

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
 -

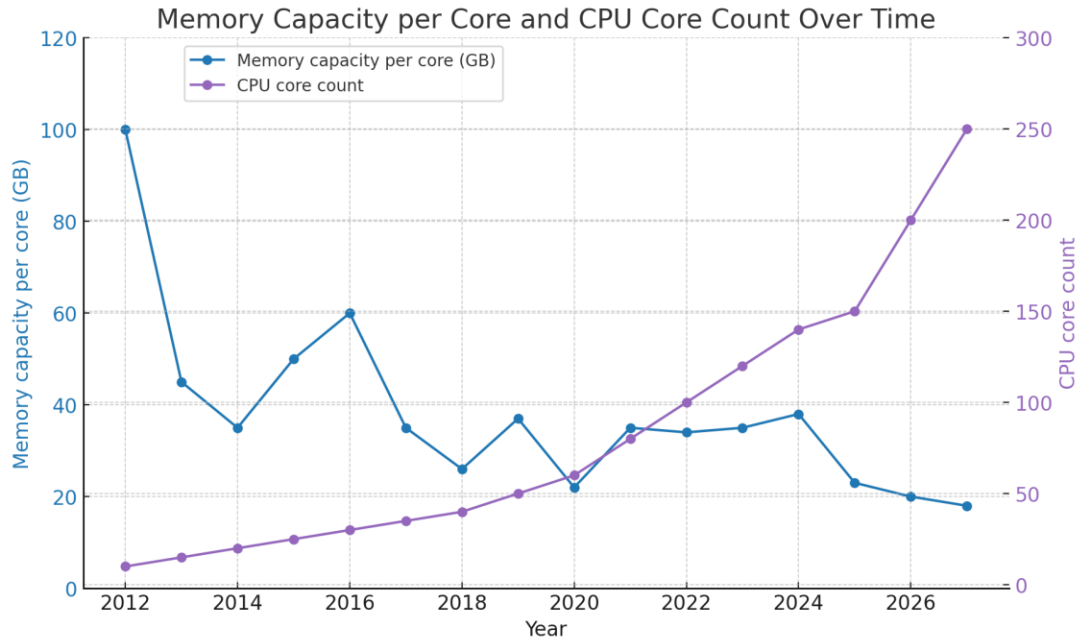


Growing Memory Demands

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

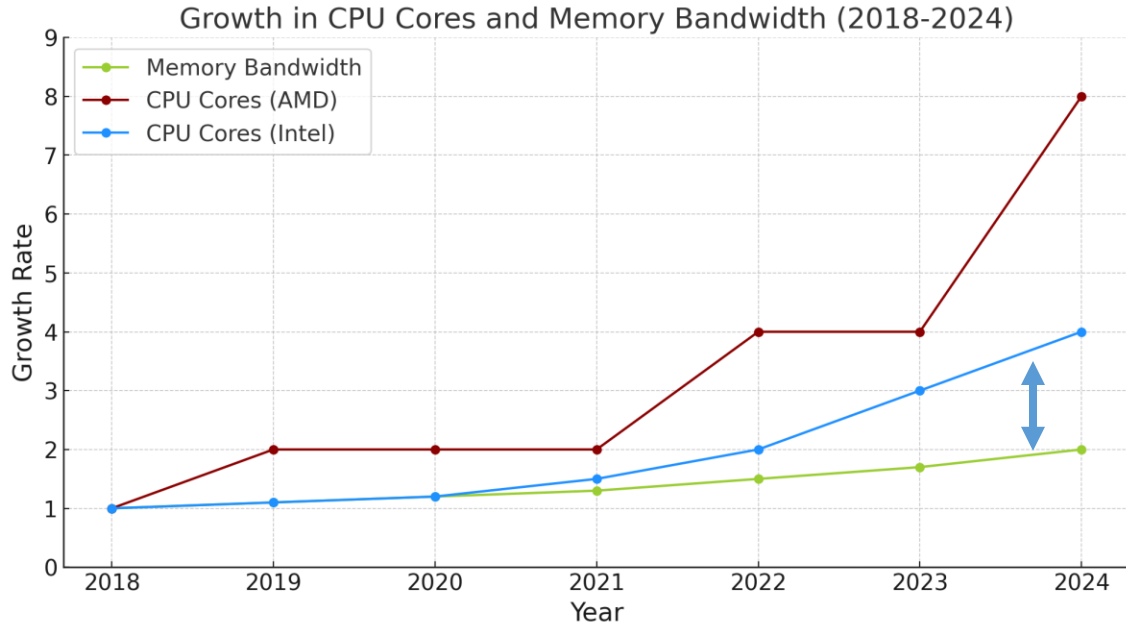
Hitting the Memory Wall

- Memory capacity



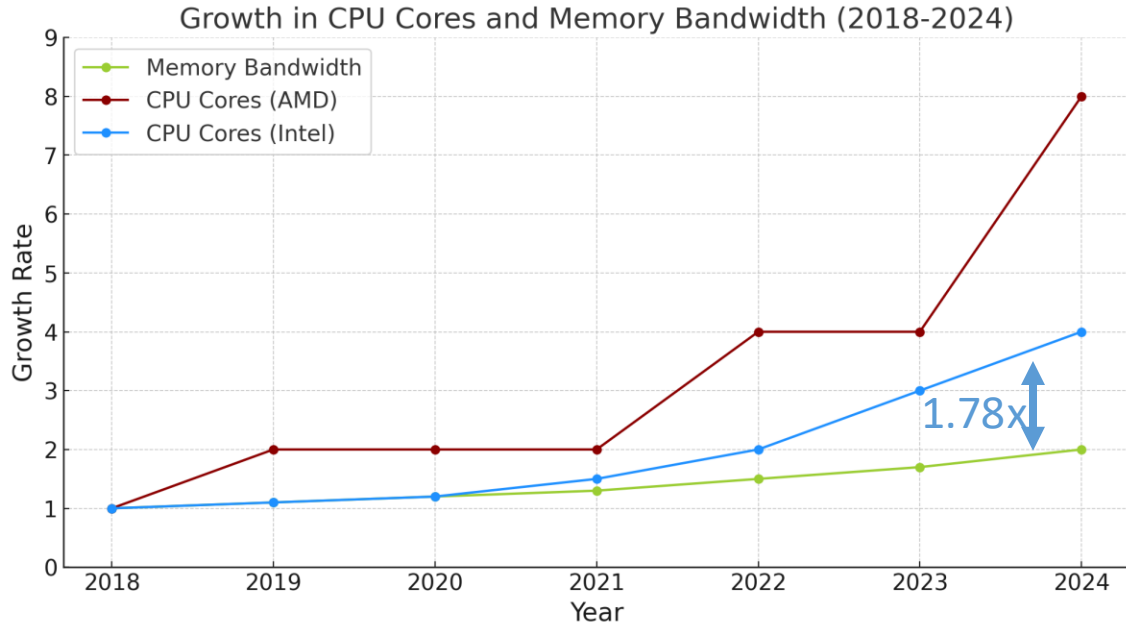
Hitting the Memory Wall

- Memory bandwidth



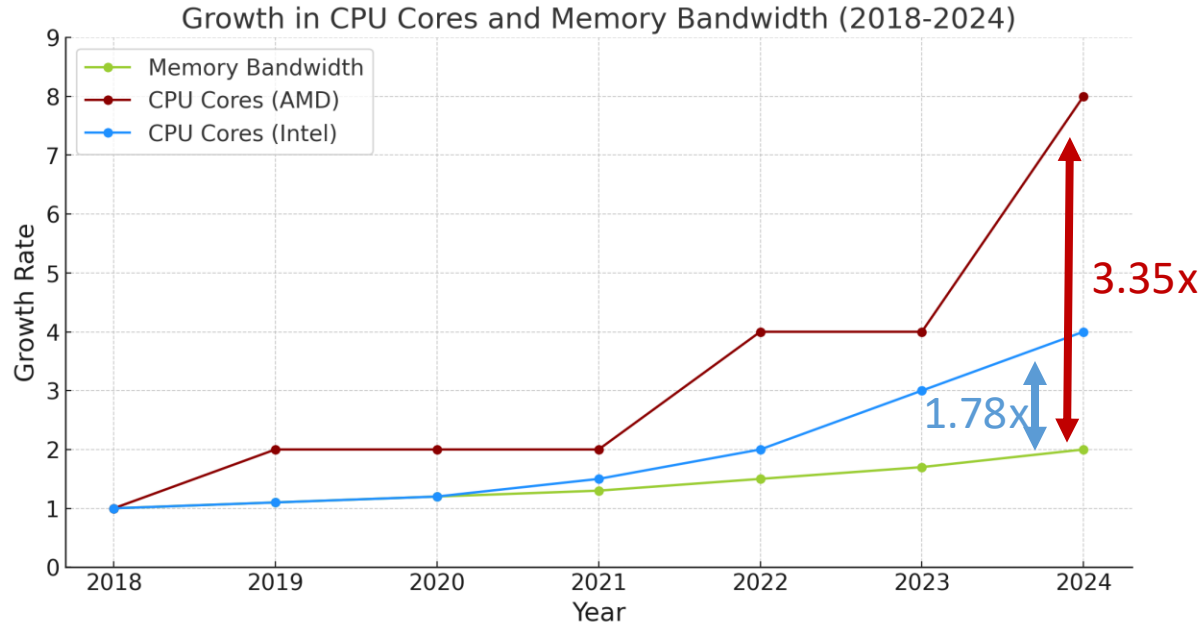
Hitting the Memory Wall

- 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: accelerators w/o memory.

Usage:

- PGAS NIC
- NIC atomics

Protocol:

- CXL.cache
- CXL.io



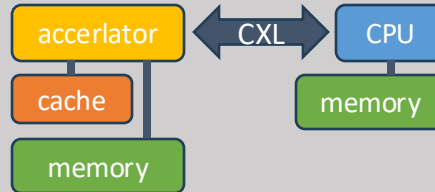
Type 2: accelerators w/ memory.

Usage:

- GPU
- FPGA

Protocol:

- CXL.cache
- CXL.mem
- CXL.io



CXL End-Point Devices

Type 1: accelerators w/o memory.

Usage:

- PGAS NIC
- NIC atomics

Protocol:

- CXL.cache
- CXL.io



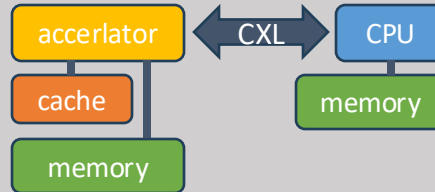
Type 2: accelerators w/ memory.

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



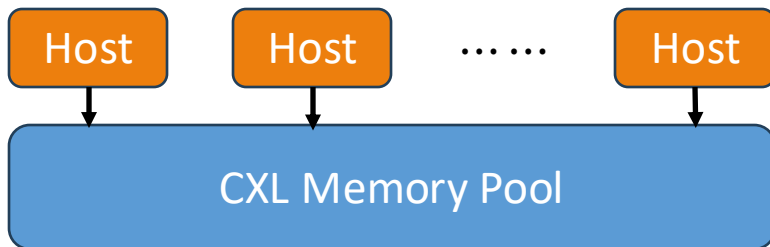
Scale-Out vs. Scale-Up

- **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



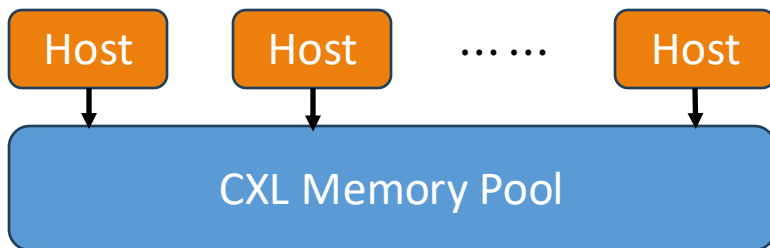
Scale-Out vs. Scale-Up

- **Scale-out**

- CXL 2.0/3.0
- resource pooling
- economical cost

- **Scale-up**

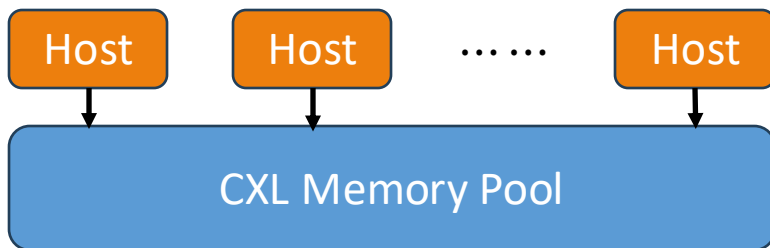
- CXL 1.1
- capacity expansion
- bandwidth expansion
- lower TCO



Scale-Out vs. Scale-Up

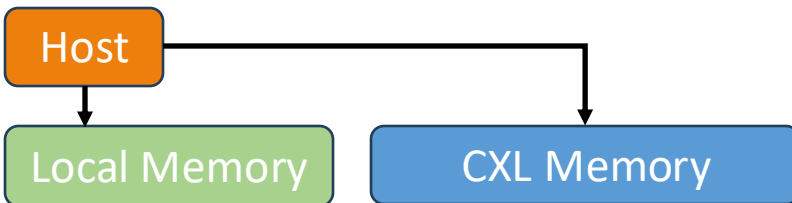
- **Scale-out**

- 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 Rickenbrauck
Hasso Plattner Institute
Potsdam, Germany
niklas.rickenbrauck@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]. Accesses over CXL exhibit different memory characteristics, such as higher latency than local memory connected via Double Data Rate (DDR) [1], [2]. Traditional disk-based database management systems (DBMS) 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 *Spiffre*, 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 *Spiffre* 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).
- 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.

¹Source code: <https://github.com/hyrise/hyrise/tree/master/buffermanager>

CXL Scale-Up Expansion

- Expansion via tiering

A Th
Niklas
Hasso P
Potsda
niklas.rickert@

Abstract—Comp
for attaching byte-
to a CPU. The lat
data on local mem
storage. While the
memory (PMEM)
integrated into a
ing three-tier buff
evaluating its adre
hard-to-implement
buffer manager th
combines hardware
translation and p
on which tier pag
approaches combi
memory into a dat
with different cost
benchmark on a C
demonstrates that
memory can be use
spilling data to slo
Index Terms—C
database system, v

Compute Engin
on the physical la
device with a CP
device memory [1
different memory
local memory con
Traditional data
(DBMS) use seco
For query proces
memory. With ad
located on three t
memory and on p
managers exist fo
solid-state drive (S
been integrated in
HBM is a sil
and DRAM as se
Zhou et al. [4] es
a concurrent buffe
policy to determin
superior perform

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

Junhyeok Jang¹, Hanjin Choi^{1*}, Hanyeoreum Bae¹, Seungjun Lee², Miryeong Kwon¹, Myoungsoo Jung^{1*}
¹Computer Architecture and Memory Systems Laboratory, KAIST
²Pamnesia, 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 11.1% higher QPS with 35.3% 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/matrix-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 (ANN) [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



Figure 1: Various billion-scale ANNS characterizations.

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 1, these compression and hierarchical approaches can achieve the best KNN search performance and scalabil-

CXL Scale-Up Expansion

- Expansion via tiering

A Th

Niklas
Hasso P
Potsda
niklas.rickert

Abstract—Comp
for attaching by-
to a CPU. The lat
data on local mem
storage. While the
memory (PMem)
integrated into a
ing three-tier buff
evolving to addre
hard-to-implement
buffer manager th
combine hardware
translation and p
on which tier po
approach combin
memory into a dat
with different cost
benchmark on a C
demonstrates that
memory can be us
spilling data to slo
Index Terms—C
database system, v

Compute Expre
on the physical la
device with a CP
device memory [1
different memory
local memory con
Traditional dis
(DBMSs) use seco
For query process
memory. With ad
located on three t
memory and on p
managers exist fo
solid-state drive (S
been integrated in
HAdem is a sil
and DRAM as sel
Zhao et al. [4] ex
a concurrent buffe
policy to determin
superior perform

CXL-AN
Com

Junhyeok Jang

Huaicheng Li
Virginia Tech
Carnegie Mellon University
USA

Daniel Ernst
Microsoft Azure
USA

Manish Shah
Microsoft Azure
USA

Ishwar Agarwal
Intel
USA

Daniel S. Berger
Microsoft Azure
University of Washington
USA

Pantea Zardoshti
Microsoft Azure
USA

Samir Rajadnya
Microsoft Azure
USA

Mark D. Hill
Microsoft Azure
University of Wisconsin-Madison
USA

Ricardo Bianchini
Microsoft Azure
USA

Lisa Hsu
Unaffiliated
USA

Stanko Novakovic
Google
USA

Scott Lee
Microsoft
USA

Marcus Fontoura
Stone Co
USA

Compute Express Link (CXL), memory disaggregation, memory pooling, database, cloud computing.

ACM Performance Forum
Huaicheng Li, Daniel S. Berger, Lisa Hsu, Daniel Ernst, Pantea Zardoshti, Stanko Novakovic, Manish Shah, Samir Rajadnya, Scott Lee, Ishwar Agarwal, Mark D. Hill, Marcus Fontoura, and Ricardo Bianchini. 2023. Pond: CXL-Based Memory Pooling Systems for Cloud Platforms. In *Proceedings of the 20th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '23)*. March 28–30, 2023, Vancouver, BC, Canada. ACM, New York, NY, USA, 11 pages. <https://doi.org/10.1145/3576893.3576935>

1 INTRODUCTION
Motivation. Many public cloud customers deploy their workloads in the form of virtual machines (VMs), for which they get virtualized compute with performance approaching that of a dedicated cloud, but without having to manage their own on-premises datacenter. This creates a major challenge for public cloud providers: achieving excellent performance for opaque VMs (i.e., providers do not know and should not impact what is running inside the VMs) at a competitive hardware cost.
A key driver of both performance and cost is main memory. The gold standard for memory performance is for accesses to be served by the same NIMMA node as the cores that issue them, leading to latencies in the order of nanoseconds. A common approach is to preallocate all VM memory on the same NIMMA node as the VM's cores. Preallocating and statically pinning memory also facilitates the use of virtualization accelerators [5], which are enabled by default, for example, on AWS S and Azure [12, 14]. At the same time, DRAM has become a major portion of hardware cost due to its poor scaling properties with only scant alternatives [7]. For example,

Compute Express Link (CXL), memory disaggregation, memory pooling, database, cloud computing.

ACM Performance Forum
Huaicheng Li, Daniel S. Berger, Lisa Hsu, Daniel Ernst, Pantea Zardoshti, Stanko Novakovic, Manish Shah, Samir Rajadnya, Scott Lee, Ishwar Agarwal, Mark D. Hill, Marcus Fontoura, and Ricardo Bianchini. 2023. Pond: CXL-Based Memory Pooling Systems for Cloud Platforms. In *Proceedings of the 20th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '23)*. March 28–30, 2023, Vancouver, BC, Canada. ACM, New York, NY, USA, 11 pages. <https://doi.org/10.1145/3576893.3576935>

1 INTRODUCTION
Motivation. Many public cloud customers deploy their workloads in the form of virtual machines (VMs), for which they get virtualized compute with performance approaching that of a dedicated cloud, but without having to manage their own on-premises datacenter. This creates a major challenge for public cloud providers: achieving excellent performance for opaque VMs (i.e., providers do not know and should not impact what is running inside the VMs) at a competitive hardware cost.
A key driver of both performance and cost is main memory. The gold standard for memory performance is for accesses to be served by the same NIMMA node as the cores that issue them, leading to latencies in the order of nanoseconds. A common approach is to preallocate all VM memory on the same NIMMA node as the VM's cores. Preallocating and statically pinning memory also facilitates the use of virtualization accelerators [5], which are enabled by default, for example, on AWS S and Azure [12, 14]. At the same time, DRAM has become a major portion of hardware cost due to its poor scaling properties with only scant alternatives [7]. For example,

Compute Express Link (CXL), memory disaggregation, memory pooling, database, cloud computing.

ACM Performance Forum
Huaicheng Li, Daniel S. Berger, Lisa Hsu, Daniel Ernst, Pantea Zardoshti, Stanko Novakovic, Manish Shah, Samir Rajadnya, Scott Lee, Ishwar Agarwal, Mark D. Hill, Marcus Fontoura, and Ricardo Bianchini. 2023. Pond: CXL-Based Memory Pooling Systems for Cloud Platforms. In *Proceedings of the 20th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '23)*. March 28–30, 2023, Vancouver, BC, Canada. ACM, New York, NY, USA, 11 pages. <https://doi.org/10.1145/3576893.3576935>

CXL Scale-Up Expansion

- Expansion via tiering

Niklas
Hasso P
Potsda
niklas.rickert@

Abstract—Comp for attaching byte- to a CPU. The lat data on local memory (PMem) integrated into a n ing three-tier buff evading to addre hard-to-implement buffer manager the combine hardware translation and p on which tier out approaches combi memory into a dat with different cost benchmark on a C demonstrates that e memory can be not spilling data to slo **Index Terms**—C database system, v

Compute Expre on the physical la device with a CP dense memory [1 different memory local memory con Traditional di (DBMS) use seco For query proces memory. With ad located on three t memory and on pe managers exist fo solid-state drive (S been integrated in HMem is a sil and DRAM as sel Zhou et al. [4] e concurrent buffi policy to determin superior perform

A Th
Junhyeok Jang

CXL-AN
Com

We propose CXL approach to enable be search (ANN) DRAM from the place all essential CXL memory po point graphs wit search performe far-memory-like considers the nod in local memory. For the uncached most likely to visi behaviors of AN textural structure different hardware for nearest neigh

Further, it relaxe tasks and maxim utilizes all hard) Our empirical exhibit 11.1% t than state-of-the- ANNS also outp DRAM-only (wit 3.8 s., in terms of

1 Introduction
Dense retrieval (taken on an imp port for various machine learning tems [1–8]. In c recent object using objects, similar neighbor (ANN) input information of each object, c can encode a wi documents, souc put query's sem

USENIX Associa

Pond: C:

H
V
Camrige
D
M
M
M
Ish

ABSTRACT
Public cloud provi ments and low h cost is main memo utilization and the ing under cloud p- Pond, the first me performance goals an on the Compute E to pool memory as production trace s to achieve most of with low access la learning model it pool memory to a same-NUMA node workload shows 1 formance within 1

CCS CONCEPTS
Computer system Hardware — Em

This work is licensed under Creative Commons License.

ARXIV: 23, March 20, 2024
© 2024 Copyright held by ACM ISBN 978-1-60558-140-1
https://doi.org/10.1145/3610957

ARXIV: 23, March 20, 2024, Vancouver, BC, Canada
© 2024 Copyright held by the owner/authors. Publication rights licensed to ACM.
ACM ISBN 978-1-60558-140-1, \$3.00
https://doi.org/10.1145/3610957

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

Hasan Al Maruf
University of Michigan
USA
Johannes Weiner
Meta Inc.
USA
Chris Petersen
Meta Inc.
USA

Hao Wang
NVIDIA
USA
Niket Agarwal
NVIDIA
USA
Mosharaf Chowdhury
University of Michigan
USA
Prakash Chaudhan
Meta Inc.
USA

Abhishek Dhanotia
Meta Inc.
USA
Pallab Bhattacharya
NVIDIA
USA
Shobhit Kanaujia
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 the 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 support. TPP makes a tiered memory system perform as

an ideal baseline (+15 pgs) that has all the memory in the local tier. It is 18% better than today's Linux, and 5.7% 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.

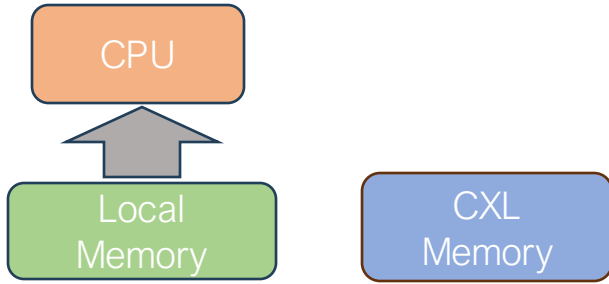
KEYWORDS
Datacenters; Operating Systems; Memory Management; Tiered-Memory; CXL-Memory; Heterogeneous System

ACM Reference Format:
Hasan Al Maruf, Hao Wang, Abhishek Dhanotia, Johannes Weiner, Niket Agarwal, Pallab Bhattacharya, Chris Petersen, Mosharaf Chowdhury, Shobhit Kanaujia, and Prakash Chaudhan. 2024. TPP: Transparent Page Placement for CXL-Enabled Tiered-Memory. In *Proceedings of the 20th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, Volume 1 (ASPLOS '24), March 25–28, 2024, Vancouver, BC, Canada. ACM, New York, NY, USA, 10 pages. <https://doi.org/10.1145/3620164.3620163>

1 INTRODUCTION
The surge in memory needs for datacenter applications [12, 61], combined with the increasing DRAM cost and technology scaling challenges [15, 64] has led to memory becoming a significant infrastructure expense in hyperscale datacenters. Non-DRAM memory technologies provide an opportunity to alleviate this problem by holding tiered memory subsystems and adding higher memory capacity at a cheaper \$GB point [5, 15, 35, 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

CXL Scale-Up Expansion

- Expansion via tiering



Niklas
Hasso P
Potsda
niklas.rickerts

Abstract—Comp for attaching byte- to a CPU. The lat data on local mem storage. While the memory (PMem) integ into a n ing three-tier buff evading to addre hard-to-implement buffer manager the combine hardware translation and p on which tier out approaches combi memory into a dat with different cod benchmark on a C demonstrates that e memory can be ut spilling data to slo **Index Terms**—C database system, v

Compute Expre on the physical la device with a CP dense memory [1 different memory local memory con Traditional di (DBMS) use seco For query proces memory. With ad located on three t memory and on p managers exist fo solid-state drive (S been integrat in HMem is a sil and DRAM as sel Zhou et al. [4] es a concurrent buffi policy to determin superior perform

Junhyeok Jang

Pond: C:
H
V
Camrige

Abstract
Public cloud provi ments and low h cost is main memo utilization and the ing under cloud p- Pond, the first meo formance goals an on the Compute E to pool memory as production trace s to achieve most of with low access la learning modifi pool memory to a same-NUMA deveo workloads show 1 formance within 1

CCS CONCEPTS
• Computer system Hardware — Em

This work is licensed under a Creative Commons Attribution 4.0 International License.
ARXIV: 23. March 20-2
© 2023 Copyright held
ACM ISBN 978-1-60959-100-7
https://doi.org/10.1145/3582563

USENIX Associa

Pond: C:
H
V
Camrige

Abstract
Public cloud provi ments and low h cost is main memo utilization and the ing under cloud p- Pond, the first meo formance goals an on the Compute E to pool memory as production trace s to achieve most of with low access la learning modifi pool memory to a same-NUMA deveo workloads show 1 formance within 1

CCS CONCEPTS
• Computer system Hardware — Em

This work is licensed under a Creative Commons Attribution 4.0 International License.
ARXIV: 23. March 20-2
© 2023 Copyright held
ACM ISBN 978-1-60959-100-7
https://doi.org/10.1145/3582563

USENIX Associa

Pond: C:
H
V
Camrige

Abstract
Public cloud provi ments and low h cost is main memo utilization and the ing under cloud p- Pond, the first meo formance goals an on the Compute E to pool memory as production trace s to achieve most of with low access la learning modifi pool memory to a same-NUMA deveo workloads show 1 formance within 1

CCS CONCEPTS
• Computer system Hardware — Em

This work is licensed under a Creative Commons Attribution 4.0 International License.
ARXIV: 23. March 20-2
© 2023 Copyright held
ACM ISBN 978-1-60959-100-7
https://doi.org/10.1145/3582563

USENIX Associa

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 Chaudhan 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 interconnects 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 support. TPP makes a tiered memory system perform as

an ideal baseline (+15 pgs) that has all the memory in the local tier. It is 18% better than Intel's Lmem, 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.

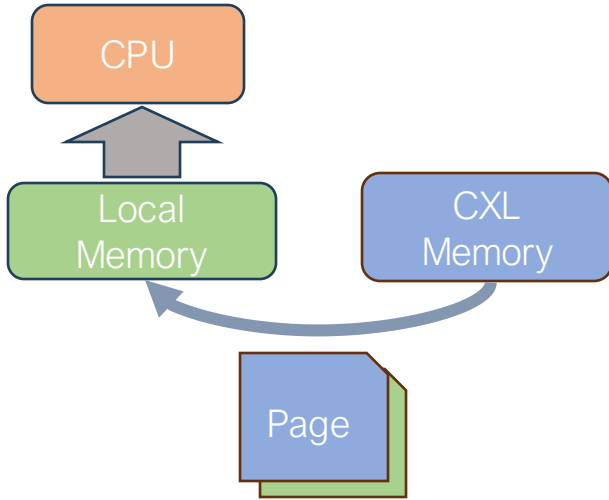
KEYWORDS
Datacenters, Operating Systems, Memory Management, Tiered-Memory, CXL-Memory, Hypervisor, Hypervisor System

ACM Reference Format:
Hasan Al Maruf, Hao Wang, Abhishek Dhanotia, Johannes Weiner, Niket Agarwal, Pallab Bhattacharya, Chris Petersen, Mosharaf Chowdhury, Shobhit Kanaujia, and Prakash Chaudhan. 2023. TPP: Transparent Page Placement for CXL-Enabled Tiered-Memory. In *Proceedings of the 20th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, Volume 7 (ASPLOS '23), March 25–28, 2023, Vancouver, BC, Canada. ACM, New York, NY, USA, 10 pages. <https://doi.org/10.1145/3582563>

1 INTRODUCTION
The surge in memory needs for datacenter applications [12, 61], combined with the increasing DRAM cost and technology scaling challenges [15, 64] has led to memory becoming a significant infrastructure expense in hyperscale datacenters. Non-DRAM memory technologies provide an opportunity to alleviate this problem by holding tiered memory subsystems and adding higher memory capacity at a cheaper \$GB point [5, 15, 35, 35, 66]. 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

CXL Scale-Up Expansion

- Expansion via tiering
 - capacity expansion ✓



Niklas
Hasso P
Potsda
niklas.rickert@

Abstract—Comp
for attaching by-t
to a CPU. The lat
data on local mem
storage. While the
memory (PMem)
integrated into a
ing three-tier buff
evaluating to add
hard-to-implement
buffer manager th
combine hardware
translation and p
on which tier pag
approaches combin
memory into a dat
with different cod
benchmark on a C
demonstrates that
memory can be use
spilling data to slo
Index Terms—C
database system, v

Compute Expre
on the physical lat
device with a CP
device memory [1
different memory
local memory con
Traditional data
than state-of-the
ANNS also outp
DRAM-only (wit
3.8 s, in terms of
1 **Introduc**
Dense retrieval (t
taken on an imp
port for various
machine learning
tasks [1–8]. In c
Zhou et al. [4] se
and a concurrent
objects using obj
objects, similar
neighborhood (ANN)
input information
of each object, c
can encode a set
documents, sou
put query's sem

Junhyeok Jang

**CXL-AN
Com**

We propose CXL
approach to enable
for search (ANN)
DRAM from the
place all essenti
CXL memory po
point graphs wit
search performan
far-memory-like
considers the no
in local memory).
For the uncached
most likely to vis
behaviors of AN
textural structure
different hardware
for nearest neigh

Compute Expre
on the physical lat
device with a CP
device memory [1
different memory
local memory con
Traditional data
than state-of-the
ANNS also outp
DRAM-only (wit
3.8 s, in terms of
1 **Introduc**
Dense retrieval (t
taken on an imp
port for various
machine learning
tasks [1–8]. In c
Zhou et al. [4] se
and a concurrent
objects using obj
objects, similar
neighborhood (ANN)
input information
of each object, c
can encode a set
documents, sou
put query's sem

USENIX Associa

Pond: C:

H
V
Cambridge

ABSTRACT
Public cloud provi
ments and low h
cost is main memo
utilization and the
ing under cloud p
Pond, the first me
formance goals an
on the Compute E
to pool memory as
production traces
to achieve most
of with low access
learning models th
pool memory to a
same-NUMA deved
workloads shows
1 **formance within**

CCS CONCEPTS
• Computer sys
Hardware — Em

This work is licensed
under a Creative Commons
Attribution 4.0 International License.
arXiv:23.04.04321v2 [cs.LG] 2023.04.04
ACM ISBN 978-1-60959-100-7
https://doi.org/10.1145/3582063

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 Weimer 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 Chaudhan 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 interconnects 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 module.

We evaluate TPP with diverse memory-sensitive workloads in the production server fleet with early samples of new x86 CPUs with CXL-1 support. TPP makes a tiered memory system perform as an ideal baseline (+15 pgs) that has all the memory in the local tier. It is 18% better than Intel's L3m, 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.

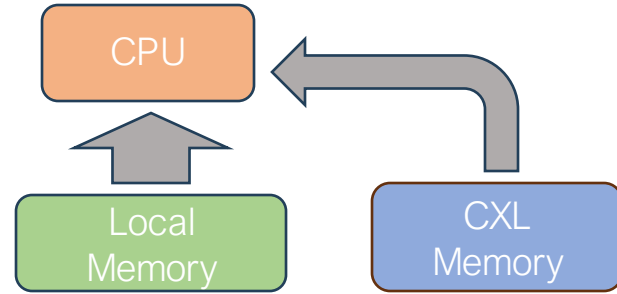
KEYWORDS
Datacenters, Operating Systems, Memory Management, Tiered-Memory, CXL-Memory, Heterogeneous System

ACM Reference Format:
Hasan Al Maruf, Hao Wang, Abhishek Dhanotia, Johannes Weimer, Niket Agarwal, Pallab Bhattacharya, Chris Petersen, Mosharaf Chowdhury, Shobhit Kanaujia, and Prakash Chaudhan. 2023. TPP: Transparent Page Placement for CXL-Enabled Tiered-Memory. In *Proceedings of the 20th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1 (ASPLOS '23)*, March 25–28, 2023, Vancouver, BC, Canada. ACM, New York, NY, USA, 14 pages. <https://doi.org/10.1145/3582063>

1 INTRODUCTION
The surge in memory needs for datacenter applications [11, 61], combined with the increasing DRAM cost and technology scaling challenges [15, 64] has led to memory becoming a significant infrastructure expense in hyperscale datacenters. Non-DRAM memory technologies provide an opportunity to alleviate this problem by holding tiered memory subsystems and adding higher memory capacity at a cheaper \$/GB point [5, 15, 35, 39, 66]. 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

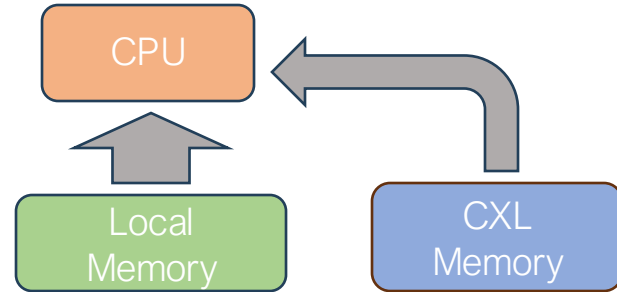
CXL Scale-Up Expansion

- Expansion within a unified tier



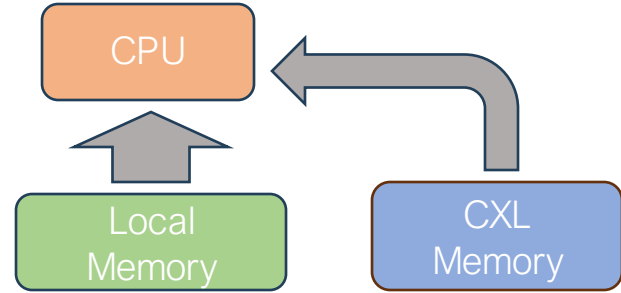
CXL Scale-Up Expansion

- Expansion within a unified tier
 - capacity expansion ✓



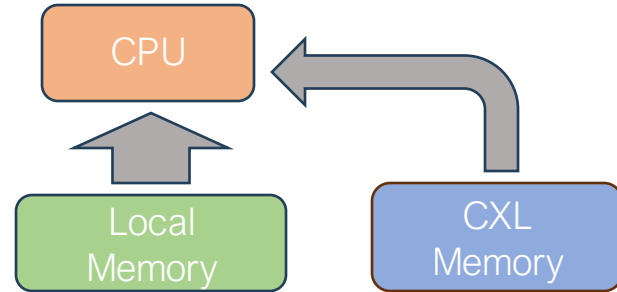
CXL Scale-Up Expansion

- 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

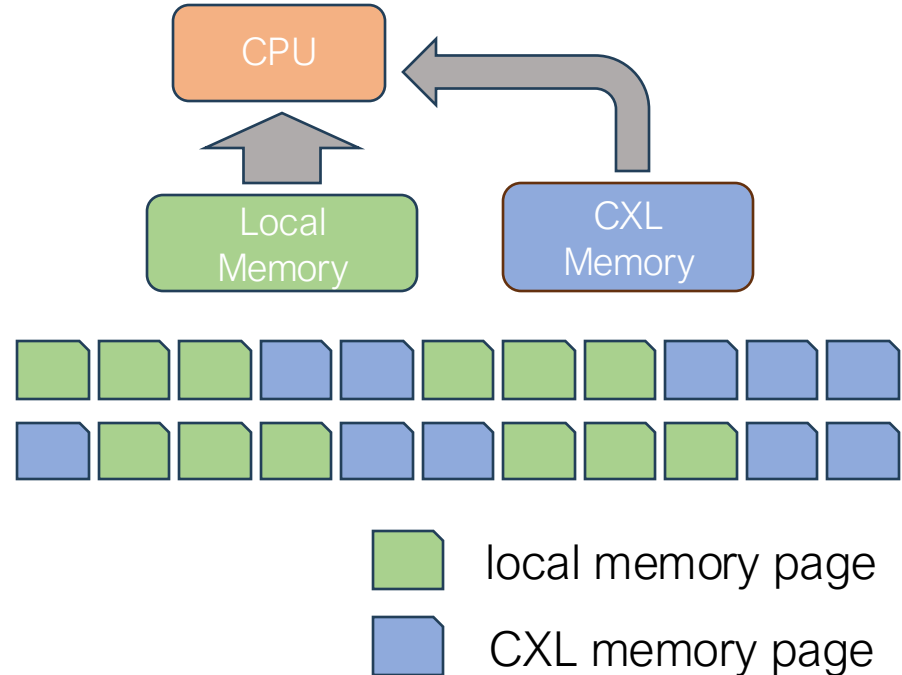


CXL Scale-Up Expansion

- Expansion within a unified tier

- capacity expansion ✓
- bandwidth expansion ✓
 - DDR bandwidth
 - PCIe bandwidth

- Memory interleaving



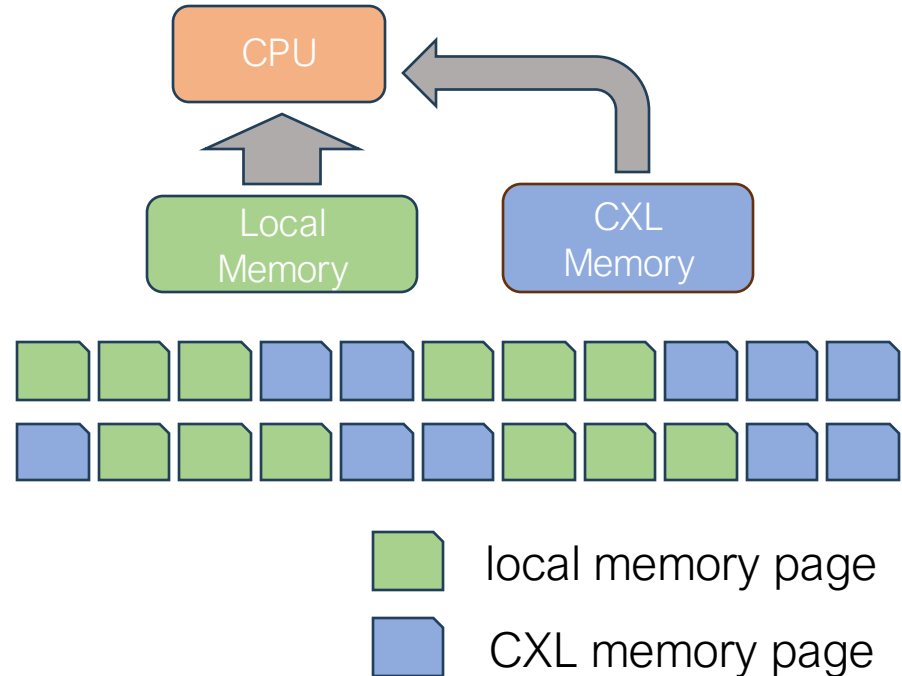
CXL Scale-Up Expansion

- 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

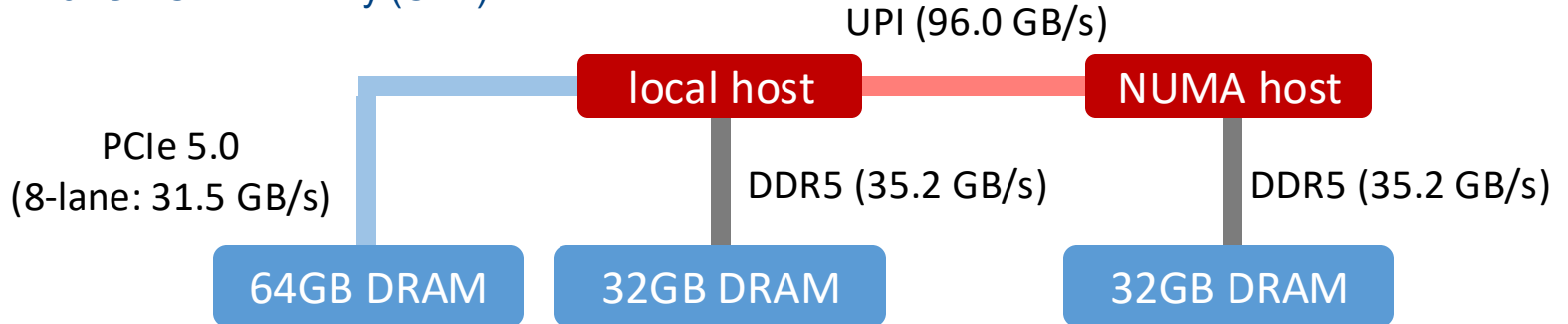
- Testbed

- 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

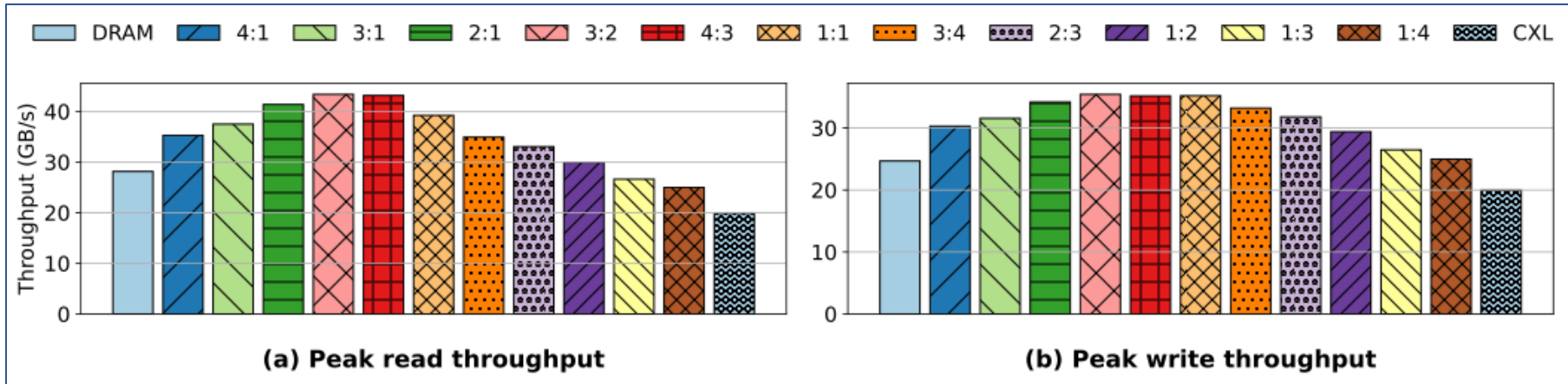
• Testbed

- 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)



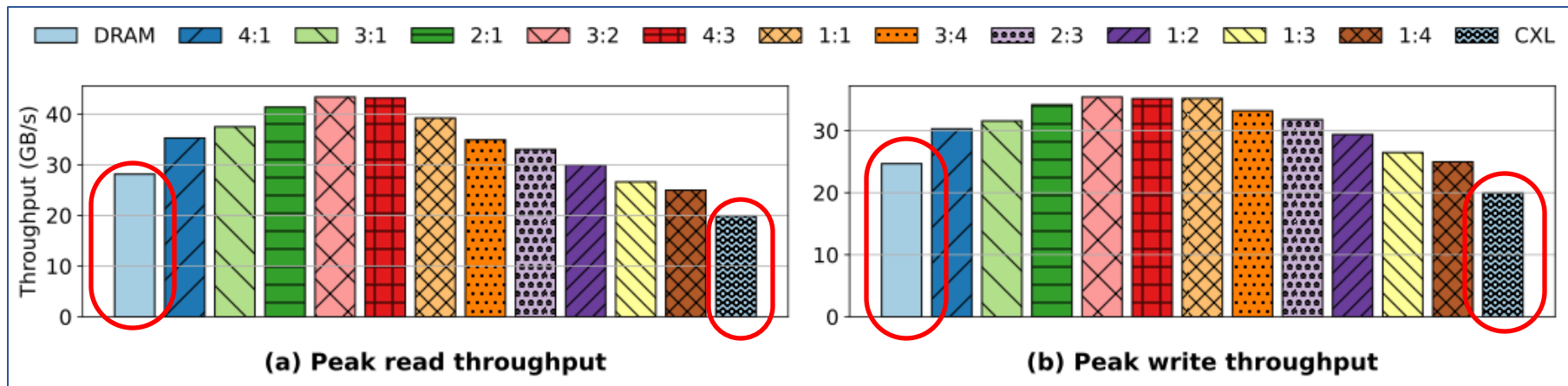
Throughput w.r.t. Memory Interleaving Ratio

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



Throughput w.r.t. Memory Interleaving Ratio

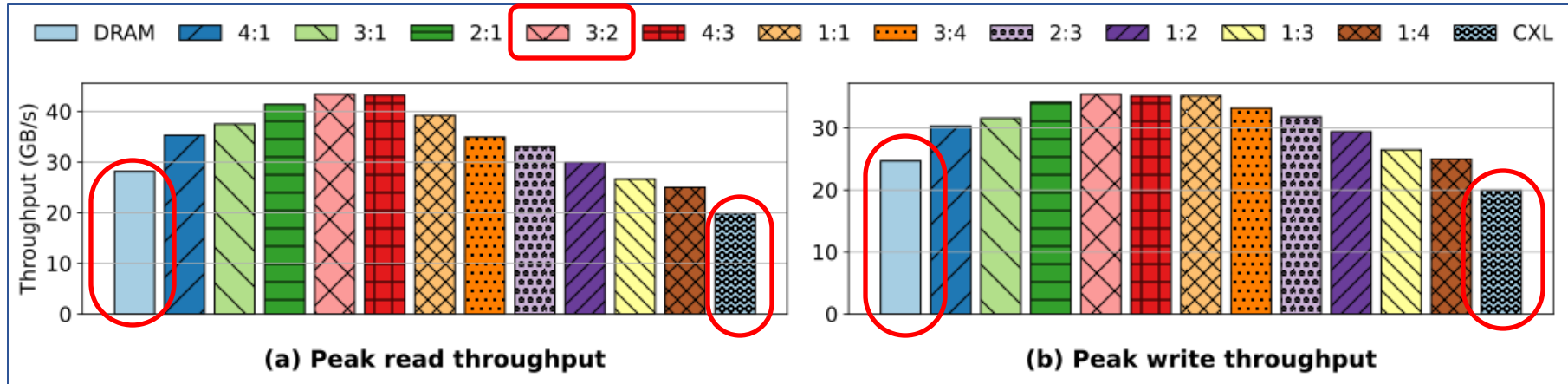
- Experiment workload
 - sequential access over 2 billion records (8-byte) with 32 cores.



Throughput w.r.t. Memory Interleaving Ratio

- Experiment workload

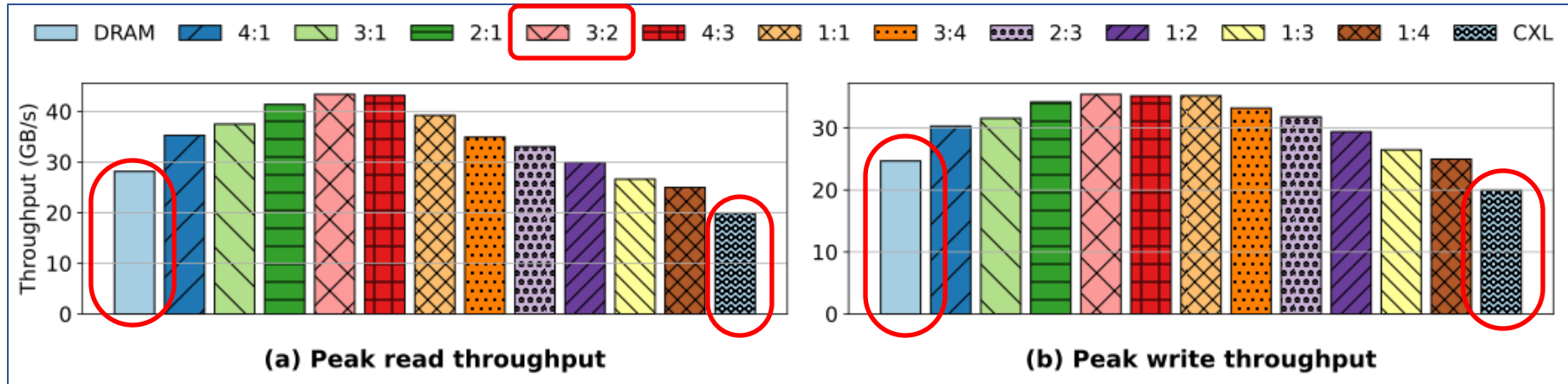
- sequential access over 2 billion records (8-byte) with 32 cores.



Throughput w.r.t. Memory Interleaving Ratio

- Experiment workload

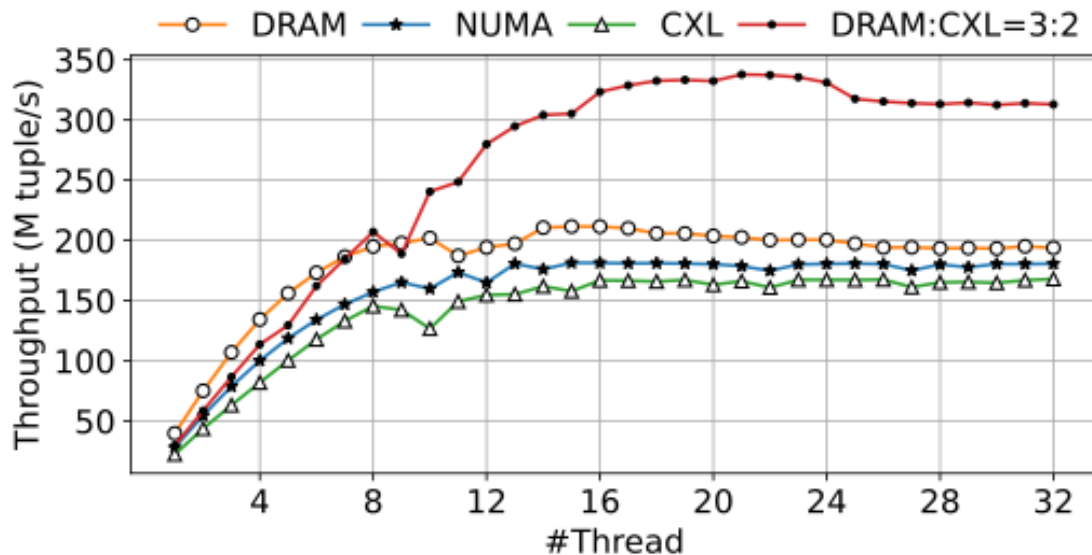
- sequential access over 2 billion records (8-byte) with 32 cores.



1.61x throughput gain

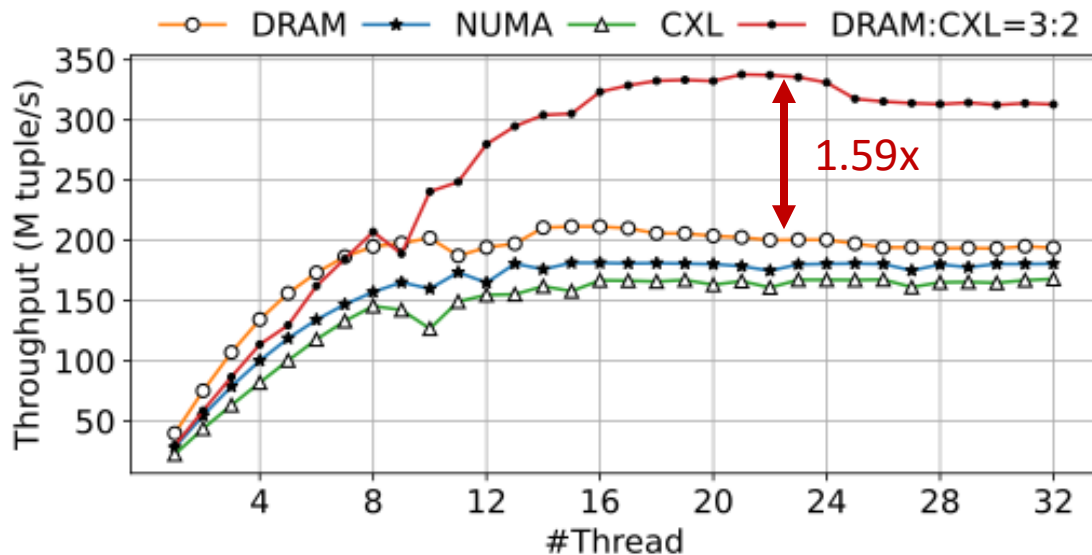
Hash Join Evaluation

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



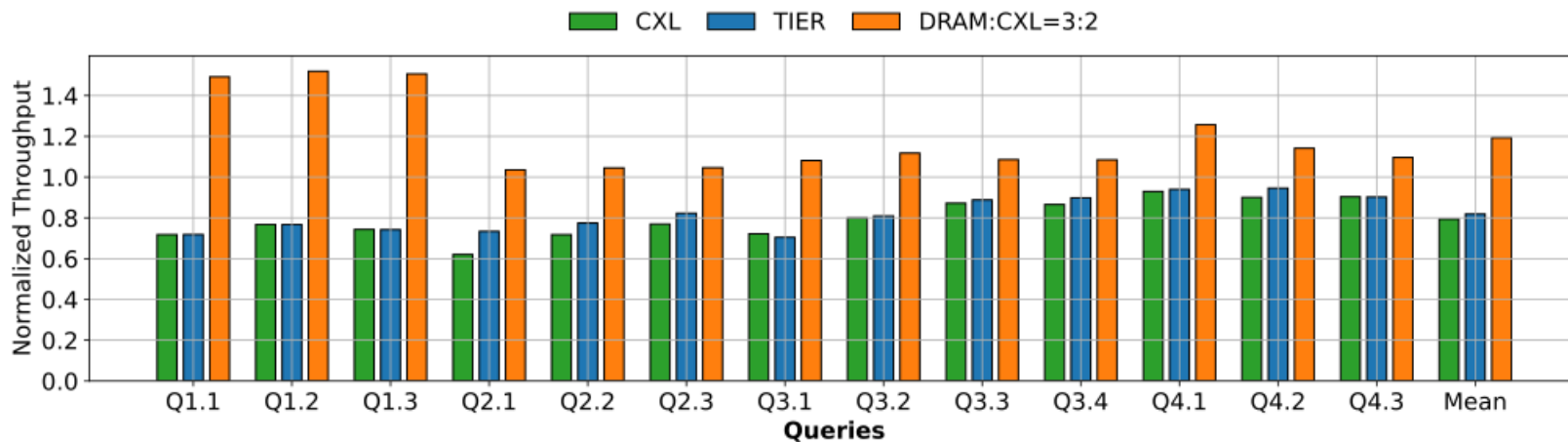
Hash Join Evaluation

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



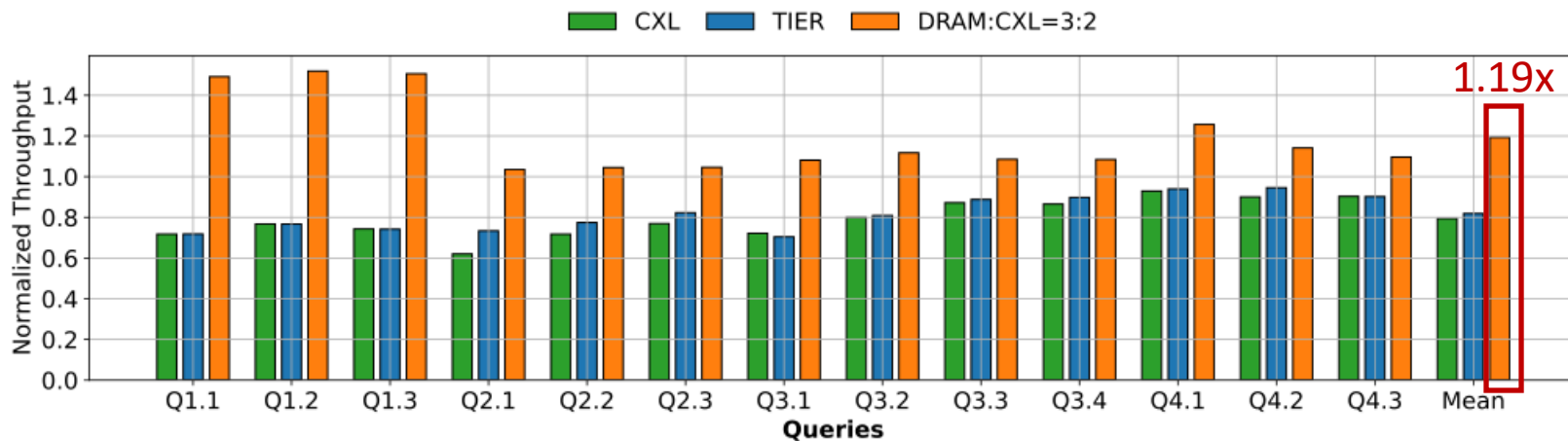
Star Schema Benchmark

- SSB workload
 - column store
 - scaling factor: 30



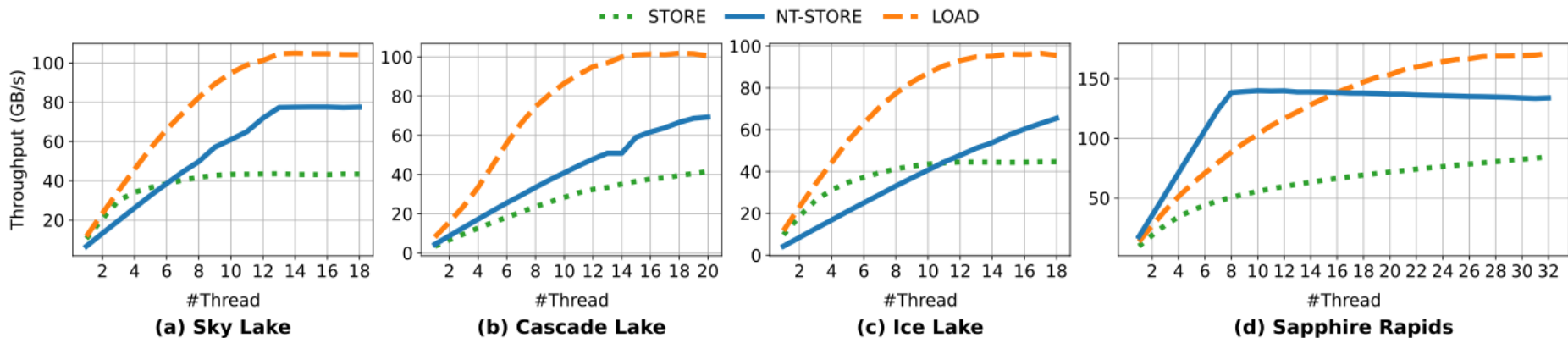
Star Schema Benchmark

- 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



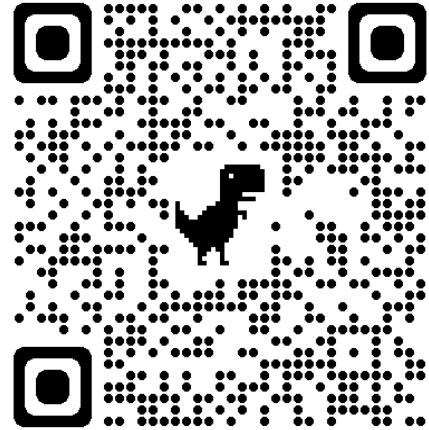
Summary

- 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