SNIA DEVELOPER CONFERENCE

SDC 24

BY Developers FOR Developers

September 16-18, 2024
Santa Clara, CA

CXL for AI/ML

A Practical Guide to Unleashing AI and ML Performance

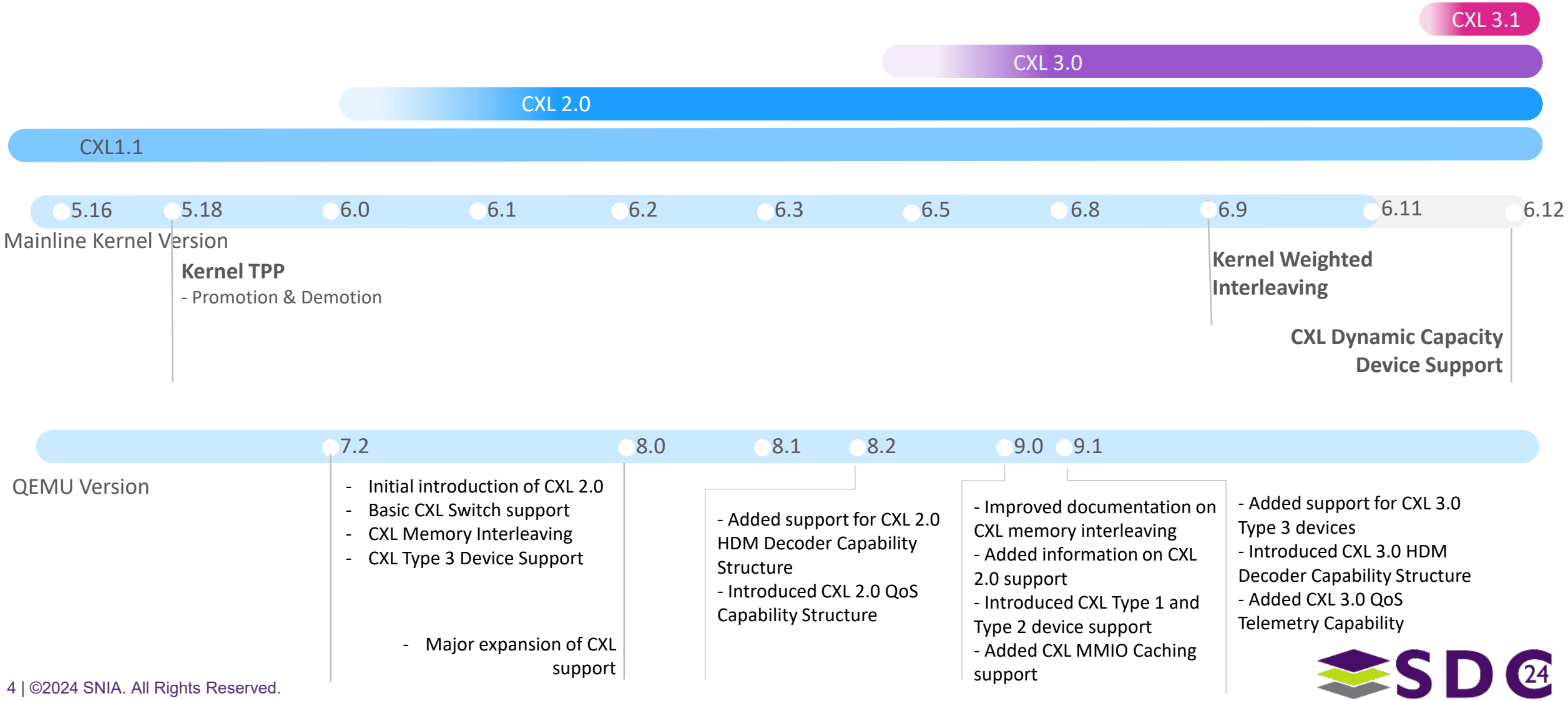
Steve Scargall: Director of Product Management for CXL and AI @ MemVerge

Agenda

- The State of the CXL Software Ecosphere
- Benchmarking CPU, Memory, and GPU Performance
- Memory Placement and Movement Strategies
- The Importance of Memory in AI/ML Workloads
- Building CXL Solutions for AI/ML
- AI/ML Workloads
- The Magic of RAG is in the **R**etrieval
- In-Memory Data Stores
- Call To Action

The State of the CXL Software Ecosphere

Linux Kernel & QEMU CXL Specification Support Roadmap



Useful Tools & Software

Administration

- cxl
- daxctl
- dmesg
- lspci
- lstopo (>3.0)

Fabric Mgt

- Jack Rabbit Labs
- Vendor-Specific APIs and Tools
- fm_cxl**
- cxl-fabric-manager*

Memory Tiering

- Kernel AutoNUMA
- numactl
- Kernel TPP
  - Latency (v5.18)
- Kernel Weighted NUMA Interleaving/Bandwidth (v6.9)
- MemVerge Memory Machine
  - Latency
  - Bandwidth
- Intel 'Heterogeneous Memory' BIOS Options
- AMD 'Special Purpose Memory' BIOS Options

Telemetry

- MemVerge Memory Machine
- CPU Specific Tools
  - Intel PCM
  - AMD μProf
- perf & eBPF

Emulation

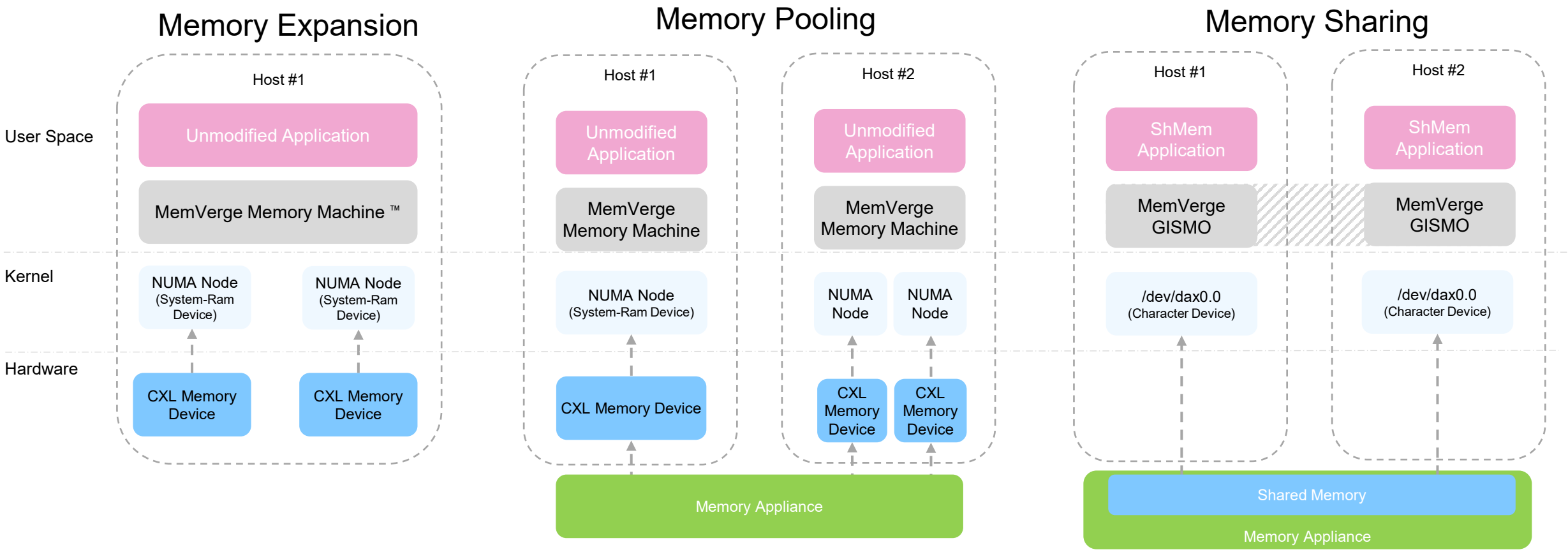
- QEMU
  - Expansion
  - Sharing **
  - DCD **

* Tech Preview

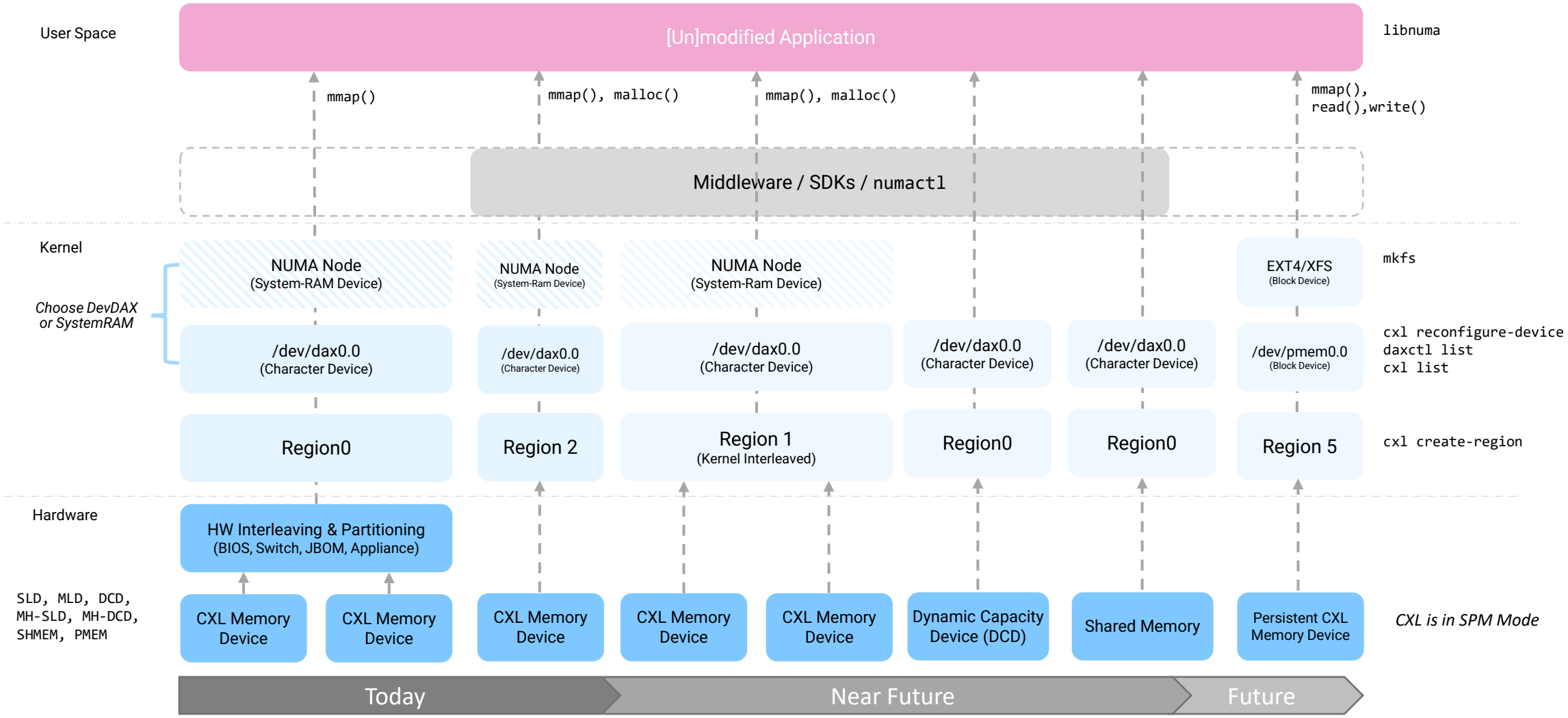
** Requires Patches

SDC 24

How Applications will use CXL for Large-Scale Datasets



CXL Architecture (Type 3 .mem)



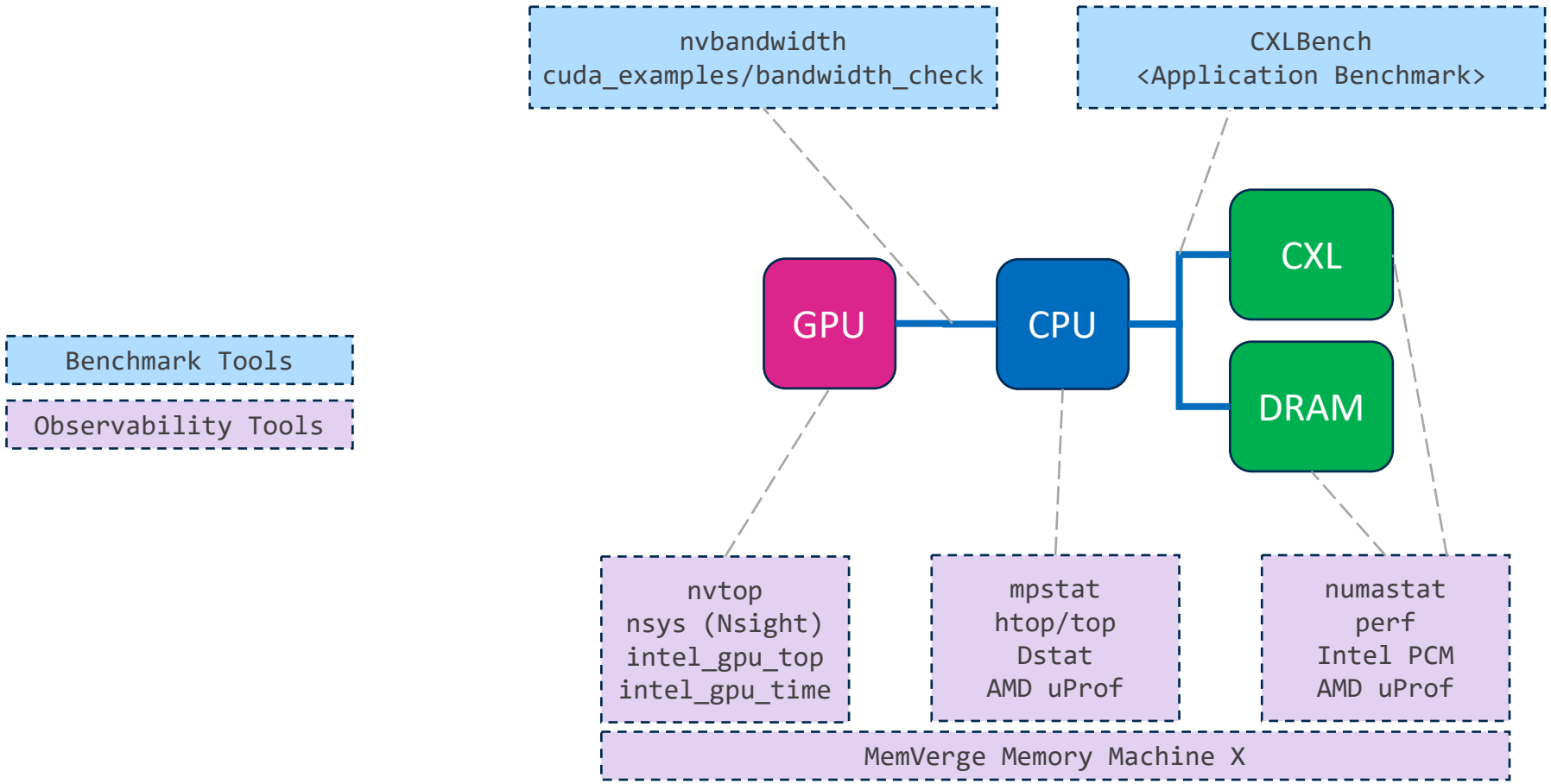
References

- Linux Kernel Documentation:
  - https://www.kernel.org/doc/html/latest/
- Linux CXL Subsystem Maturity Map:
  - https://docs.kernel.org/driver-api/cxl/maturity-map.html
- NDCTL/CXL Utility Releases:
  - https://github.com/pmem/ndctl/releases
- Linux Kernel Release Dates:
  - https://phb-crystal-ball.sipsolutions.net/
- Linux Kernel Mailing List (NVDIMM/CXL):
  - https://lore.kernel.org/nvdimm/
- QEMU Mailing List:
  - https://lore.kernel.org/qemu-devel/
- QEMU CXL Documentation:
  - https://www.qemu.org/docs/master/system/devices/cxl.html

Benchmarking CPU, Memory, and GPU Performance

The Tools of the Trade

Benchmarking CPU, Memory, and GPU Performance



	GPU0	GPU1	GPU*n*
NUMA0			
NUMA1			
NUMA*n*			

Test Matrix
(per CPU Socket)

Reference: https://stevescargall.com/blog/2024/08/benchmarking-gpus-measuring-throughput-between-cpu-and-gpu/

Benchmark Examples

NVidia nvbandwidth

```
$ ./nvbandwidth -t host_to_device_memcpy_ce
nvbandwidth Version: v0.5
Built from Git version: v0.5-1-g4da7d7e

NOTE: This tool reports current measured bandwidth on your system.
Additional system-specific tuning may be required to achieve maximal peak bandwidth.

CUDA Runtime Version: 12040
CUDA Driver Version: 12040
Driver Version: 550.54.15

Device 0: NVIDIA A10

Running host_to_device_memcpy_ce.
memcpy CE CPU(row) -> GPU(column) bandwidth (GB/s)
           0
 0     25.17

SUM host_to_device_memcpy_ce 25.17
```

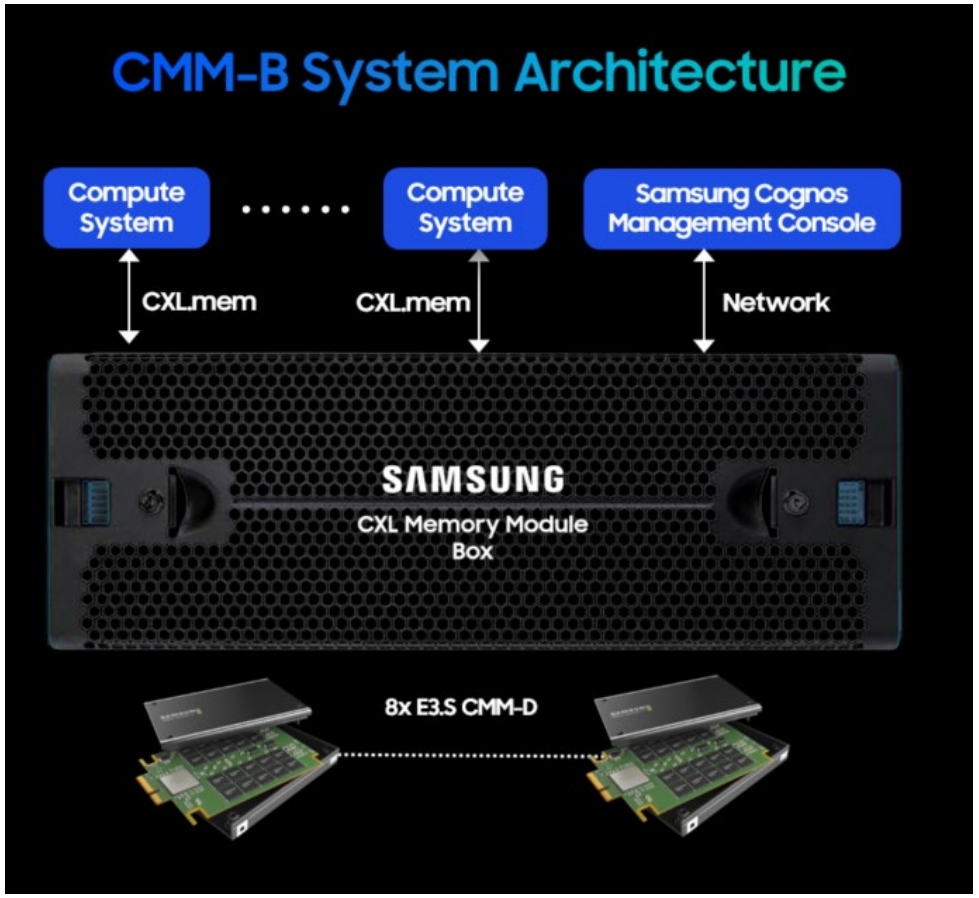
CUDA Examples: Bandwidth_Check

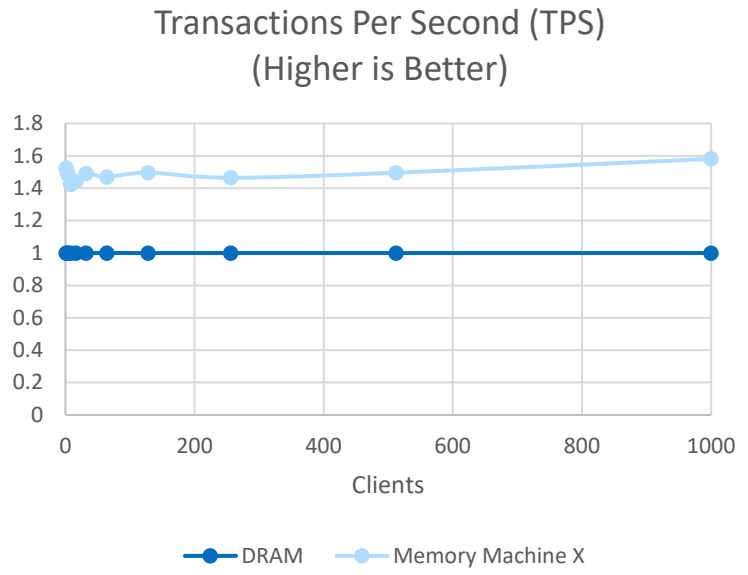
```
$ for f in {1..10}; do numactl --cpunodebind=1 --membind=1 ./main -i 2000 -s 8 -w 5; done
Bandwidth: 25.149124 GB/s
Bandwidth: 25.151405 GB/s
Bandwidth: 25.150078 GB/s
Bandwidth: 25.150782 GB/s
Bandwidth: 25.147242 GB/s
Bandwidth: 25.148067 GB/s
Bandwidth: 25.147434 GB/s
Bandwidth: 25.147575 GB/s
Bandwidth: 25.147165 GB/s
Bandwidth: 25.146469 GB/s
```

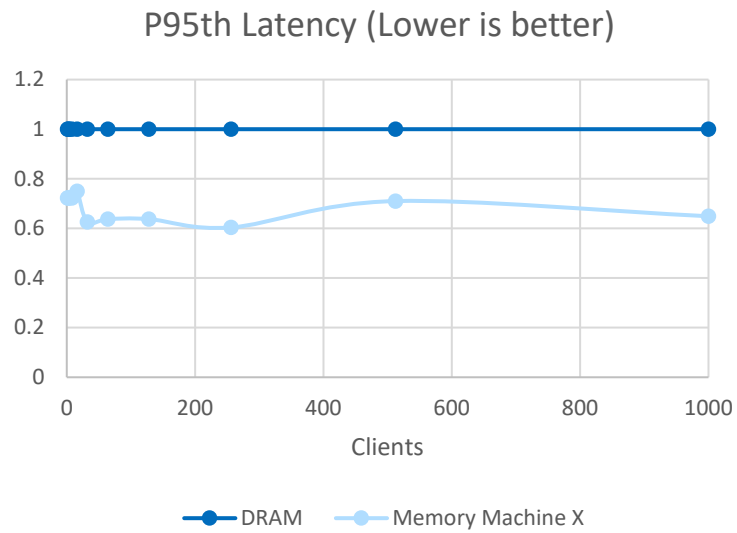
Tips:

- Use numactl to manage host memory and CPU locality
- Use pinned host memory for optimal throughput
  - TradeOff: Memory Tiering doesn't work, but Weighted Interleaving does

Accelerating TPC-C with Samsung CMM-B Memory Pooling Appliance for Computing







Benefits:

- Up to 60% Higher Transactions per Second
- Up to 40% Lower P95 Latency

SUT: 2 x Intel 5th Gen Xeon 8570 (56 Cores), 512GB DRAM Total (32GB x 16 @ 4800 MT/s), 512GB CXL, Ubuntu 22.04 running Kernel 5.18.0, MemVerge Memory Machine v1.5.0

References

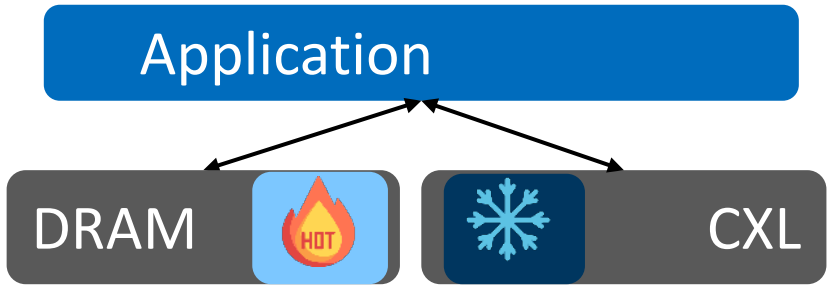
- CXLBench
  - https://github.com/cxlbench/cxlbench
- NVidia NVBandwidth
  - https://github.com/NVIDIA/nvbandwidth
- CUDA Examples
  - https://github.com/drkennetz/cuda_examples

Memory Placement and Movement Strategies

Latency and Bandwidth Optimized Policies

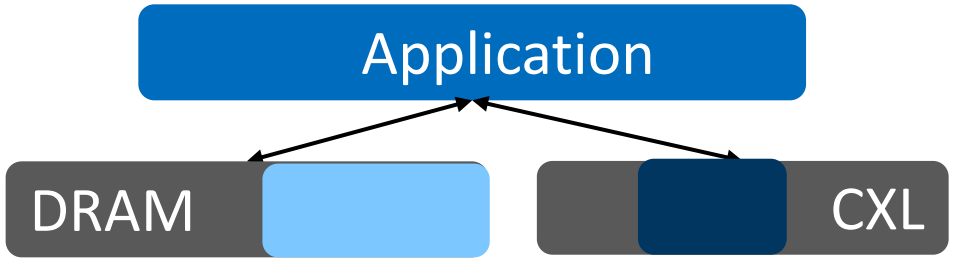
Memory Placement & Movement Policies

Latency Optimized (Tiering)



Latency tiering policies intelligently manage data placement and movement to match the "temperature" of memory pages – Hot or Cold – with the right tier of memory devices.

Bandwidth Optimized



Bandwidth Tiering Policies strategically place data between different tiers of memory proportionate to the bandwidth ratio of the tiers

Weighted NUMA Interleaving (Bandwidth Optimized)

- Contributed by MemVerge & SK hynix
- Available in Kernel 6.9 or newer: https://github.com/torvalds/linux/blob/master/mm/mempolicy.com
- Weighted interleave is a **new policy** intended to use heterogeneous memory environments appearing with CXL.
- Weighted interleave distributes memory across nodes according to a provided weight. (Weight = # of page allocations per round).
- As bandwidth is pressured, latency increases. Figure 1
- Allows greater use of the total available bandwidth in a heterogeneous hardware environment. Figure 2

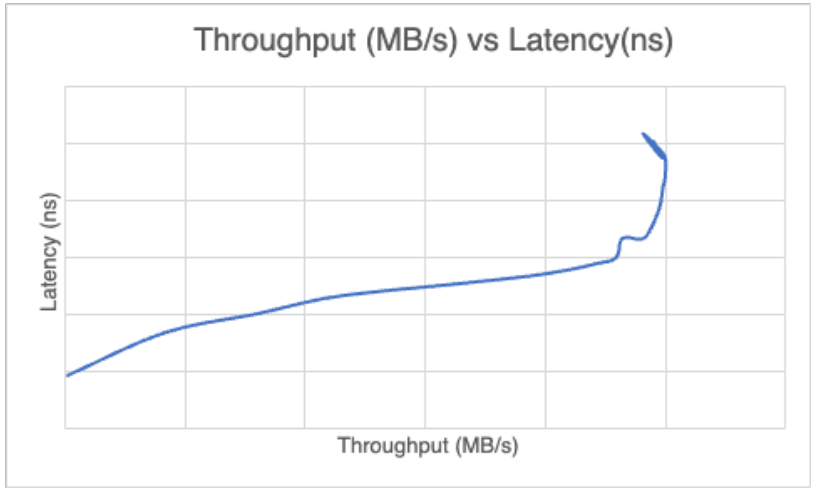


Figure 1: Throughput vs Latency (Hockey Stick)

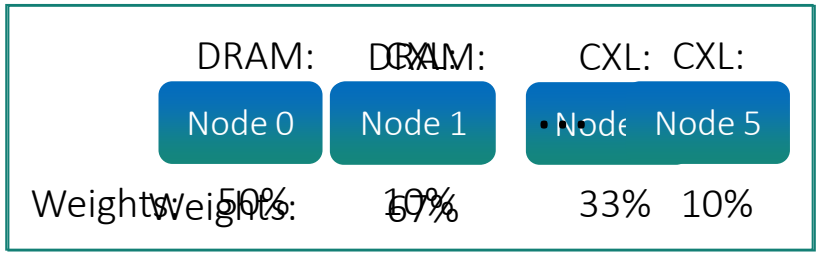


Figure 2: Weighted Interleave Example

NUMA Quality-of-Service (QOS)

- Available in Kernel 6.8 or newer
- Applications may choose which node to allocate their memory based on the NUMA node's performance characteristics.
- The performance characteristics the kernel provides for the local initiators are exported as follows:

```
# tree -P "read*|write*" /sys/devices/system/node/nodeY/access0/initiators/
/sys/devices/system/node/nodeY/access0/initiators/
|-- read_bandwidth (MB/s)
|-- read_latency (nanosec)
|-- write_bandwidth (MB/s)
`-- write_latency (nanosec)
```

- Documentation: https://www.kernel.org/doc/html/latest/admin-guide/mm/numaperf.html

References

- Introducing Weighted Interleaving in Linux for Enhanced Memory Bandwidth Management
  - https://memverge.com/introducing-weighted-interleaving-in-linux-for-enhanced-memory-bandwidth-management/
  - Primer: What it is, How it works, and How to use it.
  - Requires Kernel 6.9+ and one or more CXL devices (or QEMU)
- Linux Kernel Docs
  - https://www.kernel.org/doc/html/latest/admin-guide/mm/numa_memory_policy.html#components-of-memory-policies
  - See MPOL_WEIGHTED_INTERLEAVE

The Importance of Memory in AI/ML Workloads

The Importance of Memory in AI/ML workloads

Main System Memory: The Unsung Hero of AI/ML Performance

- Capacity Demands:
  - Large AI models (e.g., GPT-3, BERT) require hundreds of GB to TB of memory
  - Training datasets often exceed available DRAM capacity
  - Model parallelism and data sharding limited by memory constraints
- Bandwidth Bottlenecks:
  - AI/ML operations are memory-bound, not compute-bound
  - High-speed data movement is crucial for model training and inference
  - GPUs are often starved for data due to insufficient system memory bandwidth

The Importance of Memory in AI/ML workloads

Critical Memory Requirements in AI/ML:

- Training Large Models:
  - Memory capacity directly impacts model size and complexity
  - Larger memory capacities allow for bigger batch sizes, improving convergence
- Real-time Inference:
  - Low-latency memory access is essential for responsive AI applications
  - In-memory processing reduces data movement, enhancing inference speed
- Data Preprocessing and Feature Engineering:
  - Memory-intensive operations for data cleaning and transformation
  - Faster memory enables quicker iteration in feature selection
- Distributed Learning:
  - High-bandwidth memory is crucial for efficient parameter sharing
  - Reduces communication overhead in multi-node training setups

The Importance of Memory in AI/ML workloads

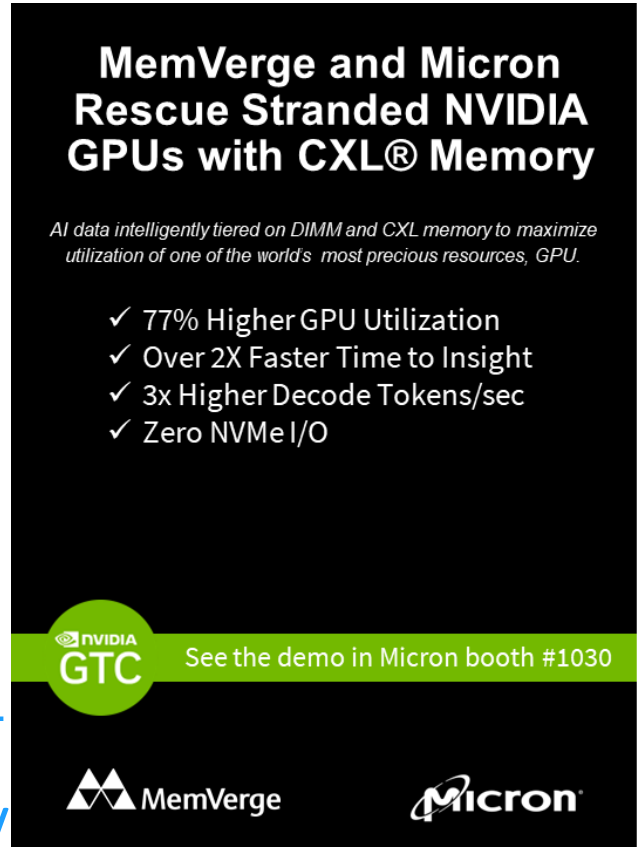
CXL: Addressing the Memory Challenge

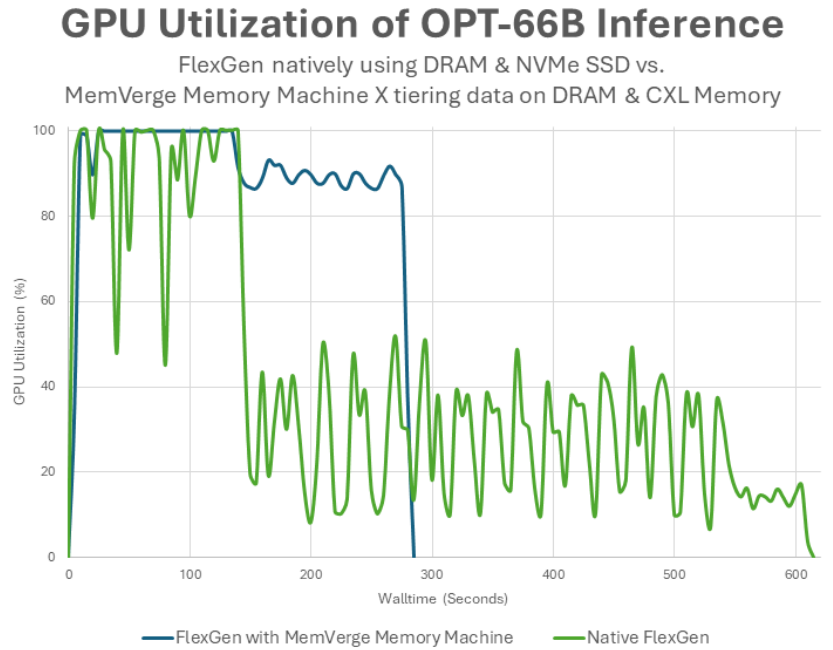
- Expands memory capacity beyond DRAM limits
- Provides high-bandwidth, low-latency access to *limitless* memory pools
- DCDs enable flexible memory allocation for dynamic AI/ML workloads
- Shared Memory reduces data movement, reducing network and storage I/O

AI/ML Workloads

Tiered Memory Inferencing

- FlexGen is a high-throughput generation engine for running large language models with limited GPU memory.
- FlexGen allows high-throughput generation by efficient offloading to main memory (or CXL)
- Paper: https://arxiv.org/abs/2303.06865
- GitHub: https://github.com/FMInference/FlexGen





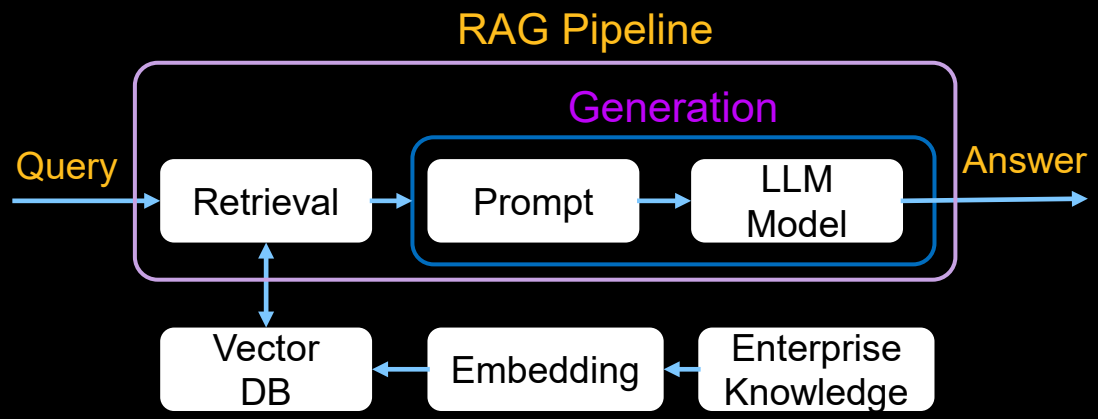
FlexGen Native: Supermicro Petascale Server, AMD Genoa 9634 DP/UP 84C/168T, 8 * 32GB Micron DDR5-4800, 2 x Micron 7450 960GB M.2, Nvidia A10 GPU, Ubuntu 22.04.04 using Kernel 5.15.0, FlexGen AI

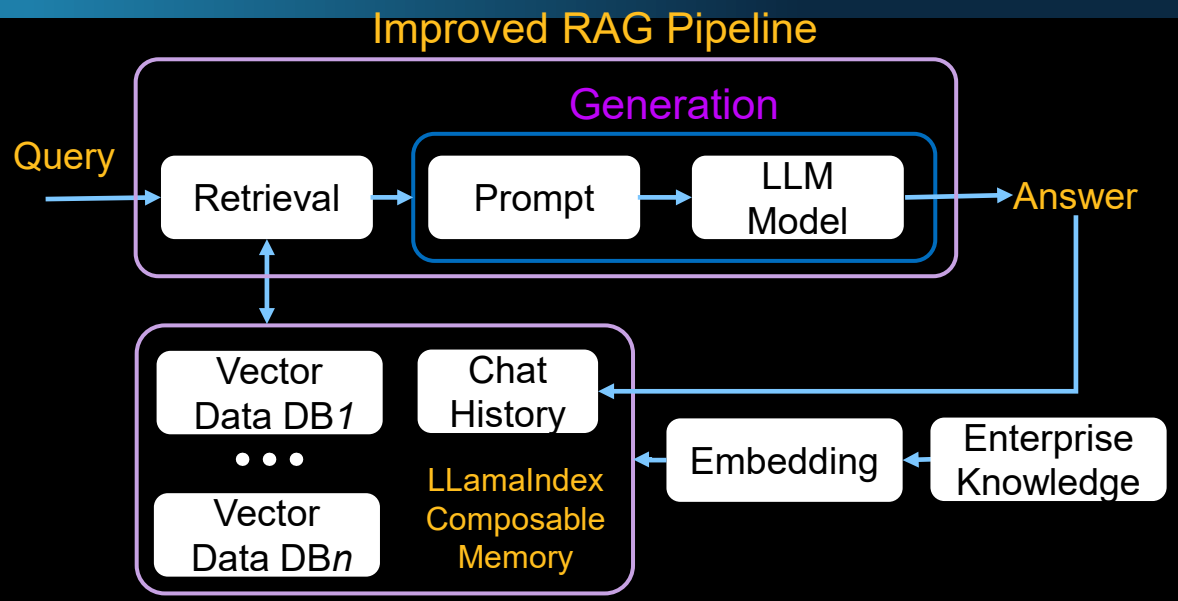
FlexGen using MemVerge Memory Machine: Supermicro Petascale Server, AMD Genoa 9634 DP/UP 84C/168T, 8 * 32GB Micron DDR5-4800, 2 x Micron 7450 960GB M.2, 1 x Micron CZ120 CXL, Nvidia A10 GPU, Ubuntu 22.04.04 using Kernel 5.15.0, MemVerge Memory Machine 2.5.1, FlexGen AI

The Magic of RAG is in the Retrieval

Optimizing the RAG Pipeline for Efficient Data Retrieval

Improved RAG Pipeline - LlamaIndex example





Common Bottlenecks in a RAG Pipeline:

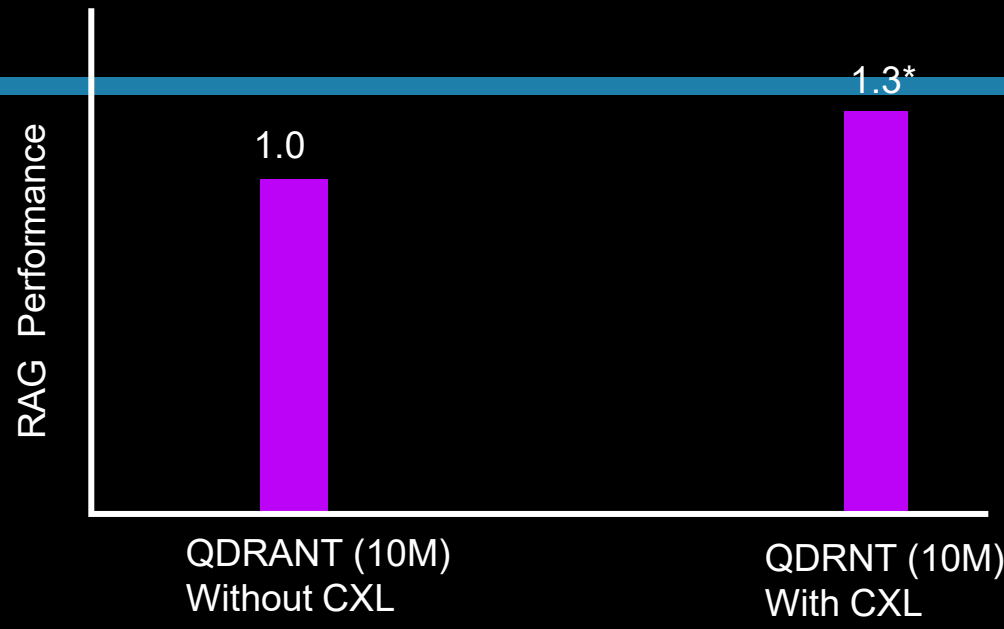
- CPU: Embedding generation
- GPU: LLM inference
- Memory: Storing large datasets and embeddings spills to disk
- Storage I/O: Reading/writing vector data

Benefits of Adding CXL Memory in a RAG Pipeline:

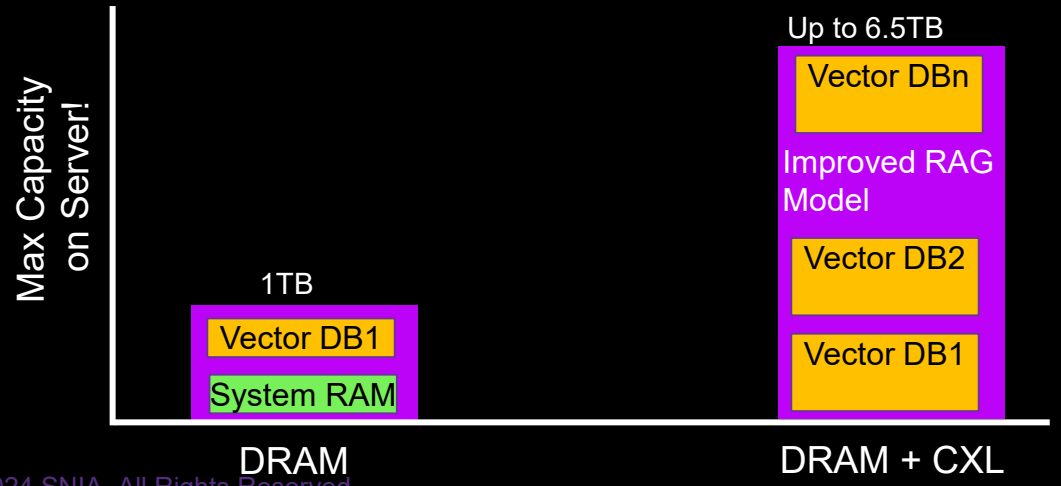
- **Up to 30%** higher Requests per Second from each Qdrant DB
- Efficient retrieval of large-scale and multiple vector databases
- Improved User answer quality due to rich retrieved context
- Large memory capacity can be shared and pooled across nodes
- Improved RAG performance improves GPU utilization

SUT:Supermicro Petascale Server, 2 x Intel 5th Gen CPUs (32 Cores), 1 TB RAM (16 x 64GB DDR5-5600), 4 x Micron CZ120 CXL Memory Expansion Devices, NVidia A10, RHEL 8 running Kernel 6.8, MemVerge Memory Machine 1.5.0, LLamaIndex, Qdrant Vector Database, Llama2 Model

RAG Workload Analysis for CXL



* Based on VectorDDBench and Memory Machine v1.5.0 with 9:1 DDR/CXL Interleaving ratio; measured at p99 latency



RAG Workload Result Summary

- 30% Improvement by adding CXL Memory to Server
  - Tuning of the interleaving ratio required to optimize perf
- Vector DB Index tables MUST fit in memory to avoid steep performance drop
- CXL adds more modularity and capacity to “scale up” RAG model

Also, CXL adds TCO benefits via Pooling and Sharing

- DRAM memory allocated for RAG execution be freed for general purpose
- Vector DB tables in CXL Mem can be shared and pooled across other RAG nodes/VMs
- Higher CXL capacity reduces power by reducing data transfers over storage network

How Much RAM Could a Vector Database Use If a Vector Database Could Use RAM

A formula to estimate the raw and total memory required to store the data (dense vectors):

Raw: Memory (bytes) = Vector Length × Number of Vectors × Size of Data Type
Total: Memory (bytes) = Vector Length × Number of Vectors × Size of Data Type × DB Metadata Ratio

Raw Vector Requirements:

Vector Length	Data Type	Data Type Size (bytes)	1M Vectors	10M Vectors	25M Vectors	50M Vectors	100M Vectors
384	FP32	4	1.536 GB	15.36 GB	38.4 GB	76.8 GB	153.6 GB
768	FP32	4	3.072 GB	30.72 GB	76.8 GB	153.6 GB	307.2 GB
1024	FP16	2	2.048 GB	20.48 GB	51.2 GB	102.4 GB	204.8 GB
2048	BF16	2	4.096 GB	40.96 GB	102.4 GB	204.8 GB	409.6 GB
4096	INT8	1	4.096 GB	40.96 GB	102.4 GB	204.8 GB	409.6 GB

Reference: https://stevescargall.com/blog/2024/08/how-much-ram-could-a-vector-database-use-if-a-vector-database-could-use-ram/

How Much RAM Does My Data Need?

CXL 3.1 Specification Document

Attribute	Value
Original PDF File Size	11.68 MB
Number of Pages	1,166
Number of Paragraphs	7,335
Number of Words	422,967
Number of Characters	2,691,875
Number of Tables	700
Number of Images	335

Properties Used:
- Chunk Size: 512
- Chunk Overlap: 100
- Effective Chunk Size: 412
- Vector Dimensionality: 1536
- 5 Characters per Token
- 150 Tokens per Figure description
- FP32 Vector Data Type (4 bytes)

Attribute	Value
Embedding Model	OpenAI's text-embedding-ada-002
Number of Tokens (Document Text)	538,375 tokens
Number of Vectors (Document Text)	1,307 vectors
Number of Vectors (Tables)	700 vectors
Number of Vectors (Images) (150 tokens per image)	122 vectors
Total Number of Vectors	2,129 vectors
Memory per Vector (1536 * 4 bytes)	6,144 bytes (6 KB)
Total Memory for Raw Vectors	13.06MB
Total Memory with DB Overhead (Assume 50%)	19.59MB

Calculations are close approximations for illustration only.

The Cost of Storing Your Data at Scale (Public Cloud)

A cost comparison table using Qdrant's cloud calculator, which provides insights into the monthly expenses using **reserved** AWS instances for various vector sizes and quantities (RAS is not considered here)

Vector Size	1 Million	5 Million	10 Million	25 Million	50 Million	100 Million
384	$219.00	$219.00	$219.00	$438.00	$876.00	$1,752.00
1024	$219.00	$438.00	$876.00	$1,752.00	$3,504.00	$7,008.00
2048	$438.00	$876.00	$1,752.00	$3,504.00	$7,008.00	$14,016.00
4096	$876.00	$1,752.00	$3,504.00	$7,008.00	$14,016.00	$19,681.00

Qdrant's cloud calculator limits instances to 256GB RAM and will scale out across nodes if more memory is required. This adds complexity (sharding) and network latency.

Reference: https://stevescargall.com/blog/2024/08/how-much-ram-could-a-vector-database-use-if-a-vector-database-could-use-ram/

Cost of Storing Your Data (On-Prem) using DRAM & CXL

Configurations (Dual CPU, unless otherwise noted)	Server Memory Spec			Socket DRAM - DDR 5				CXL DRAM					
	Total Size (GB)	Total Cost	Per GB Cost	DIMM size (GB)	# of DIMMs	Size Subtotal	Cost Subtotal	DIMM type	DIMM size (GB)	# of DIMMs per AIC	# of AICs	Size Subtotal	Cost Subtotal
A. Socket DRAM	4,096	$46,080	$11.25	128	32	4,096	$46,080						
B. Socket & CXL DRAM	4,096	$22,928	$5.60	64	32	2,048	$6,464	DDR5	64	8	4	2,048	$16,464
C. Socket & CXL DRAM	4,096	$20,072	$4.90	64	32	2,048	$6,464	DDR4	128	8	2	2,048	$13,608
D. Socket DRAM, Quad CPU	8,192	$92,160	$11.25	128	64	8,192	$92,160						
E. Socket & CXL DRAM	8,192	$47,288	$5.77	64	32	2,048	$6,464	DDR4	128	8	6	6,144	$40,824
F. Socket & CXL DRAM	11,264	$62,656	$5.56	96	32	3,072	$12,224	DDR4	128	8	8	8,192	$50,432
G. Socket & CXL DRAM, Quad CPU	32,768	$425,600	$12.99	256	64	16,384	$204,800	DDR5	256	8	8	16,384	$220,800

DRAM Pricing from May 2024
CXL Pricing is estimated

In-Memory Data Stores

Accessing Your Data at Memory Speeds

In-Memory Data Stores

MemVerge GISMO

- Uses CXL 3.0 Sharing
- Object Store
- Access Data using PUT(), GET(), DELETE(), UPDATE()
- Not Persistent

FAMFS

- Uses CXL 3.0 Sharing
- File System
- Access Data using open(), close(), read(), write(),…
- Not Persistent

Products are in development

Call To Action

Call To Action

- "*The Future is Now! Don't Wait*." Andrey Kudryavtsev, Micron
- Talk to your OEM for system and device information and availability.
- Use QEMU CXL Emulation Features to become familiar with CXL
  - Attend the "Emulating CXL with QEMU" session at 13:30-14:20 in Cypress
- Join and Contribute to the Communities that matter to you

Thank You

Please take a moment to rate this session.

Your feedback is important to us.

