

OPEN
Compute Project

CMS: End User Use Cases

Version 1.0

Date: Oct 17, 2023

Author (s) :
Manoj Wadekar, Meta
Samir Rajadnya, Microsoft
Vikrant Soman, Uber

Reviewer (s):
Anjaneya Reddy Chagam, Intel Corporation
Durgesh Shrivastava, Nvidia
Aran Ziv, Sunrise Corporation

Executive Summary

This document defines the **Open Compute Project (OCP) Composable Memory Systems (CMS)** Use cases from End User perspective. CMS is an emerging paradigm that facilitates dynamic and unified memory management across diverse memory technologies, interconnects, and hierarchies. The “CMS End User Use CasesCases White Paper” serves as a vital reference point for component/system designers, data center operators, and end customers as they navigate the landscape of commercial implementations and validation of solutions against target use cases. This entails selecting an ideal combination of host processor platforms, CMS devices, chassis configurations and SW components. The initial CMS Use Cases target value of CMS to Caching/DB/Data Warehouse, Virtualization/Containers and AI Inference use cases.

Table of Contents

1. Compliance with OCP Tenets **4**
1.1 Openness 4
1.2 Efficiency 4
1.3 Impact 4
1.4 Scale 4
1.5 Sustainability 4
2. Introduction **5**
3. Use Cases **6**
4. Cache/Databases/Data Warehouse **7**
5. Virtualization/Containers **9**
Virtualized environments 9
Containerized environments 10
6. AI/ML - Inference **13**
7. Conclusion **16**
8. Glossary **17**
9. References **18**
10. License **19**
11. About Open Compute Foundation **20**

1. Compliance with OCP Tenets

1.1 Openness

The OCP CMS End User Use Cases were developed in dialogue with the entire OCP CMS community membership and the resulting document represents the initial set of use-cases for developing composable memory solutions. CMS usage models are nascent, so this document will evolve over time, and that this document represents the wisdom of the collective ecosystem on the usage scenarios for creating CXL based composable memory systems.

1.2 Efficiency

The OCP CMS end user use cases document lays the foundation for developing hardware/software co-designed composable memory solutions to meet cloud service provider needs.

1.3 Impact

Having a common set of End User Use Cases helps us guide CMS workstream scope to develop future white papers, specifications, case studies and vendor reference implementations.

1.4 Scale

CMS End User Use Cases enable consistent vendor neutral value added composable solutions using open eco-system software, hardware form factor specifications and best practices.

1.5 Sustainability

CMS End User Use Cases and reference implementations are designed to optimize performance and resource utilization for underlying physical memory technologies thereby achieving data center energy efficiency.

2. Introduction

As industry continues to expand in the Enterprise and Hyperscale market space, memory continues to play an even more pivotal role. Data intensive applications like AI/ML, databases etc. are pushing the boundaries of today's server performance. For example, Generative AI, like Large Language Models (LLMs), relies on servers with extensive memory to achieve superior performance and generate high-quality, context-aware text efficiently. The industry is currently developing diverse new and evolving memory technologies, interconnects & hierarchies, which will provide better memory capacity, latency, throughput, sharing, and scalability for data intensive applications. These hardware innovations merged with appropriate usage will provide better performance and optimum resource utilization of the underlying physical memories. Some applications continue to demand higher capacity of memory (driven by constant increase in compute performance). And other categories of applications continue to demand higher performance of memory to unlock platform limitations. Design engineers continue to look for balanced system solutions that take advantage of new technologies, products and software capabilities that enable these applications for rapid growth. New hardware and software solutions and abstractions are expected to release new value for memory use. It's expected to expand the boundaries of location transparency, automated tiering across hybrid memory technologies, enable pooled capacities with unified namespaces.

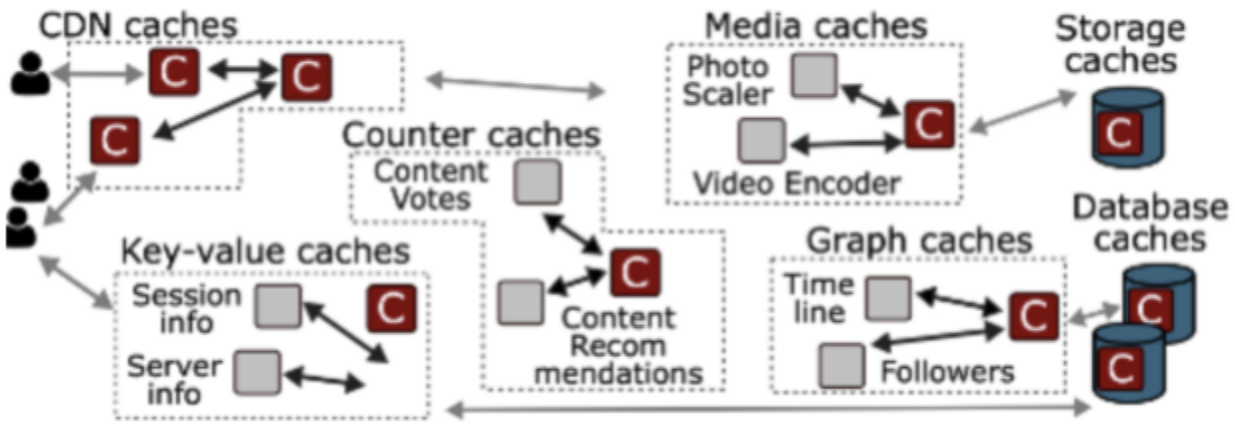
The Open Compute Project Foundation (OCP) announced during Global OCP 2022 that formal work has been undertaken under the **Composable Memory Systems** (CMS) Project. In this white paper we will explore use cases for Composable Memory systems architecture.

3. Use Cases

CMS group hosted discussions for what are the driving use cases for next generation composable memory systems and this white paper summarizes these use cases. These use cases fall into following categories:

1. Cache/Database/Data Warehouse use cases
2. Virtualization Use Cases
3. AI/ML Inference Use Case

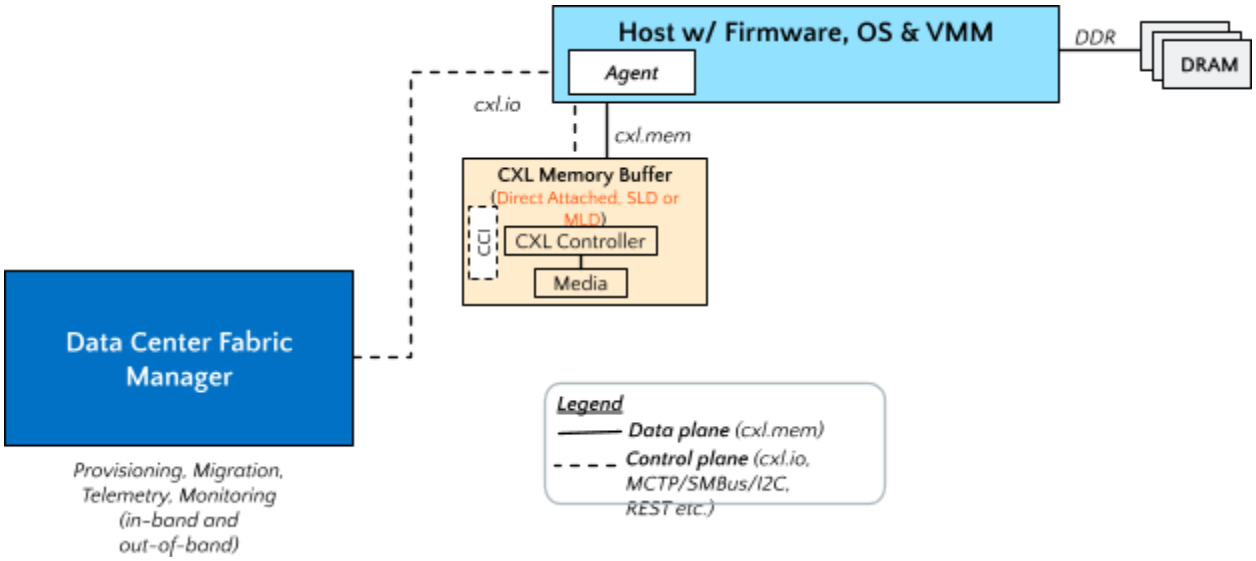
4. Cache/Databases/Data Warehouse



Large scale web service providers rely on caching to deliver a great user experience. Caching solutions facilitate the easy development, scaling and maintenance of high performing caches. Memcached is a well known, simple, in-memory caching solution. Memcached requires server(s) with large memory to accommodate the cached data. As data volume increases, there is a substantial rise in the demand for greater memory capacity.

Meta leverages Memcached as a building block to construct and scale a distributed key-value store that supports the world's largest social network. Meta has open sourced the code that drives Memcache - it's called Cachelib (https://cachelib.org/) . Cache library (CacheLib) is a C++ library for accessing and managing cache data. It is a thread-safe API that enables developers to build and customize scalable, concurrent caches.

Caching relies on having data available in the cache for targeted amounts of "Retention Time". This drives the need for a higher amount of memory within a platform to achieve efficiency to avoid CPU stranding. Memcache can trade off memory performance for higher capacity to a reasonable degree to optimize Queries Per Second (QPS) for a given system.



Rationale of how CMS helps the application:

- CMS can enable memory expansion within a system in a cost effective manner by complimenting memory in the system with CXL-attached memory capacity.
- Some of the efficiency gains may come from
    - Increasing memory capacity
    - Increasing memory bandwidth
    - Driving cost efficiency through right technology
- CXL based memory extension though adds additional latency

To achieve the desired performance with tiered memory solution - there are two ways of achieving the right placement of the data in the memory tiers.

1. **Kernel Managed driven tiering**: Kernel can make tiering and data movement decisions based on the data access patterns. This allows applications to run without any changes. E.g. Cachelib can directly take advantage of CXL memories if the kernel manages the movement of hot/cold pages between DRAM and CXL-attached-memory in the above diagram.
2. **Application driven tiering**: Caching and tiering decisions can be driven by applications as well if they have better information about the access patterns and have more control of the system for managing the memory within the system.

5. Virtualization/Containers

Cloud service providers and hyperscalers with a large footprint of virtual machines and containers are always pushing for the highest utilization of the hardware infrastructure in order to get the best cost economics for running these VMs and containers. There are two big challenges that CSP/hyperscalers are currently facing in order to achieve the best possible utilization of their hardware used for virtualization/containerization.

Firstly, the virtual machines and containers that customers run are of varying sizes and at varying times which leads to uneven allocation and utilization of resources from host to host. **The result is "stranded" resources per host, the most expensive of all being memory in this case**. When you have even a few GB of stranded memory across millions of hosts that is an inefficiency that costs millions of dollars.

Secondly, we are seeing core densities in a socket increase rapidly leading to higher demand of memory per socket that cannot be supported without expensive high capacity memory modules and limited number of memory channels behind each socket. **This causes stranded resources in terms of cores leading to underutilization of the system and thus higher costs**.

Memory pooling using CMS provides an elegant solution by decoupling tight affinity of memory in the system with the cores in the socket. Essentially, memory as a resource becomes more elastic allowing operators of large scale VM and container clusters to fine tune the usage of cores and memory independently and get better overall utilization of their hardware deployments and lower the cost of operation. Given that memory is shared across multiple servers, the utilization of memory goes up, further optimizing resource allocation. In the end both customers and datacenter operators benefit from the cost savings realized by using CMS memory pooling for virtualization and containerization. The sections below provide some more details on the use case

Virtualized environments

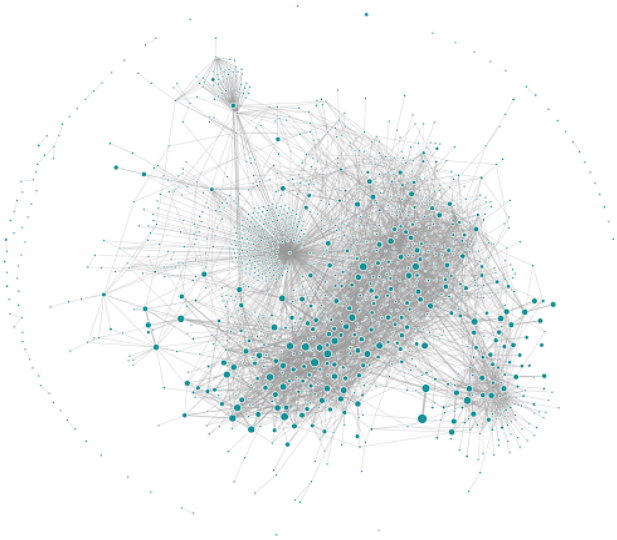
Memory is a key driver of both performance and cost in a cloud environment. DRAM can be 50% of server cost. In this environment memory stranding is identified as the dominant source of memory waste and a potential source of massive cost savings. Memory stranding happens when all cores of a server are rented in the form of customer VMs. Cloud environment is more challenging because any attempt to reduce memory cost has a severe impact on customer VM performance. Existing techniques for process level memory compression require page fault handling leading to increase in latency.

Hence memory pooling is a promising approach given memory is shared among multiple servers and stranded memory can be returned to the disaggregated pool and used by other servers. (More details: "Pond: CXL-Based Memory Pooling Systems for Cloud Platforms")

Containerized environments

Uber adopted a microservice oriented architecture to support high feature velocity and hypergrowth of users during the early days. At present, there are 1000s of containerized microservices running on Ubers datacenter hardware. This network of microservices is spread across millions of cores and PB of memory, incurring a significant TCO every year on datacenter hardware.

Not all services are created equal. Each microservice serves a unique function in the overall operation



of the application. This implies the business logic and complexity for each service will be different resulting in different amounts of compute resource requirements for each. For example some services could use a quota of 2 vcpus and 4GB while some other services may use a size of 6 vcpus and 24GB. **The average memory used for our compute clusters is around 60% @ non-peak average CPU utilization levels.** From a TCO perspective, this means wasted $ since memory is the most expensive component in the server.

This sort of memory stranding problem is similar to the virtualization use case discussed in the previous use cases.

Solutions:

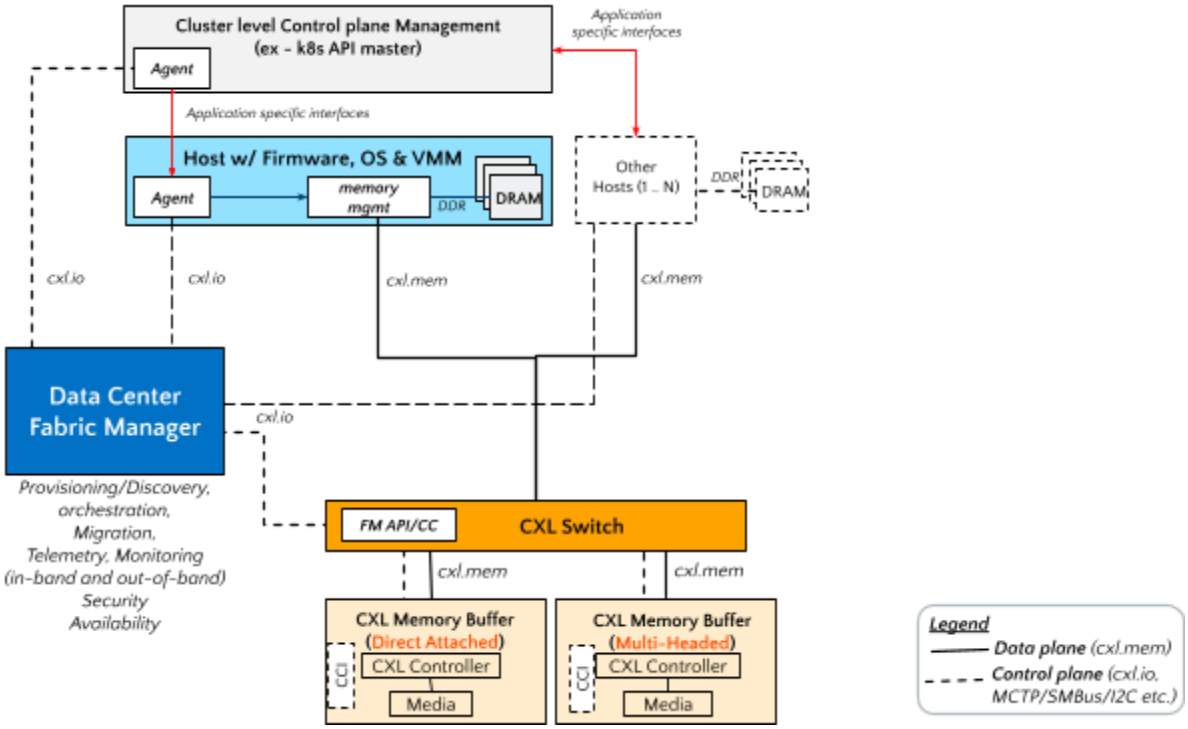
1) A naive way to resolve memory stranding is to create tight coupling between the amount of memory available on the host and the size of the containers running on the host. Containers with smaller memory footprint (e.g., 2GB) would run on hosts with total memory = # of vcpus * 2GB and containers with higher memory needs (e.g., 8GB) would run on hosts that have total memory = # of vcpus * 8GB. This solution is simple to implement from a hardware standpoint

but has a **huge software overhead** associated with managing clusters across different hardware types and scheduling of containers of different sizes on each of these SKUs. In addition, having fixed “T shirt” sizes for containers means that the stranded memory now effectively resides in the container

2) Given that we don't use all the memory allocated, another approach is to reduce the core to memory ratio to a single value (e.g., 1vcpu:4GB). However, this could mean potential stranding of compute resources in periods when higher memory is needed. **Stranding of compute resources** also results in wasted $.

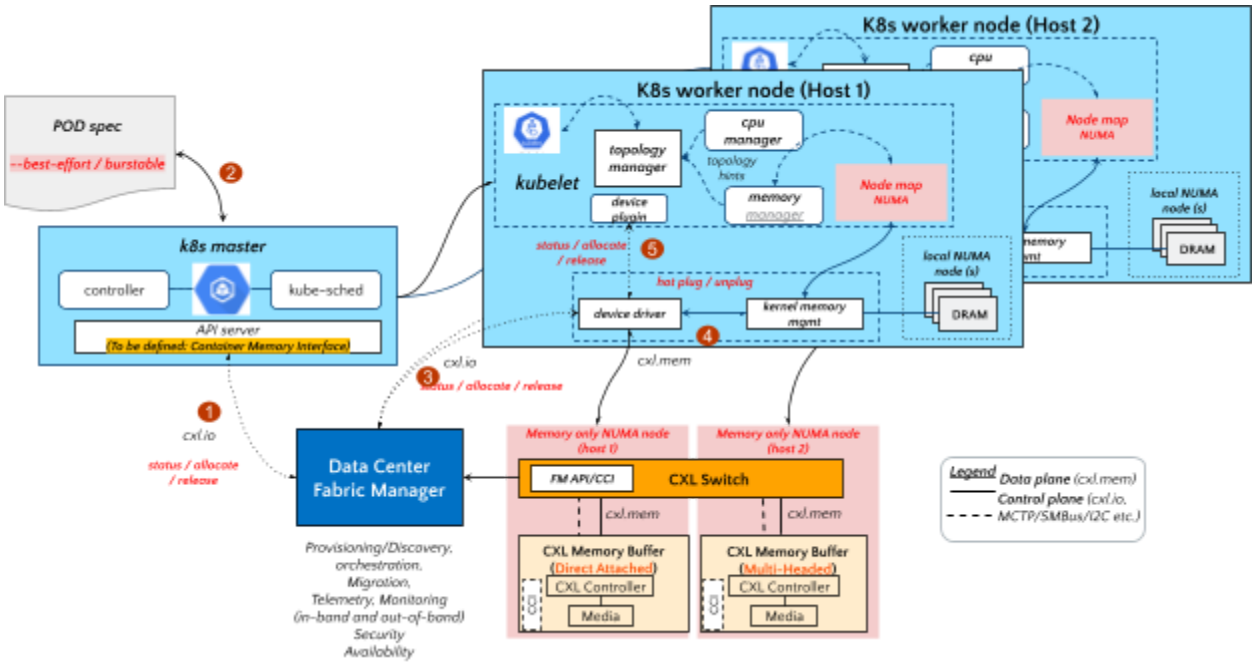
CMS pooling provides an ideal solution with ***low software overhead*** with ***flexibility*** of container core to memory sizing on an ***cost effective*** homogeneous hardware platform.

A pooled memory solution using CMS logical architecture building blocks is as below



Uber uses open source K8s to schedule and run containers on large clusters built using standard form factor servers and cloud instances. Key requirements of deploying CMS pooling solutions within k8s clusters include not treating pooled memory as special. This is because **Memory attributes are not visible to service owners** when requesting deployment of their instances. In addition, we should be able to leverage existing **cpuset.mem semantics** that use linux NUMA topology mapping.

Using the blocks in the CMS pooling solution, an example of a pooled memory that works in a k8s cluster could be as shown below:



1. During bring up, **appliance registers itself with k8s** master as a “MemoryClass” object
2. When POD spec is parsed, kube-sched will check if there are any restrictions on using only local memory. If there are no such restrictions, it implies the POD can handle a NUMA hop latency without violating service SLA.
3. If not, and none of the nodes have enough local memory, k8s sched will hotplug a slice from the appliance to one of the hosts, host device driver will invoke a hot plug event and it will show up under “memory only NUMA domain” (kernel managed memory)
4. Kubelet on that node will see memory in the updated node map and accept the pod spec for deployment on combined “memory only numa domain” + local memory
5. Once the pod is unallocated, memory is released back to the pool

6. AI/ML - Inference

AI workloads are used at scale across many applications and services and drive a portion of Hyperscale infrastructure (both HW and SW).

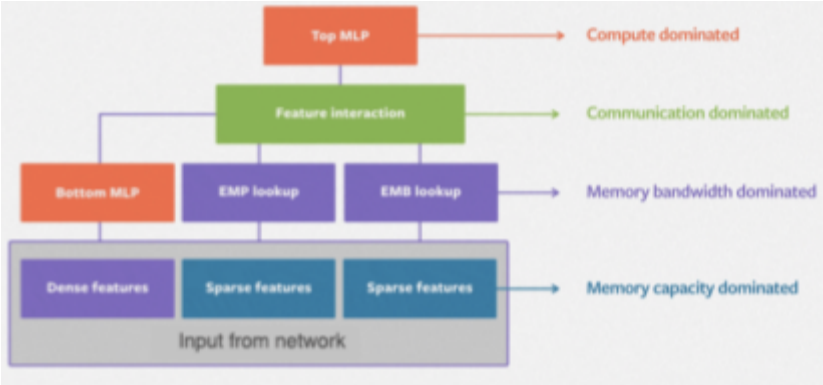


At a high level one can see the AI workloads fall into following categories:

1. Ranking and Recommendation (for example used to make video suggestions)
    a. Video suggestions, Ranking, Search, etc
2. Content understanding (e.g. computer vision such as face recognition in pictures)
    a. Image classification, Object Detection, Video Understanding, etc
3. Language (translation - text and speech, Content understanding, GenAI)

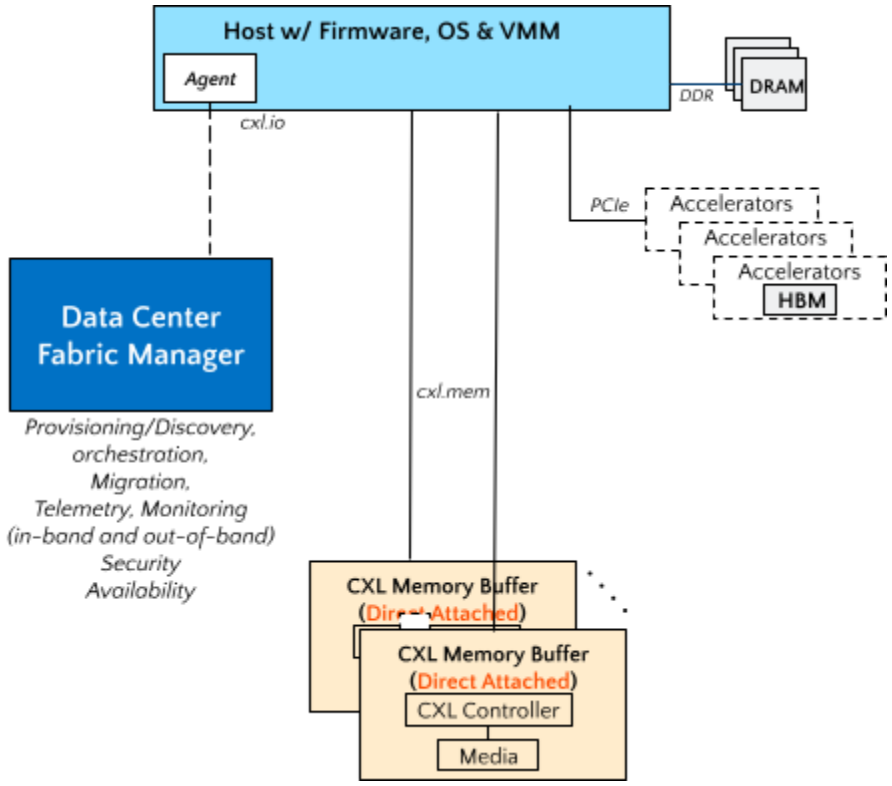
For this white paper we will focus on Ranking and Recommendation - Inference use case.

With the advent of deep learning, neural network-based recommendation models have emerged as an important tool for tackling personalization and recommendation tasks. These networks differ significantly from other deep learning networks due to their need to handle categorical features and are not well studied or understood. In this paper we will discuss the deep learning recommendation model (DLRM) and provide its impact on hardware choices specifically for memory capacity and bandwidth.



In the DLRM model, categorical features are processed using embeddings, while continuous features are processed with a bottom multilayer perceptron (MLP). Then, second-order interactions of different features are computed explicitly. Finally, the results are processed with a top MLP and fed into a sigmoid function in order to give a probability of a click.

As the above diagram shows DLRM has significant dependency on capacity of memories as well as memory bandwidth. As the amount of memory capacity available to accelerators through HBMs is limited, there is a need for providing a second tier of memory within the system. This memory can trade off performance as compared to HBM and DRAM, but with a tighter latency profile than NAND devices. This allows performance vs. density trade off for the system. CXL can fit well in this requirement through appropriate HW/SW design that can be validated with DLRM benchmarks at a system level.



Above logical architecture of the DLRM system discusses the use of CXL memory expansion modules connected to the host directly and accessed through CXL.mem. Accelerators can utilize SW solutions to tier the data between DRAM and CXL attached memory to provide enough scaling of capacity for the accelerators at acceptable performance.

As can be seen in this current implementation in the above picture, the accelerators are accessing the memories through PCIe. One of the solutions is to use tiering between DDR-DRAM and CXL-DRAM to reduce the impact of CXL latencies while achieving higher capacity.

7. Conclusion

Composability of memory is a key requirement for multiple use cases in the industry. Virtualization, containerization can benefit from pooling the capacity and avoiding stranding of expensive memory capacity and thereby enabling optimized business solutions for large hyperscale deployments. Caching/database use cases allow scaling of systems to allow maximum resource utilization of the servers to achieve the target application performance. This reduces overall cluster size for the given performance goals of the cluster and thereby improves the business case for such solutions. This requires next generation memory solutions that allow scaling of memory capacity and bandwidth. New memory interconnects like CXL are providing promise for such systems but will need co-design of hardware and software components to enable the end use cases and need to be validated with benchmarks for such workloads.

8. Glossary

Term/Acronym	Definition
AI	Artificial Intelligence
CXL	Compute Express Link, a low-latency, high-bandwidth link that supports dynamic protocol muxing of coherency, memory access, and I/O protocols, thus enabling attachment of coherent accelerators or memory devices.
CCI	Component Command Interface
CXL.io	PCIe-based non-coherent I/O protocol with enhancements for accelerator support
CXL.mem	Memory access protocol that supports device-attached memory.
DMTF	Distributed Management Task Force
HBM	High-Bandwidth Memory
HDM	Host-managed Device Memory. Device-attached memory that is mapped to system coherent address space and accessible to the Host using standard write-back semantics.
HMAT	Heterogeneous Memory Attribute Table as defined in ACPI Specification
ML	Machine Learning
PCIe	PCI Express
SLD	Single Logical Device
SRAT	System Resource Affinity Table as defined in ACPI Specification

9. References

1. Compute Express Link (CXL) 1.1/2.0/3.0 specifications: https://www.computeexpresslink.org/
2. Compute Express Link (CXL™) Memory Module Base Standard JESD317 www.jedec.org
3. Cachelib open-source software: https://cachelib.org/
4. Supporting Massive DLRM Inference Through Software Defined Memory
5. Pond: CXL-Based Memory Pooling Systems for Cloud Platforms

10. License

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.



OCP encourages participants to share their proposals, specifications and designs with the community. This is to promote openness and encourage continuous and open feedback. It is important to remember that by providing feedback for any such documents, whether in written or verbal form, that the contributor or the contributor's organization grants OCP and its members irrevocable right to use this feedback for any purpose without any further obligation.

It is acknowledged that any such documentation and any ancillary materials that are provided to OCP in connection with this document, including without limitation any white papers, articles, photographs, studies, diagrams, contact information (together, “Materials”) are made available under the Creative Commons Attribution-ShareAlike 4.0 International License found here: https://creativecommons.org/licenses/by-sa/4.0/, or any later version, and without limiting the foregoing, OCP may make the Materials available under such terms.

As a contributor to this document, all members represent that they have the authority to grant the rights and licenses herein. They further represent and warrant that the Materials do not and will not violate the copyrights or misappropriate the trade secret rights of any third party, including without limitation rights in intellectual property. The contributor(s) also represent that, to the extent the Materials include materials protected by copyright or trade secret rights that are owned or created by any third-party, they have obtained permission for its use consistent with the foregoing. They will provide OCP evidence of such permission upon OCP’s request. This document and any "Materials" are published on the respective project's wiki page and are open to the public in accordance with OCP's Bylaws and IP Policy. This can be found at http://www.opencompute.org/participate/legal-documents/. If you have any questions please contact OCP.

11. About Open Compute Foundation

At the core of the Open Compute Project (OCP) is its Community of hyperscale data center operators, joined by telecom and colocation providers and enterprise IT users, working with vendors to develop open innovations that, when embedded in products, are deployed from the cloud to the edge. The OCP Foundation is responsible for fostering and serving the OCP Community to meet the market and shape the future, taking hyperscale led innovations to everyone. Meeting the market is accomplished through open designs and best practices, and with data center facility and IT equipment embedding OCP Community-developed innovations for efficiency, at-scale operations and sustainability. Shaping the future includes investing in strategic initiatives that prepare the IT ecosystem for major changes, such as AI & ML, optics, advanced cooling techniques, and composable silicon. Learn more at www.opencompute.org.