
  - resource pooling
  - economical cost



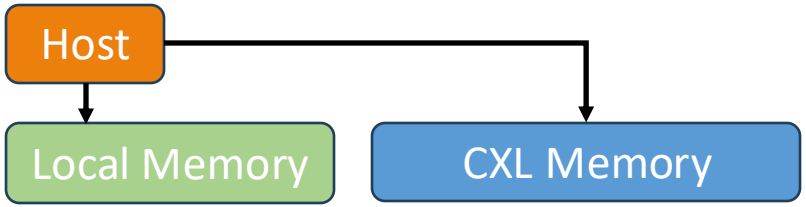
- Scale-up
  - CXL 1.1
  - capacity expansion
  - bandwidth expansion
  - lower TCO

Scale-Out vs. Scale-Up

- Scale-out
  - CXL 2.0/3.0
  - resource pooling
  - economical cost



- Scale-up
  - CXL 1.1
  - capacity expansion
  - bandwidth expansion
  - lower TCO



CXL Scale-Up Expansion

- Expansion via tiering

CXL Scale-Up Expansion

- Expansion via tiering

A Three-Tier Buffer Manager Integrating CXL Device Memory for Database Systems

Niklas Riekenbrauck
Hasso Plattner Institute
Potsdam, Germany
niklas.riekenbrauck@student.hpi.de

Marcel Weisgut
Hasso Plattner Institute
Potsdam, Germany
marcel.weisgut@hpi.de

Daniel Lindner
Hasso Plattner Institute
Potsdam, Germany
daniel.lindner@hpi.de

Tilmann Rabl
Hasso Plattner Institute
Potsdam, Germany
tilmann.rabl@hpi.de

***Abstract*—Compute Express Link (CXL) is a new interconnect for attaching byte-addressable memory on PCI-connected devices to a CPU. The interconnect allows a database system to place data on local memory, CXL device memory, and persistent disk storage. While three-tier buffer managers integrating persistent memory (PMem) exist, CXL device memory has not been integrated into a multi-tier buffer manager architecture. Existing three-tier buffer managers integrating PMem use pointer swizzling to address buffered pages, which is an invasive and hard-to-implement technique. This work presents a three-tier buffer manager that integrates CXL device memory. The design combines hardware-supported virtual memory for efficient page translation and a probabilistic page migration policy to determine on which tier pages are located. We demonstrate that these approaches combined allow a simple integration of CXL device memory into a database system. We evaluate the buffer manager with different configurations and workloads based on the YCSB benchmark on a CXL Type 3 device prototype. Our evaluation demonstrates that expanding a server's memory with CXL device memory can be used to keep more data in memory and to reduce spilling data to slow disk storage.**

***Index Terms*—CXL, buffer manager, page management, database system, virtual memory, probabilistic page migration**

I. Introduction

Compute Express Link (CXL) is a new interconnect based on the physical layer of PCIe. CXL can connect a peripheral device with a CPU, allowing cache-coherent access to the device memory [1]. Accessing memory over CXL exhibits different memory characteristics, such as higher latency than local memory connected via Double Data Rate (DDR) [1], [2].

Traditional disk-based database management systems (DBMSs) use secondary disk storage as primary data location. For query processing, a buffer manager loads data into local memory. With additional CXL device memory, data can be located on three tiers: on byte-addressable local and device memory and on persistent disk storage. While three-tier buffer managers exist for DRAM, persistent memory (PMem), and solid-state drive (SSD) [3], [4], CXL device memory has not been integrated into a multi-tier buffer manager architecture.

HyMem is a single-threaded buffer manager using PMem and DRAM as selective caches on top of the SSD level [3]. Zhou et al. [4] extended the work on HyMem with *Spitfire*, a concurrent buffer manager. It uses a probabilistic migration policy to determine on which tier pages are located and has superior performance over HyMem's page migration.

HyMem and Spitfire use pointer swizzling to address buffered pages, which is an invasive technique that requires a buffer-managed data structure to be adapted accordingly [5]. Leis et al. proposed hardware-supported virtual memory-based translation of page identifiers (PIDs) to physical memory addresses as a non-invasive and easy-to-implement alternative to pointer swizzling [5]. While the approach shows high performance and low implementation complexity, it lacks support for multiple byte-addressable memory tiers.

In this work, we present a three-tier buffer manager that integrates CXL device memory. The design combines virtual memory-based PID translation [5] and a probabilistic page migration policy [4] to determine on which tier pages are located. This work demonstrates that these approaches combined allow a simple integration of CXL device memory into a DBMS. We evaluate the buffer manager design and its components in an isolated manner with different configurations and workloads based on the YCSB benchmark [6]. Our evaluation demonstrates that expanding a server's memory with CXL device memory can be used for a buffer manager to keep more data in memory and to reduce spilling data to slow disk storage. We present primitives to integrate the proposed design into an in-memory DBMS and demonstrate the integration of the buffer manager into the in-memory DBMS *Hyrise* [7]

In summary, this work makes the following contributions:

1) We demonstrate the integration of CXL device memory into a database system's buffer manager using virtual memory-based PID translation and probabilistic page migration (Section III). We provide integration details and an open-source implementation[1] (Section III-G).
2) We experimentally evaluate the buffer manager with prototypical CXL device memory and show its benefit of supporting larger-than-local-memory workloads with higher throughput than a traditional two-tier design locating data only on local memory and SSD (Section IV).
3) We discuss how page migration across multiple memory tiers can further be optimized (Section V).

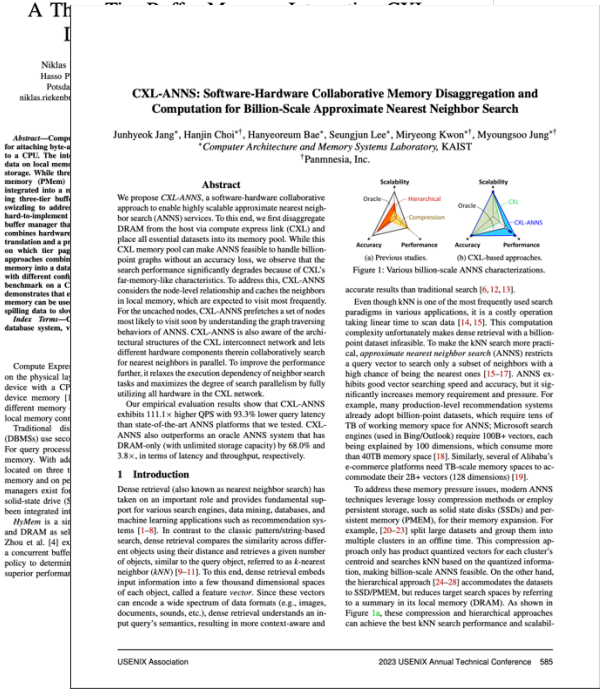
II. Background

This section introduces the CXL interconnect and buffer pool management concepts that we build our work upon.

[1]Source code: https://github.com/hyrise/hyrise/tree/paper/buffermanager.

CXL Scale-Up Expansion

- Expansion via tiering



CXL-ANNS: Software-Hardware Collaborative Memory Disaggregation and Computation for Billion-Scale Approximate Nearest Neighbor Search

Junhyeok Jang*, Hanjin Choi*†, Hanyeoreum Bae*, Seungjun Lee*, Miryeong Kwon*†, Myoungsoo Jung*†
*Computer Architecture and Memory Systems Laboratory, KAIST
†Panmnesia, Inc.

Abstract

We propose CXL-ANNS, a software-hardware collaborative approach to enable highly scalable approximate nearest neighbor search (ANNS) services. To this end, we first disaggregate DRAM from the host via compute express link (CXL) and place all essential datasets into its memory pool. While this CXL memory pool can make ANNS feasible to handle billion-point graphs without an accuracy loss, we observe that the search performance significantly degrades because of CXL's far-memory-like characteristics. To address this, CXL-ANNS considers the node-level relationship and caches the neighbors in local memory, which are expected to visit most frequently. For the uncached nodes, CXL-ANNS prefetches a set of nodes most likely to visit soon by understanding the graph traversing behaviors of ANNS. CXL-ANNS is also aware of the architectural structures of the CXL interconnect network and lets different hardware components therein collaboratively search for nearest neighbors in parallel. To improve the performance further, it relaxes the execution dependency of neighbor search tasks and maximizes the degree of search parallelism by fully utilizing all hardware in the CXL network.

Our empirical evaluation results show that CXL-ANNS exhibits 111.1× higher QPS with 93.3% lower query latency than state-of-the-art ANNS platforms that we tested. CXL-ANNS also outperforms an oracle ANNS system that has DRAM-only (with unlimited storage capacity) by 68.0% and 3.8×, in terms of latency and throughput, respectively.

1 Introduction

Dense retrieval (also known as nearest neighbor search) has taken on an important role and provides fundamental support for various search engines, data mining, databases, and machine learning applications such as recommendation systems [1–8]. In contrast to the classic pattern/string-based search, dense retrieval compares the similarity across different objects using their distance and retrieves a given number of objects, similar to the query object, referred to as k-nearest neighbor (kNN) [9–11]. To this end, dense retrieval embeds input information into a few thousand dimensional spaces of each object, called a feature *vector*. Since these vectors can encode a wide spectrum of data formats (e.g., images, documents, sounds, etc.), dense retrieval understands an input query's semantics, resulting in more context-aware and accurate results than traditional search [6, 12, 13].

Even though kNN is one of the most frequently used search paradigms in various applications, it is a costly operation taking linear time to scan data [14, 15]. This computation complexity unfortunately makes dense retrieval with a billion-point dataset infeasible. To make the kNN search more practical, *approximate nearest neighbor search* (ANNS) restricts a query vector to search only a subset of neighbors with a high chance of being the nearest ones [15–17]. ANNS exhibits good vector searching speed and accuracy, but it significantly increases memory requirement and pressure. For example, many production-level recommendation systems already adopt billion-point datasets, which require tens of TB of working memory space for ANNS; Microsoft search engines (used in Bing/Outlook) require 100B+ vectors, each being explained by 100 dimensions, which consume more than 40TB memory space [18]. Similarly, several of Alibaba's e-commerce platforms need TB-scale memory spaces to accommodate their 2B+ vectors (128 dimensions) [19].

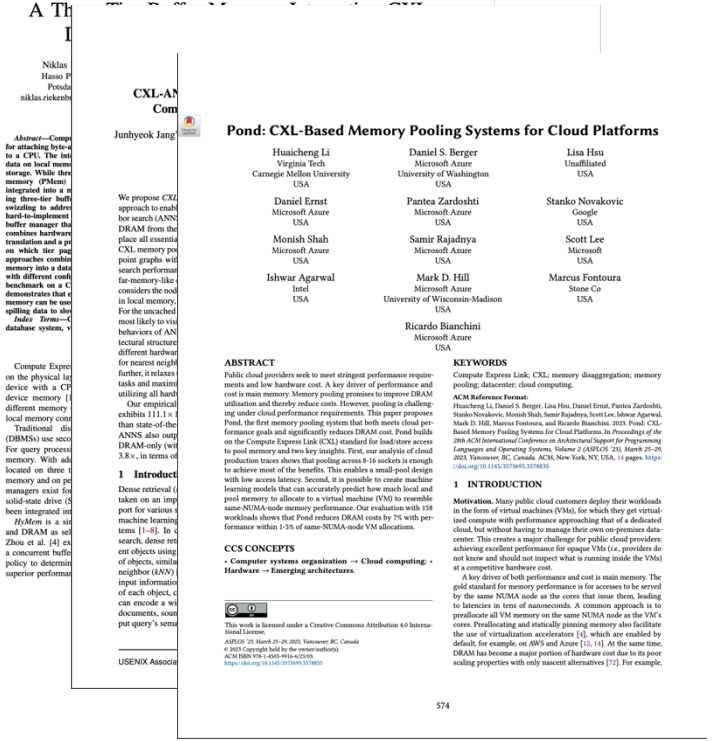
To address these memory pressure issues, modern ANNS techniques leverage lossy compression methods or employ persistent storage, such as solid state disks (SSDs) and persistent memory (PMEM), for their memory expansion. For example, [20–23] split large datasets and group them into multiple clusters in an offline time. This compression approach only has product quantized vectors for each cluster's centroid and searches kNN based on the quantized information, making billion-scale ANNS feasible. On the other hand, the hierarchical approach [24–28] accommodates the datasets to SSD/PMEM, but reduces target search spaces by referring to a summary in its local memory (DRAM). As shown in Figure 1a, these compression and hierarchical approaches can achieve the best kNN search performance and scalabil-

(a) Previous studies. (b) CXL-based approaches.

Figure 1: Various billion-scale ANNS characterizations.

CXL Scale-Up Expansion

- Expansion via tiering

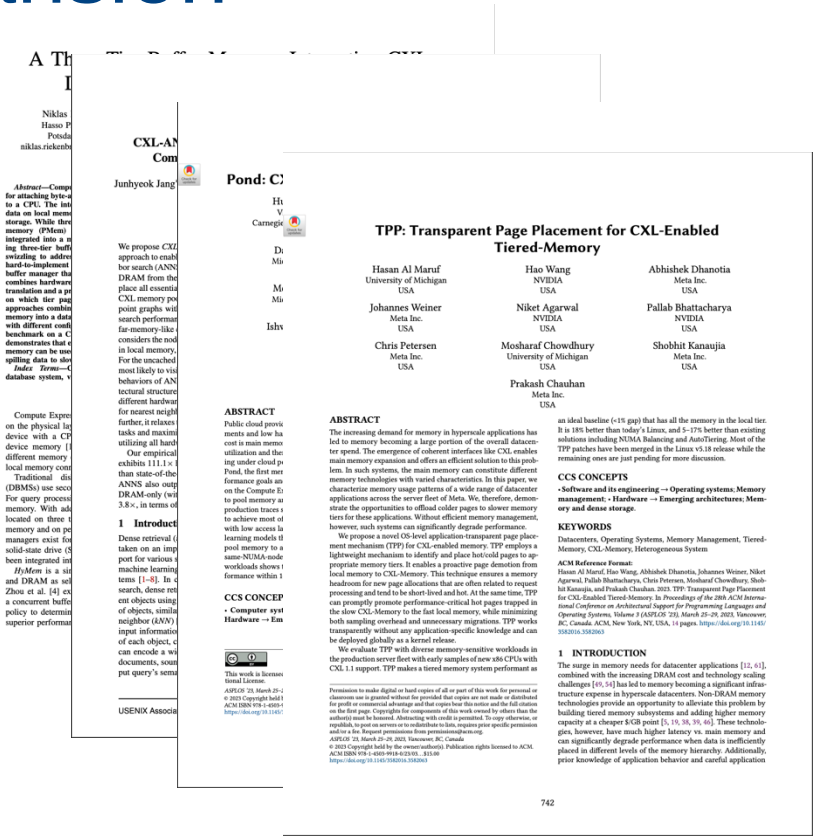


Pond: CXL-Based Memory Pooling Systems for Cloud Platforms

Huaicheng Li, Virginia Tech, Carnegie Mellon University, USA
Daniel S. Berger, Microsoft Azure, University of Washington, USA
Lisa Hsu, Unaffiliated, USA
Daniel Ernst, Microsoft Azure, USA
Pantea Zardoshti, Microsoft Azure, USA
Stanko Novakovic, Google, USA
Monish Shah, Microsoft Azure, USA
Samir Rajadnya, Microsoft Azure, USA
Scott Lee, Microsoft, USA
Ishwar Agarwal, Intel, USA
Mark D. Hill, Microsoft Azure, University of Wisconsin-Madison, USA
Marcus Fontoura, Stone Co, USA
Ricardo Bianchini, Microsoft Azure, USA

CXL Scale-Up Expansion

- Expansion via tiering



TPP: Transparent Page Placement for CXL-Enabled Tiered-Memory

Hasan Al Maruf, University of Michigan, USA
Hao Wang, NVIDIA, USA
Abhishek Dhanotia, Meta Inc., USA
Johannes Weiner, Meta Inc., USA
Niket Agarwal, NVIDIA, USA
Pallab Bhattacharya, NVIDIA, USA
Chris Petersen, Meta Inc., USA
Mosharaf Chowdhury, University of Michigan, USA
Shobhit Kanaujia, Meta Inc., USA
Prakash Chauhan, Meta Inc., USA

ABSTRACT

The increasing demand for memory in hyperscale applications has led to memory becoming a large portion of the overall datacenter spend. The emergence of coherent interfaces like CXL enables main memory expansion and offers an efficient solution to this problem. In such systems, the main memory can constitute different memory technologies with varied characteristics. In this paper, we characterize memory usage patterns of a wide range of datacenter applications across the server fleet of Meta. We, therefore, demonstrate the opportunities to offload colder pages to slower memory tiers for these applications. Without efficient memory management, however, such systems can significantly degrade performance.

We propose a novel OS-level application-transparent page placement mechanism (TPP) for CXL-enabled memory. TPP employs a lightweight mechanism to identify and place hot/cold pages to appropriate memory tiers. It enables a proactive page demotion from local memory to CXL-Memory. This technique ensures a memory headroom for new page allocations that are often related to request processing and tend to be short-lived and hot. At the same time, TPP can promptly promote performance-critical hot pages trapped in the slow CXL-Memory to the fast local memory, while minimizing both sampling overhead and unnecessary migrations. TPP works transparently without any application-specific knowledge and can be deployed globally as a kernel release.

We evaluate TPP with diverse memory-sensitive workloads in the production server fleet with early samples of new x86 CPUs with CXL 1.1 support. TPP makes a tiered memory system performant as an ideal baseline (<1% gap) that has all the memory in the local tier. It is 18% better than today's Linux, and 5–17% better than existing solutions including NUMA Balancing and AutoTiering. Most of the TPP patches have been merged in the Linux v5.18 release while the remaining ones are just pending for more discussion.

CCS CONCEPTS

• Software and its engineering → Operating systems; Memory management; • Hardware → Emerging architectures; Memory and dense storage.

KEYWORDS

Datacenters, Operating Systems, Memory Management, Tiered-Memory, CXL-Memory, Heterogeneous System

ACM Reference Format:

Hasan Al Maruf, Hao Wang, Abhishek Dhanotia, Johannes Weiner, Niket Agarwal, Pallab Bhattacharya, Chris Petersen, Mosharaf Chowdhury, Shobhit Kanaujia, and Prakash Chauhan. 2023. TPP: Transparent Page Placement for CXL-Enabled Tiered-Memory. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3 (ASPLOS '23), March 25–29, 2023, Vancouver, BC, Canada.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3582016.3582063

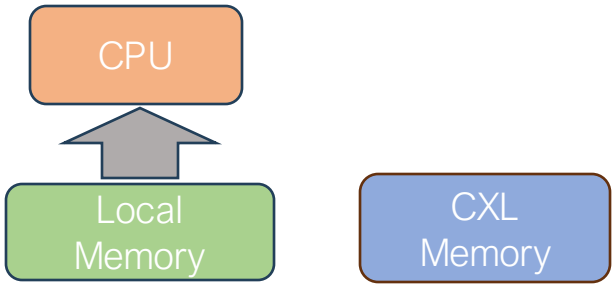
1 INTRODUCTION

The surge in memory needs for datacenter applications [12, 61], combined with the increasing DRAM cost and technology scaling challenges [49, 54] has led to memory becoming a significant infrastructure expense in hyperscale datacenters. Non-DRAM memory technologies provide an opportunity to alleviate this problem by building tiered memory subsystems and adding higher memory capacity at a cheaper $/GB point [5, 19, 38, 39, 46]. These technologies, however, have much higher latency vs. main memory and can significantly degrade performance when data is inefficiently placed in different levels of the memory hierarchy. Additionally, prior knowledge of application behavior and careful application

742

CXL Scale-Up Expansion

- Expansion via tiering




TPP: Transparent Page Placement for CXL-Enabled Tiered-Memory

Hasan Al Maruf, University of Michigan, USA
Hao Wang, NVIDIA, USA
Abhishek Dhanotia, Meta Inc., USA
Johannes Weiner, Meta Inc., USA
Niket Agarwal, NVIDIA, USA
Pallab Bhattacharya, NVIDIA, USA
Chris Petersen, Meta Inc., USA
Mosharaf Chowdhury, University of Michigan, USA
Shobhit Kanaujia, Meta Inc., USA
Prakash Chauhan, Meta Inc., USA

ABSTRACT

The increasing demand for memory in hyperscale applications has led to memory becoming a large portion of the overall datacenter spend. The emergence of coherent interfaces like CXL enables main memory expansion and offers an efficient solution to this problem. In such systems, the main memory can constitute different memory technologies with varied characteristics. In this paper, we characterize memory usage patterns of a wide range of datacenter applications across the server fleet of Meta. We, therefore, demonstrate the opportunities to offload colder pages to slower memory tiers for these applications. Without efficient memory management, however, such systems can significantly degrade performance.

We propose a novel OS-level application-transparent page placement mechanism (TPP) for CXL-enabled memory. TPP employs a lightweight mechanism to identify and place hot/cold pages to appropriate memory tiers. It enables a proactive page demotion from local memory to CXL-Memory. This technique ensures a memory headroom for new page allocations that are often related to request processing and tend to be short-lived and hot. At the same time, TPP can promptly promote performance-critical hot pages trapped in the slow CXL-Memory to the fast local memory, while minimizing both sampling overhead and unnecessary migrations. TPP works transparently without any application-specific knowledge and can be deployed globally as a kernel release.

We evaluate TPP with diverse memory-sensitive workloads in the production server fleet with early samples of new x86 CPUs with CXL 1.1 support. TPP makes a tiered memory system performant as an ideal baseline (<1% gap) that has all the memory in the local tier. It is 18% better than today's Linux, and 5–17% better than existing solutions including NUMA Balancing and AutoTiering. Most of the TPP patches have been merged in the Linux v5.18 release while the remaining ones are just pending for more discussion.

CCS CONCEPTS

• Software and its engineering → Operating systems; Memory management; • Hardware → Emerging architectures; Memory and dense storage.

KEYWORDS

Datacenters, Operating Systems, Memory Management, Tiered-Memory, CXL-Memory, Heterogeneous System

ACM Reference Format:

Hasan Al Maruf, Hao Wang, Abhishek Dhanotia, Johannes Weiner, Niket Agarwal, Pallab Bhattacharya, Chris Petersen, Mosharaf Chowdhury, Shobhit Kanaujia, and Prakash Chauhan. 2023. TPP: Transparent Page Placement for CXL-Enabled Tiered-Memory. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3 (ASPLOS '23), March 25–29, 2023, Vancouver, BC, Canada.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3582016.3582063

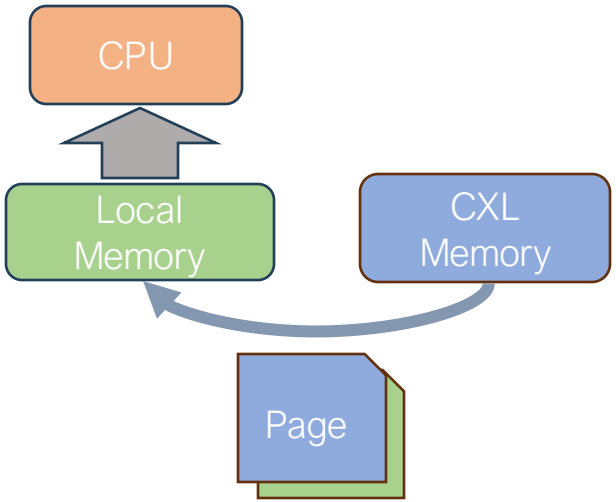
1 INTRODUCTION

The surge in memory needs for datacenter applications [12, 61], combined with the increasing DRAM cost and technology scaling challenges [49, 54] has led to memory becoming a significant infrastructure expense in hyperscale datacenters. Non-DRAM memory technologies provide an opportunity to alleviate this problem by building tiered memory subsystems and adding higher memory capacity at a cheaper $/GB point [5, 19, 38, 39, 46]. These technologies, however, have much higher latency vs. main memory and can significantly degrade performance when data is inefficiently placed in different levels of the memory hierarchy. Additionally, prior knowledge of application behavior and careful application

742

CXL Scale-Up Expansion

- Expansion via tiering

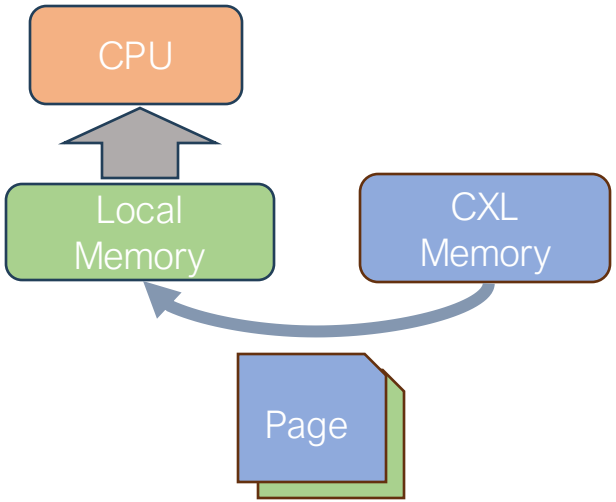


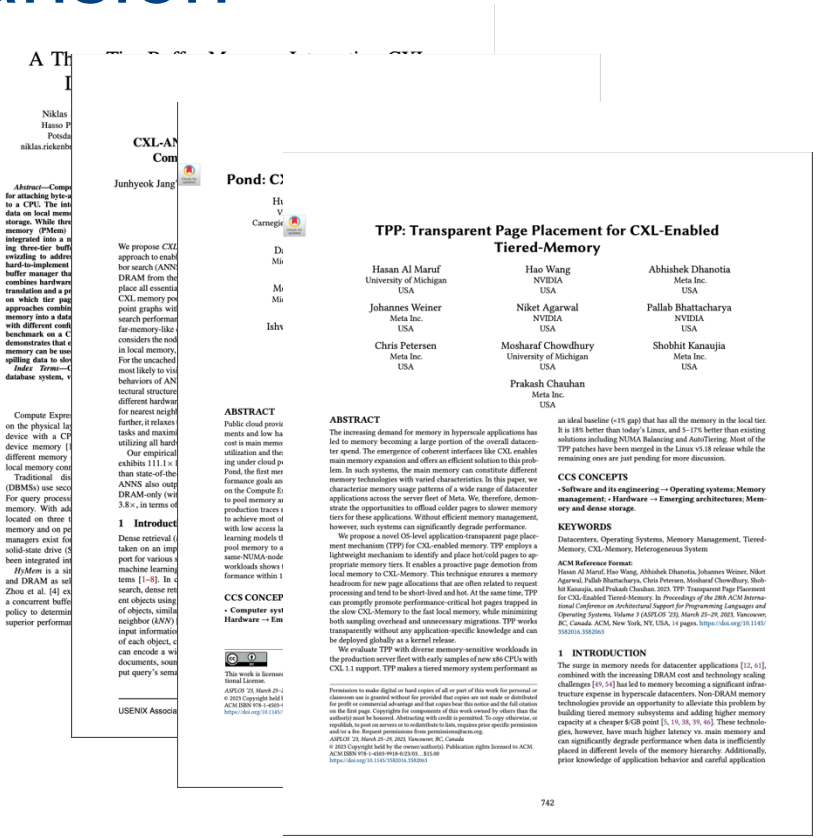

TPP: Transparent Page Placement for CXL-Enabled Tiered-Memory

Hasan Al Maruf, University of Michigan, USA
Hao Wang, NVIDIA, USA
Abhishek Dhanotia, Meta Inc., USA
Johannes Weiner, Meta Inc., USA
Niket Agarwal, NVIDIA, USA
Pallab Bhattacharya, NVIDIA, USA
Chris Petersen, Meta Inc., USA
Mosharaf Chowdhury, University of Michigan, USA
Shobhit Kanaujia, Meta Inc., USA
Prakash Chauhan, Meta Inc., USA

CXL Scale-Up Expansion

- Expansion via tiering
  - capacity expansion ✓



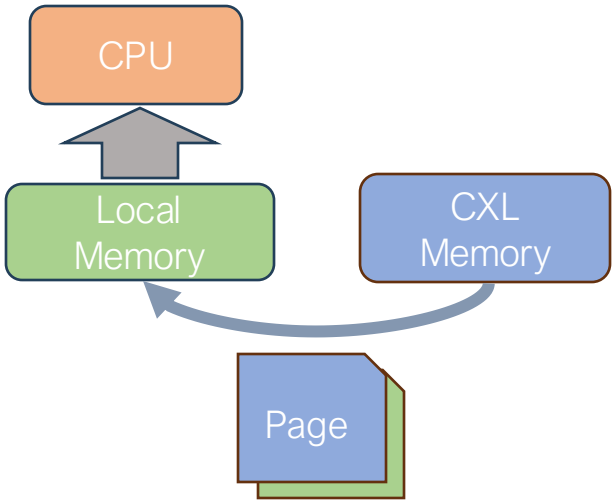


TPP: Transparent Page Placement for CXL-Enabled Tiered-Memory

Hasan Al Maruf, University of Michigan, USA
Hao Wang, NVIDIA, USA
Abhishek Dhanotia, Meta Inc., USA
Johannes Weiner, Meta Inc., USA
Niket Agarwal, NVIDIA, USA
Pallab Bhattacharya, NVIDIA, USA
Chris Petersen, Meta Inc., USA
Mosharaf Chowdhury, University of Michigan, USA
Shobhit Kanaujia, Meta Inc., USA
Prakash Chauhan, Meta Inc., USA

CXL Scale-Up Expansion

- Expansion via tiering
  - capacity expansion ✓
  - bandwidth expansion ✗

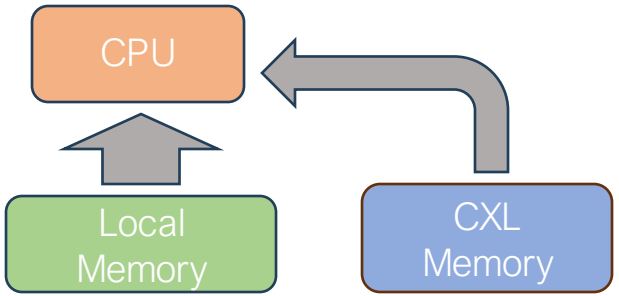


TPP: Transparent Page Placement for CXL-Enabled Tiered-Memory

Hasan Al Maruf, University of Michigan, USA
Hao Wang, NVIDIA, USA
Abhishek Dhanotia, Meta Inc., USA
Johannes Weiner, Meta Inc., USA
Niket Agarwal, NVIDIA, USA
Pallab Bhattacharya, NVIDIA, USA
Chris Petersen, Meta Inc., USA
Mosharaf Chowdhury, University of Michigan, USA
Shobhit Kanaujia, Meta Inc., USA
Prakash Chauhan, Meta Inc., USA

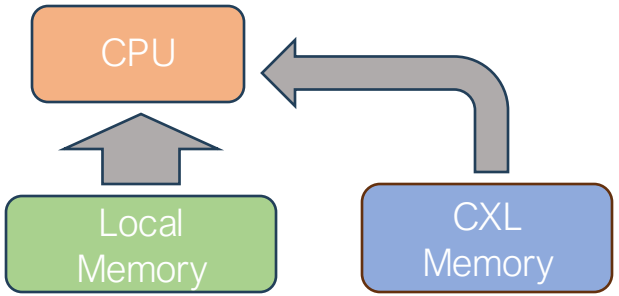
CXL Scale-Up Expansion

- Expansion within a unified tier



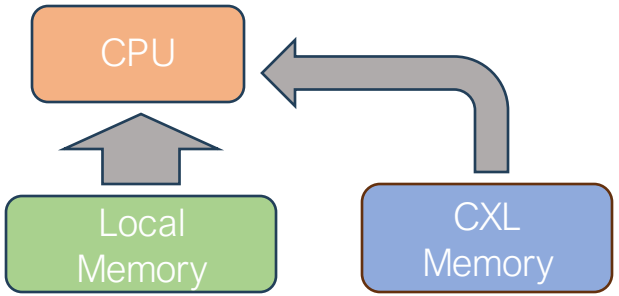
CXL Scale-Up Expansion

- Expansion within a unified tier
  - capacity expansion ✓



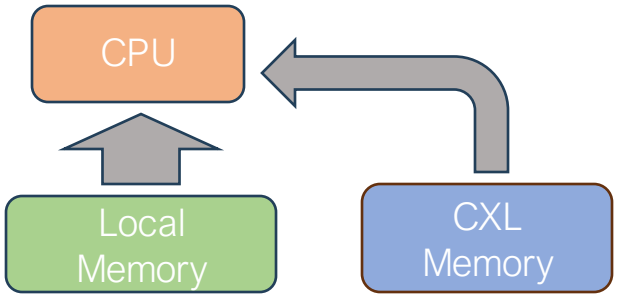
CXL Scale-Up Expansion

- Expansion within a unified tier
  - capacity expansion ✓
  - bandwidth expansion ✓
    - DDR bandwidth
    - PCIe bandwidth



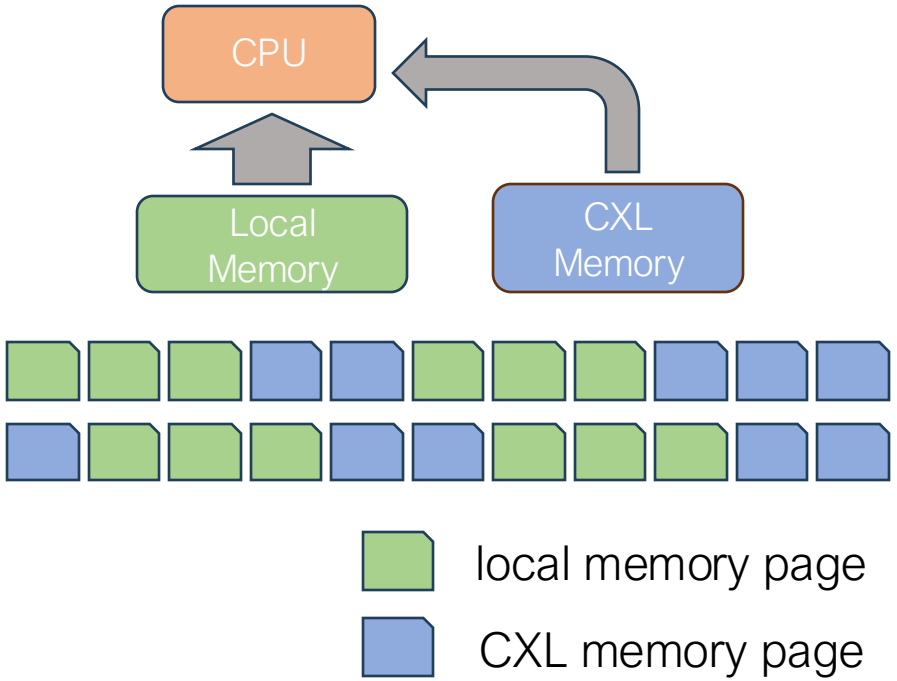
CXL Scale-Up Expansion

- Expansion within a unified tier
  - capacity expansion ✓
  - bandwidth expansion ✓
    - DDR bandwidth
    - PCIe bandwidth
- Memory interleaving



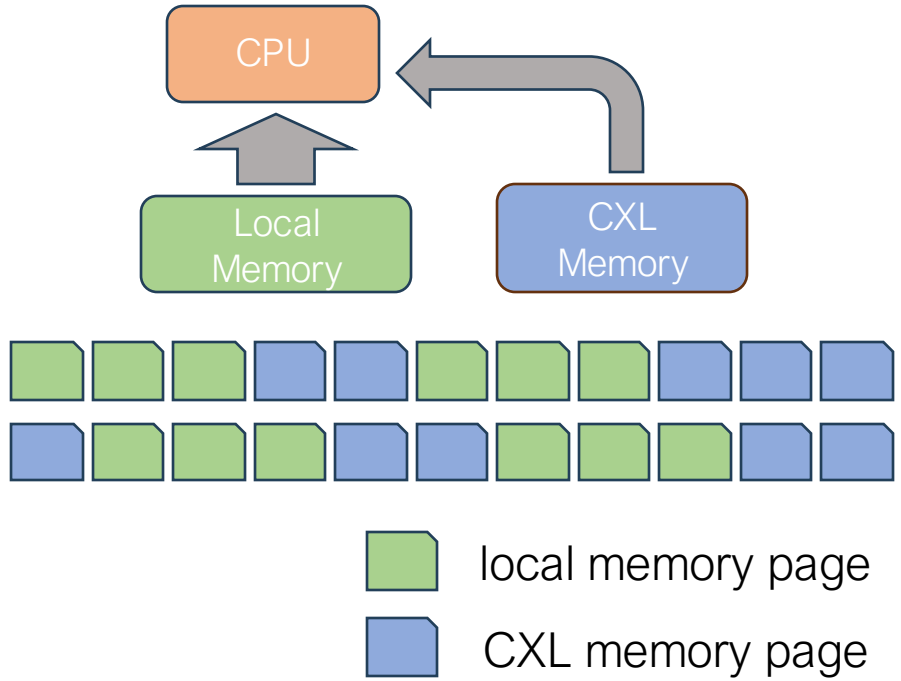
CXL Scale-Up Expansion

- Expansion within a unified tier
  - capacity expansion ✓
  - bandwidth expansion ✓
    - DDR bandwidth
    - PCIe bandwidth
- Memory interleaving



CXL Scale-Up Expansion

- Expansion within a unified tier
  - capacity expansion ✓
  - bandwidth expansion ✓
    - DDR bandwidth
    - PCIe bandwidth
- Memory interleaving
  - bandwidth-aware load balancing



Experimental Setup

- Testbed

Experimental Setup

- Testbed
  - Montage CXL type3 controller

Experimental Setup

- Testbed
  - Montage CXL type3 controller
  - Intel Sapphire Rapids CPU (32 cores)

Experimental Setup

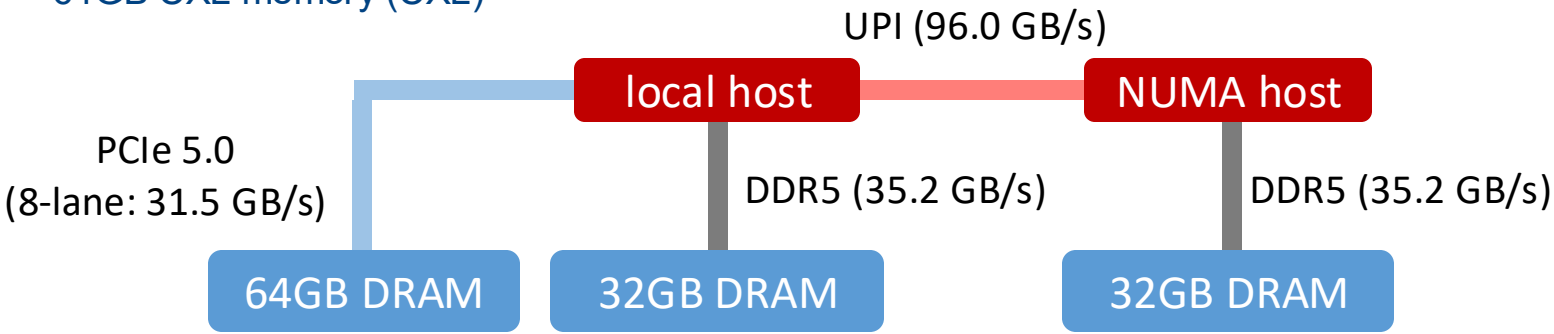
- Testbed
  - Montage CXL type3 controller
  - Intel Sapphire Rapids CPU (32 cores)
  - Sub-NUMA clustering (SNC) mode

Experimental Setup

- Testbed
  - Montage CXL type3 controller
  - Intel Sapphire Rapids CPU (32 cores)
  - Sub-NUMA clustering (SNC) mode
    - 32GB local memory (DRAM)
    - 32GB NUMA memory (NUMA)
    - 64GB CXL memory (CXL)

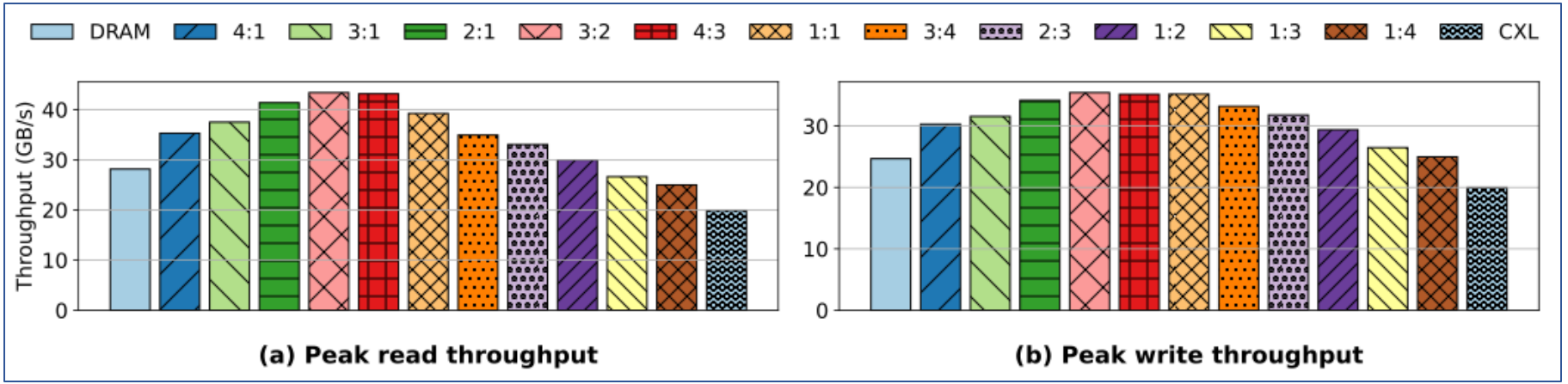
Experimental Setup

- Testbed
  - Montage CXL type3 controller
  - Intel Sapphire Rapids CPU (32 cores)
  - Sub-NUMA clustering (SNC) mode
    - 32GB local memory (DRAM)
    - 32GB NUMA memory (NUMA)
    - 64GB CXL memory (CXL)



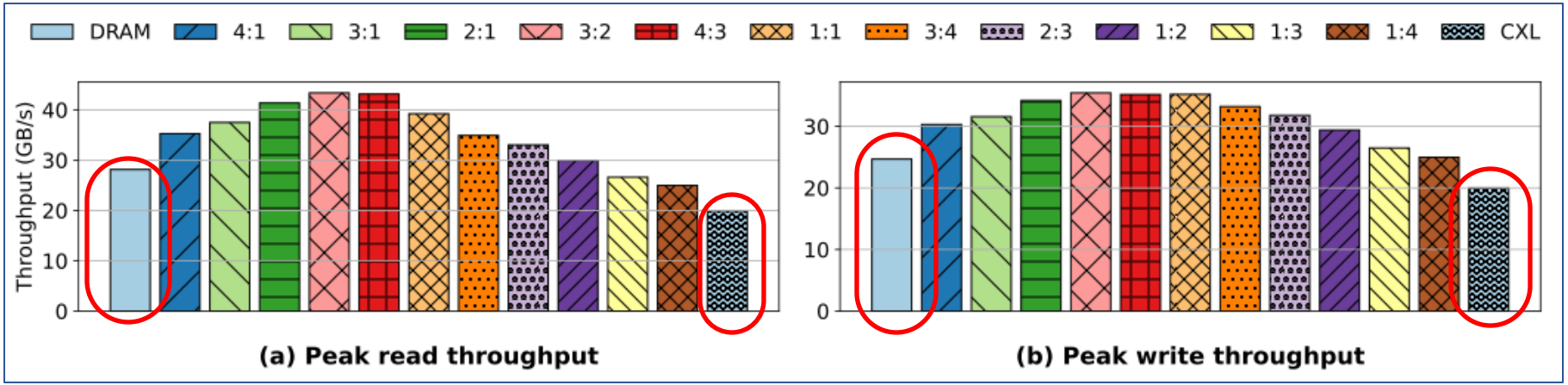
Throughput w.r.t. Memory Interleaving Ratio

- Experiment workload
  - sequential access over 2 billion records (8-byte) with 32 cores.



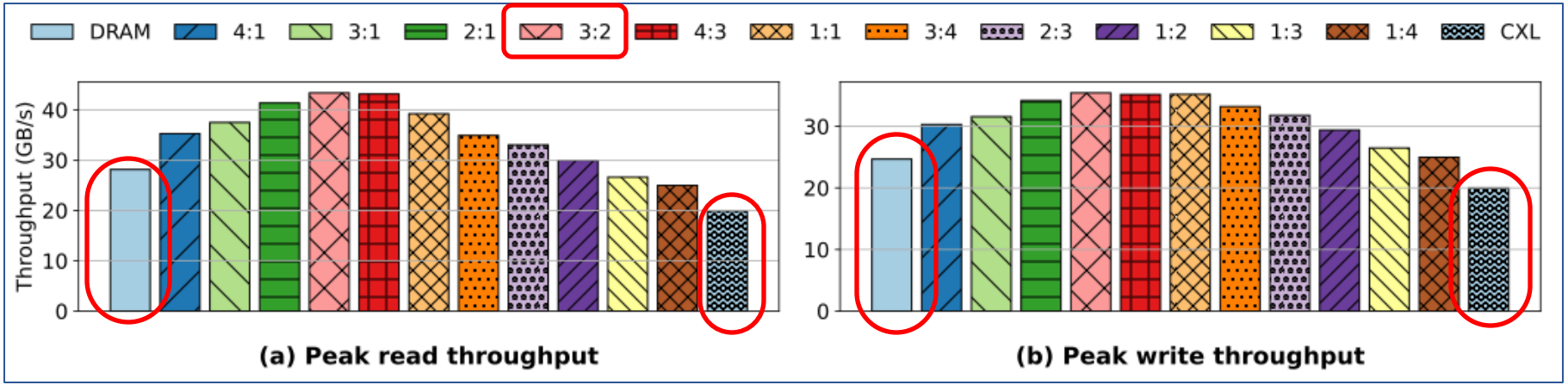
Throughput w.r.t. Memory Interleaving Ratio

- Experiment workload
  - sequential access over 2 billion records (8-byte) with 32 cores.



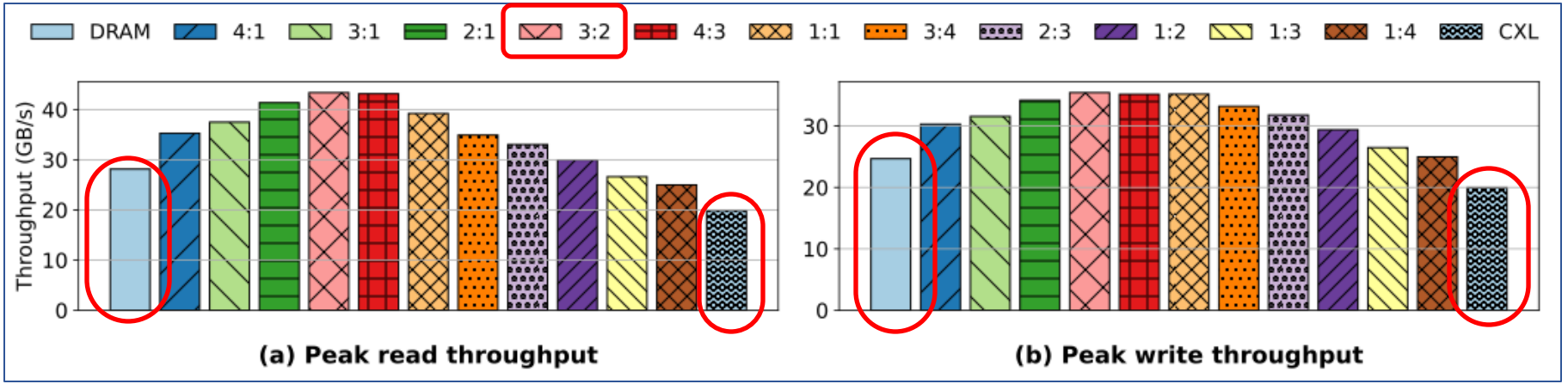
Throughput w.r.t. Memory Interleaving Ratio

- Experiment workload
  - sequential access over 2 billion records (8-byte) with 32 cores.



Throughput w.r.t. Memory Interleaving Ratio

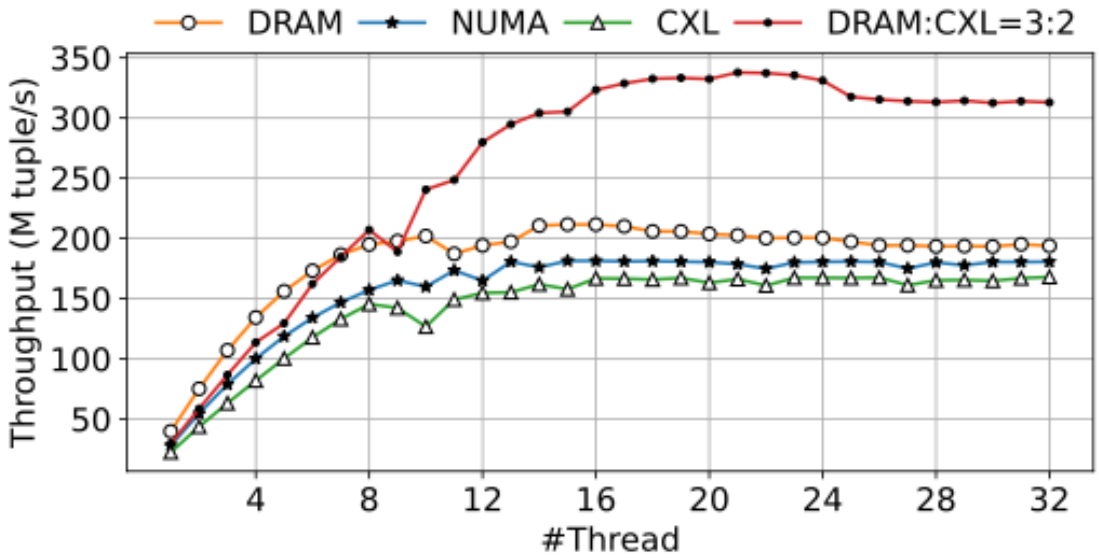
- Experiment workload
  - sequential access over 2 billion records (8-byte) with 32 cores.



1.61x throughput gain

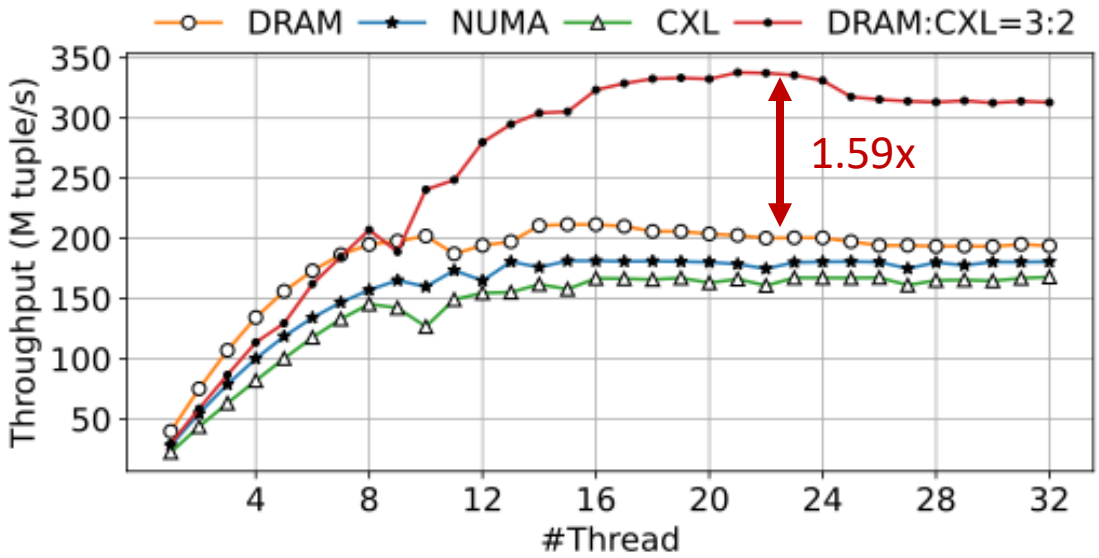
Hash Join Evaluation

- Experiment workload
  - uniform distribution, 256M ⋈ 1024M (8-byte)



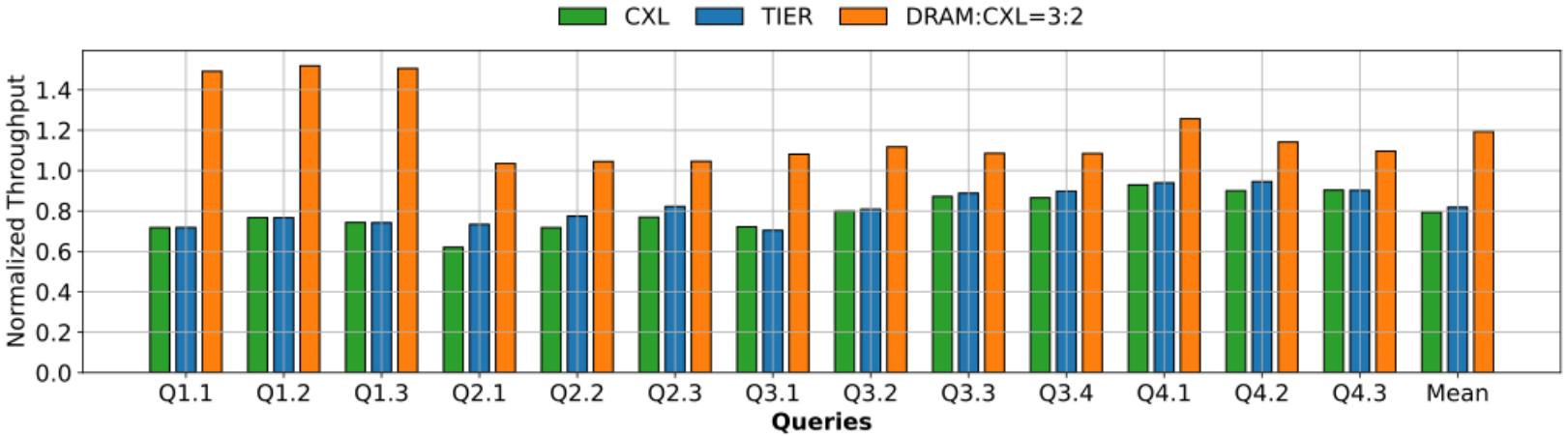
Hash Join Evaluation

- Experiment workload
  - uniform distribution, 256M ⋈ 1024M (8-byte)



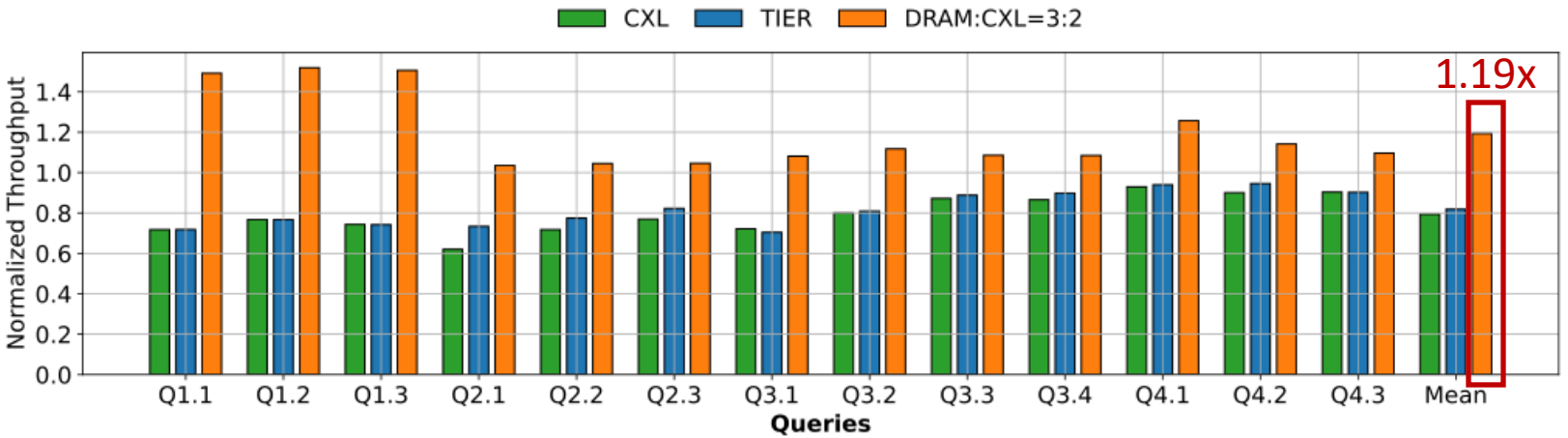
Star Schema Benchmark

- SSB workload
  - column store
  - scaling factor: 30



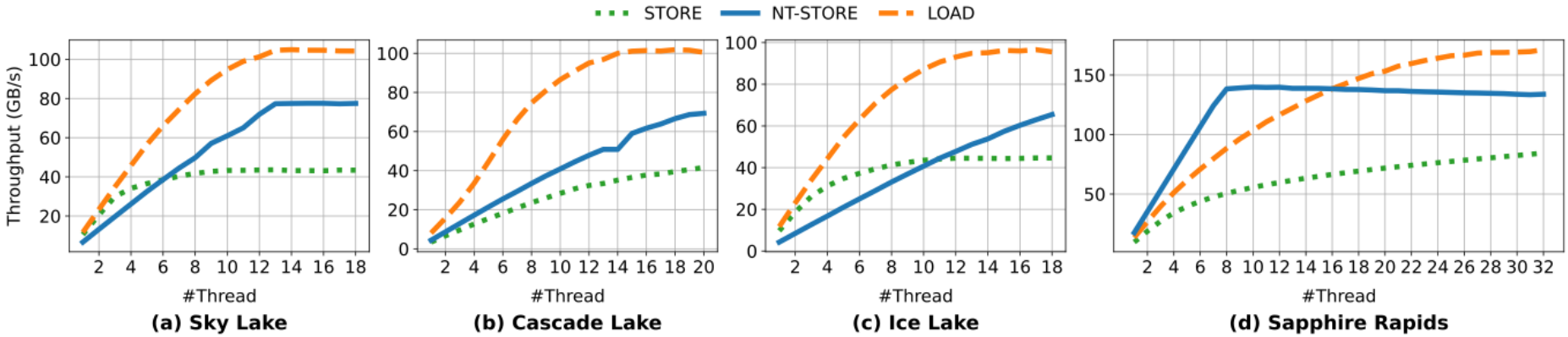
Star Schema Benchmark

- SSB workload
  - column store
  - scaling factor: 30



Limited Bandwidth in Socket

- Experiment workload
  - sequential access over 2 billion records (8-byte)



Summary

Summary

- CXL scale-up solution
  - capacity expansion
  - bandwidth expansion

Summary

- CXL scale-up solution
  - capacity expansion
  - bandwidth expansion

- Perspectives
  - memory interleaving tuning
  - fine-grained data placement
  - bandwidth-aware load balancing

Summary

- CXL scale-up solution
  - capacity expansion
  - bandwidth expansion
- Perspectives
  - memory interleaving tuning
  - fine-grained data placement
  - bandwidth-aware load balancing

GitHub Link

THANK YOU