White Paper

NVMe Over CXL Combines Storage with Memory

NVMe Over CXL™ Defines Memory Class Storage to Improve System Performance

By Bill Gervasi and San Chang

Abstract: CXL ended the fabric wars and introduced new challenges for optimizing the performance of these rapidly expanding system architectures that deploy CXL. CXL provides a great framework for unifying processing, memory, storage, and communications, and the possibility of combining functions in ways previously difficult or impossible to achieve. CXL is inevitably moving into other domains as well, such as artificial intelligence, automotive applications, and even notebook-class computing.

NVMe Over CXL merges storage and memory in ways that optimize traffic on the fabric, and also adds significant performance gains over standalone solutions deployed in traditional systems.

Introduction

The industry is in the process of adopting Compute Express Link, or CXL™, as the primary fabric for interconnecting a variety of processors, I/O resources, memory resources, and storage for data centers, hyperscalers, and similar computing clusters. Artificial intelligence is similarly soaring in popularity, and demanding massive data sets for the learning processes. It is likely to move into the next generation of automobiles as well as cars adopt data center technology. CXL allows mixing resources in a fashion that meets the needs of these systems, and flexibility for each end user to deploy a different mix.

CXL adds a family of light-weight protocols running on industry standard PCIe buses: CXL.io which encapsulates traditional PCIe protocols, CXL.cache which allows coherent memory resource sharing, and CXL.mem which adds a simpler memory-direct interface. Solid State Drives (SSDs) are the dominant non-volatile storage devices, and the Non-Volatile Memory Express (NVMe) protocol running on PCIe is the dominant software interface. Newly emerging are CXL Memory Modules (CMMs) which provide a bridge between the CXL.mem protocol and standard volatile memory such as DDR SDRAM.

NVMe Over CXL™ (NVMe-oC™) combines storage and memory into a unified CXL device that leverages both CXL.io and CXL.mem without breaking the massive volume of legacy software that runs on today's systems. NVMe-oC reduces traffic on system buses, improves performance, and provides the ability to make volatile memory persistent. NVMe-oC provides a "memory class storage", so to speak.

CXL Allows Flexible Mix-and-Match Design

In today's system architectures, storage and memory are distinct. This is an artifact of the early (nearly pre-historic) adoption of random access memory (RAM) as the medium for holding programs and data, and filesystems for paging data between RAM and non-volatile backup media. As shown in Figure 1, typical systems currently use DDR as the interface to memory and PCIe as the interface to storage. Programs execute from RAM on data also stored in RAM. Storage is essentially a paging operation to move programs and data from non-volatile media into the RAM.

RAM accesses have fine granularity that match processor caches with 64 bytes per cache line, and for this reason CXL primarily uses a 64-byte flow control unit (FLIT) as its transaction payload. Storage accesses, on the other hand, operate on large data blocks, typically 4 KB or larger.

Volatility versus non-volatility (or data persistence) is an essential aspect of this architecture as well. By definition, storage is non-volatile, meaning that if power fails, the media contents will not be lost. In contrast, RAM is volatile and its contents lost on power fail unless special mechanisms are deployed to maintain the contents of memory until power is restored. NVMe Over CXL will also address the issue of volatility.

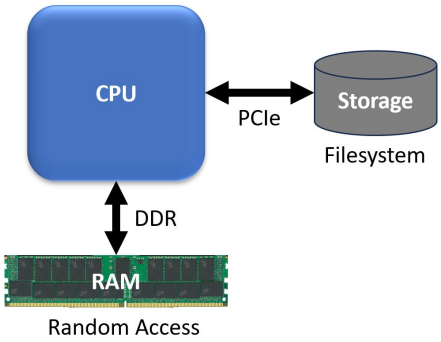

Figure 1: Typical Connectivity of CPU with RAM and Storage

CXL features a low pin count method for expanding system resources. Processing, storage, memory, and communications may be incrementally added with a unifying fabric allowing data flow between these resources. The picture in Figure 2 shows a small-scale simplified architecture; hyperscaler systems may deploy thousands of each class of node to create massively parallel processing environments.

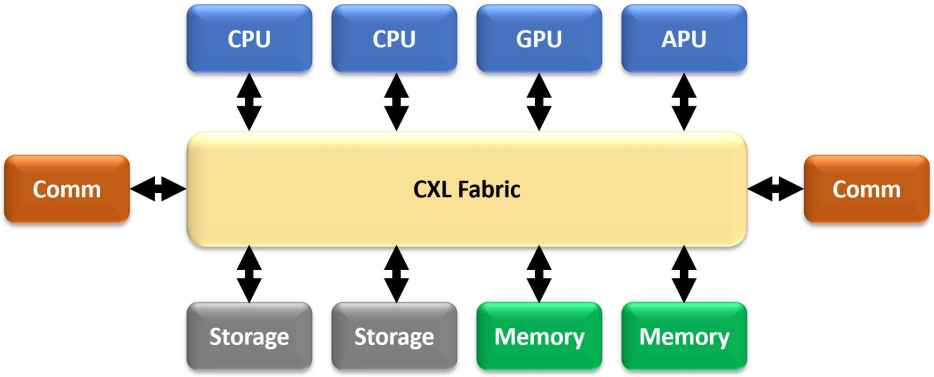

Figure 2: CXL-Based Server Architecture

The mix-and-match flexibility of CXL-based systems is a boon for system architects, but also has unique challenges. Coordinating large numbers of resources requires sophisticated mapping and control, and timing can become challenging as well. Among those challenges is the allocation of a storage connection and memory resource as a combined unit. As these are distinct operations, coordinating these resources takes time and potentially incurs race conditions in a fabric where multiple controllers may be vying for the shared resource.

NVMe and Controller Memory Buffers

NVMe specification 1.2 introduced the concept of a Controller Memory Buffer (CMB[1]). The CMB allowed the host controller to specify exact memory addresses for the source and target RAM buffers in NVMe read and write operations. The memory for storing data buffer or command queues can be resident in memory directly attached to the

[1] Since, as you know, engineers can only think in acronyms.

CPU (e.g., DDR) or may be located in the NVMe SSD (e.g., CMB) as illustrated in Figure 3.

Before initiating any block access operation, the host must allocate the CMB resource to validate and establish ownership of the required part of the storage system resources using PCIe transaction layer packet, also known as TLP. Once resource ownership is established, a host CPU may initiate a storage read operation using PCIe TLP to the storage device.

Traditionally, transferring a block of data from one SSD to another SSD involves using host DRAM as a staging buffer. With CMB, local memory can serve as a device buffer. As depicted in Figure 3, the SSD device decodes the logical to physical address of the read request to the attached Flash memory device and reads the corresponding Flash pages.

Since the target SSD equips with CMB, the Flash content can be read directly into the device buffer in a CMB using point-to-point (P2P) direct memory accesses (DMAs), which bypass the host DDR memory and reduce latency. Once the data is located in the CMB, either the host CPU can access the contents using PCIe memory read/write transactions or the device can process the data locally.

Write operations are the reverse of read operations. The CPU modifies the data stored in a CMB, then issues an NVMe write command, specifying the CMB memory address as part of the write operation. The SSD controller then uses P2P DMA to fetch the data stored in the CMB then transmits the content in the device buffer to the associated Flash locations.

A couple of observations are obvious here. Although the CPU can directly execute load/store instructions to access CMB content, the resulting PCIe memory transactions bypass the cache hierarchy, making the overall performance inefficient – these constraints limit the applications. As a result, practical examples only consider a CMB as a DMA buffer to facilitate block transfers between PCIe devices. A CMB cannot be efficiently utilized by the host CPU. This limitation, combined with the advent of CXL, motivates the proposal of a new architecture to address these problems.

Basically, CXL allows us to do CMB the right way!

NVMe Over CXL

NVMe Over CXL (NVMe-oC) combines the PCIe/CXL.io addressable NVMe logic with a CXL memory controller on the same device. As shown in Figure 4, this allows transfers between the Flash memory and the RAM to operate at the full speed of the attached Flash and RAM media devices. These DMA transfers can operate without consuming any time slots on the CXL fabric, which also avoids incremental protocol translation overhead in the transfers.

As with separate storage and memory modules, validation and ownership of the resources must be established prior to accessing either media, however with NVMe-oC this is performed using a single step, eliminating the potential race conditions of distinct operations. Once the resources are allocated, including the CMB which is located in the RAM address space on the NVMe-oC module, host operation may begin.

The procedure for read operations for NVMe-oC is compatible with legacy implementations that use CMBs. The host uses CXL.io to issue the NVMe commands to the NVMe-oC module. The logical to physical address translation is performed to access the associated Flash pages in the attached Flash memory device. Instead of copying to a local cache then moving data to host DRAM by DMA, however, the NVMe-oC controller also incorporates a RAM controller and may copy the contents of Flash directly into RAM at the assigned CMB address.

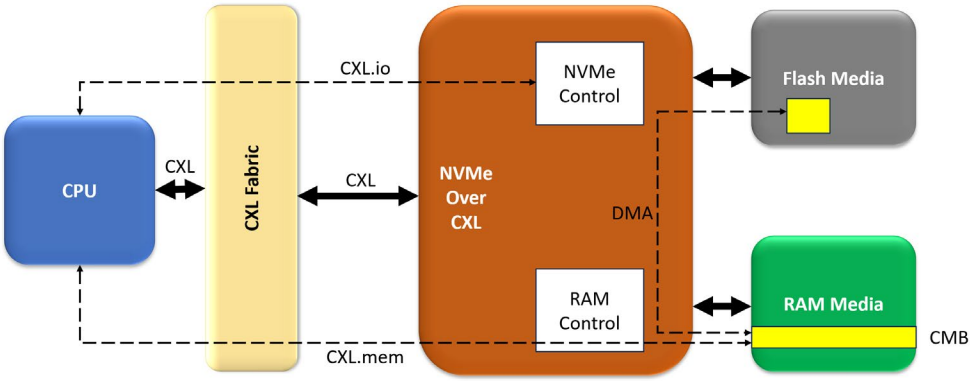

Figure 3: NVMe-oC Block Diagram

Once the contents of the Flash have been DMA transferred into the NVMe-oC RAM, the host may directly access the CMB using the CXL.mem protocol. Read and write operations access the RAM buffer which appear in the CPU address space. Each read or write operation between the host and the RAM is one FLIT or 64 bytes. Long FLITs with 256 bytes are also supported and treated as four short FLITs which may be striped across multiple RAM channels.

When the host CPU is ready to write the CMB contents back to the Flash, the reverse operation is performed. The NVMe write commands are issued to the NVMe-oC controller using CXL.io specifying the CMB address located on the NVMe-oC module. The NVMe-oC controller initiates DMAs from the RAM to the associated Flash locations.

NVMe Over CXL Advantages

While it is easy to visualize the NVMe-oC architecture, it is less clear… why? What does NVMe Over CXL improve over separate NVMe and CXL memory modules? The answer lies in the distinction between NVMe blocks and the number of bytes actually used out of these blocks to assess the efficiency of NVMe.

Example 1: Three RocksDB (KVS) use cases at Facebook[2] which deploy:

- UDB, a MySQL storage layer for social data graphs
- ZippyDB, a distributed key-value store
- UP2X, a distributed key-value store for artificial intelligence and machine learning services

As shown in Figure 5, the dominant operation is GET which performs a large number of key-value accesses to unsorted small pairs in different storage blocks.

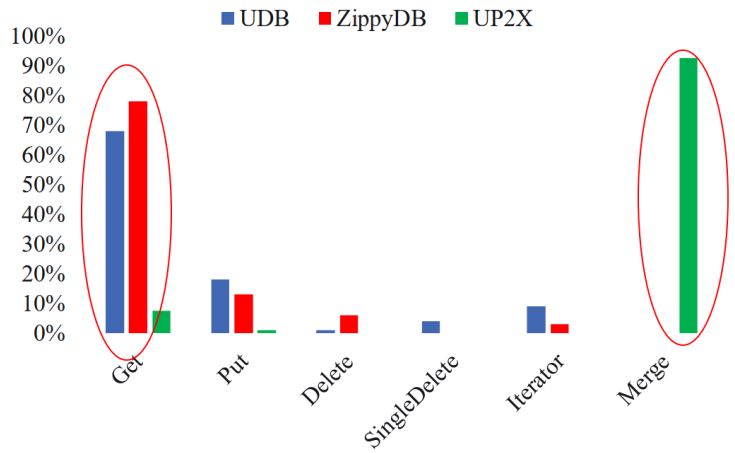

Figure 4: Operations Executed Using UDB, ZippyDB, UP2X

The resulting memory accesses to the CMBs are shown in Figure 6, which clearly highlights that one to two FLITs satisfy most requests. 128/4096 = 0.03… in other words, 97% of the transferred data is never used.

Figure 5: Access Sizes for UDB, ZippyDB, UP2X		
Case	Average Key Size (bytes)	Average Value Size (bytes)
UDB	27.1	126.7
ZippyDB	47.9	42.9
UP2X	10.45	46.8

Example 2: Twemcache use cases[3] at X (the artist formerly known at Twitter) shown in Figure 7, where the median object size is slightly over 100 bytes, also less than two FLITs.

[2] Cao, Zhichao, et al. "Characterizing, Modeling, and Benchmarking RocksDB Key-Value Workloads at Facebook" 18th USENIX Conference on File and Storage Technologies (FAST 20). 2020.

[3] *Yang, Juncheng, Yao Yue, and K. V. Rashmi. "A Large-scale Analysis of Hundreds of In-memory Key-value Cache Clusters at Twitter" ACM Transactions on Storage (TOS) 17.3 (2021): 1-35.*

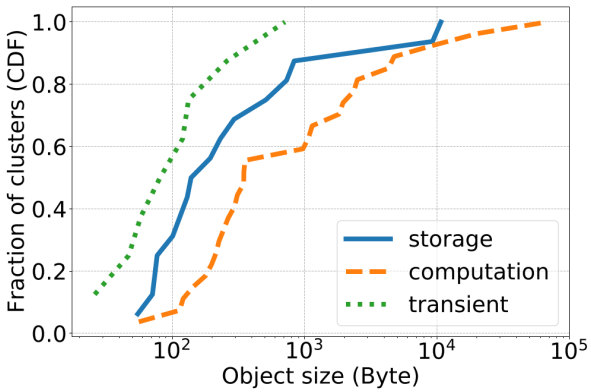

Figure 6: Twemcache, 153 Clusters

Example 3: fsync()[4] forces modified data in the open file's CMBs to be written to permanent storage, and executes a large number of small random read-modify-write operations. Accessed are blocked until data flush is complete. Figure 8 shows the significant impact of fsync() on system performance, primarily caused by the large number of blocks transferred while very little data is used in each block.

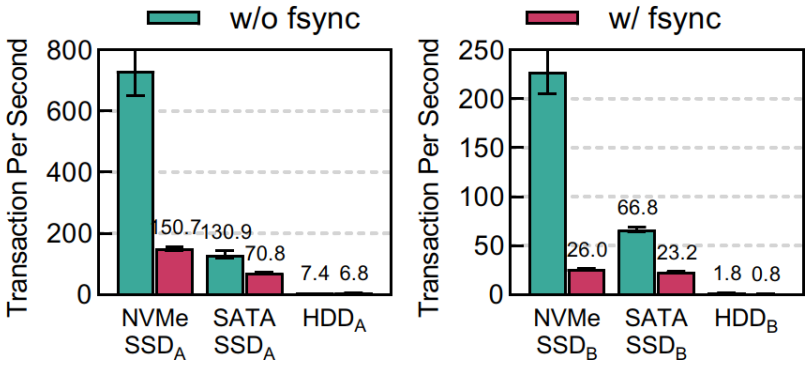

Figure 7: Performance Drop for MySQL on Aggressive fsync()

Data Flow with NVMe Over CXL

In general, the performance advantages of NVMe Over CXL compared to using individual NVMe storage modules are seen by comparing Figure 9 and Figure 10. In the first case, all traffic must engage the transfer of packets across the PCIe interface. Beyond the data itself, every command and response require that packets are exchanged across the PCIe, increasing traffic and adding to the latency of actual work.

Typically, traditional NVMe over PCIe demands a substantial effort to complete reading a block of data from an SSD, as illustrated Figure 9. In the initial step, the software driver prepares an IO block-read command in the submission queue and

[4] *He, Haochen, et al. "When Database Meets New Storage Devices: Understanding and Exposing Performance Mismatches via Configurations." Proceedings of the VLDB Endowment 16.7 (2023): 1712-1725.*

informs the device using the doorbell mechanism. Subsequently, the SSD device begins fetching commands from the host and processes them by reading block data from Flash. Once the block data is ready on the device, it must write back the block data to the host DRAM using multiple TLP memory write (MWr). After the data movement is complete, the device prepares the necessary entry for the completion queue and notifies the host after inserting the entry into the completion queue. Examining this complex flow makes it evident that reading a block of data involves numerous PCIe transactions between the host and the device. With more blocks being accessed simultaneously, this traffic will further exaggerate.

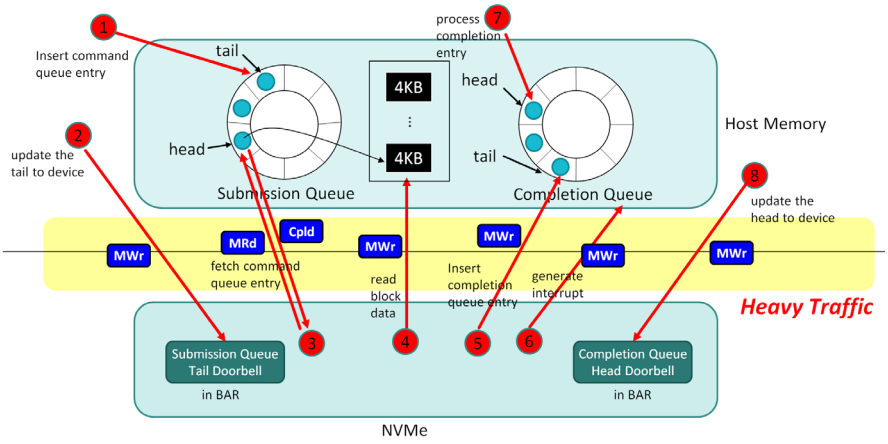

Figure 8: Traffic Jam Using Separate NVMe and CXL Memory Module

With NVMe Over CXL, the traffic is significantly reduced. Once the command has been issued to the NVMe-oC controller, the processing of the command can be done with minimal traffic on the PCIe. A significant difference lies in the allocation of the data buffer on the device memory, eliminating the need for block transfer between the host and the device. As the application can utilize CXL.mem to access any byte within those block data on device memory, there is no necessity to move those blocks to the host.

Further examining the flow of adopting the NVMe-oC architecture, the device memory allows the host to store the submission queue in the controller side. This configuration reduces the latency of command fetching, as each command is directly inserted into the device, and the device can decode the command locally.

Data transfer from Flash to RAM occurs in blocks, e.g., 4 KB, however this is accomplished at the internal data rates of the Flash and RAM interfaces. When the host needs to consume the data that has been read, it accesses the data 64 bytes at a time using CXL.mem FLITs. Importantly, this scheme introduces no modification to the NVMe flow, and the reduced traffic contributes to minimizing queueing delays for each PCIe transaction, thereby improving per-command Quality of Service (QoS).

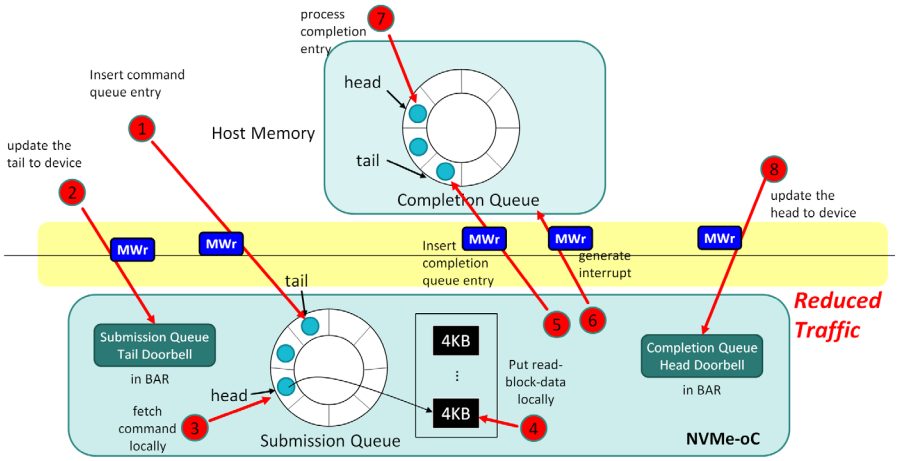

Figure 9: Traffic Relief Using NVMe Over CXL

Critical Word Access Feature

Critical word access has been a feature of cache fill operations for a long time. The idea is to transfer the data most likely to be accessed first and allow it to be processed while the rest of the cache is still being filled. NVMe-oC supports critical word access by tracking the status of the DMA from Flash to RAM and comparing this to incoming data access requests from the CXL.mem interface. If the memory address requested by the CXL.mem access has been successfully transferred, the access is allowed to complete even while the DMA from Flash continues for other bytes in the block. If the requested CXL.mem address has not been DMA transferred yet, the CXL.mem request is held off until that data is valid in RAM, and then the transfer completes.

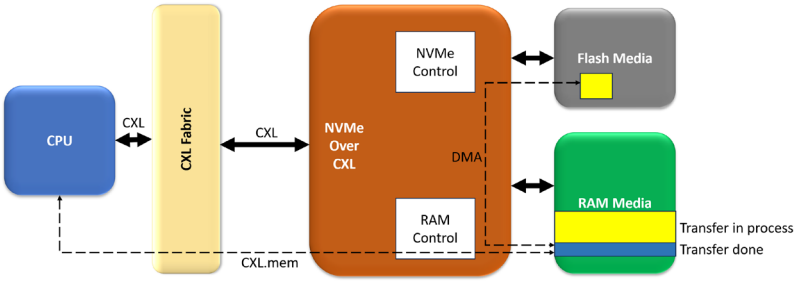

Figure 10: NVMe-OC Critical Word Access

Data Persistence Feature

The lack of data persistence is the key weakness of RAM. When power fails, the data stored in RAM is lost. NVMe-oC controllers are capable of providing persistence support to RAM contents, however. An external sideband signal, Power Failing, provides a warning from the host system if power failure is imminent. When Power Failing is asserted, the NVMe-oC controller completes any FLIT in progress then disables the PCIe interface. The entire content of RAM is written into the Flash media. When power is restored, the NVMe-oC controller transfers the stored image from Flash back to RAM before signaling to the host that it is ready for operation. Depending on the Power Failing latency to actual power down and the time required

to save RAM to Flash, the module may require an energy source to power the NVMe-oC logic until the operation completes.

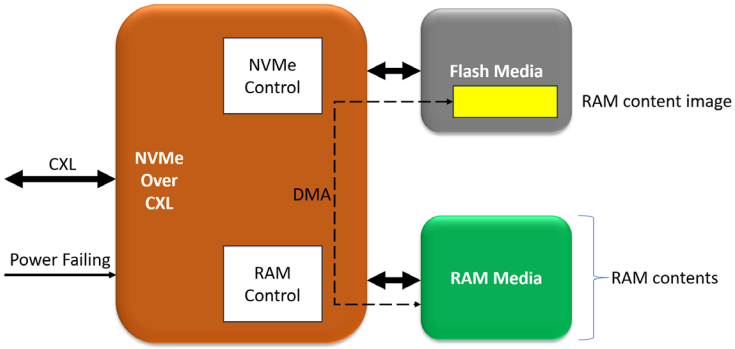

Figure 11: RAM Data Persistence Function

Software support for data persistence leverages the Byte Addressable Energy Backed Interface procedures to coordinate host and device validation of the RAM image.[5]

The data persistence feature makes NVMe Over CXL attractive for automotive applications. Current automobile designs deploy many system-on-a-chip (SoC) subsystems, however the weak link is that each SoC has a slow serial boot ROM chip to reload code. If power glitches while the car is running, say from a lightning strike impulse, the reboot time of the vehicle can be measured in seconds. Automotive fabrics already connect these SoCs to a shared SSD resource over PCIe. Adding CXL logic onto this PCIe interface, and automatically reloading critical SoC boot code from Flash into in a shared RAM space, can give these vehicles nearly instant-on capability. NVMe-oC can literally save lives.

NVMe Over CXL Also Supports Legacy Use Modes

This white paper focuses on the unique features enabled by co-locating Flash and RAM on the same module; however, these subsystems are entirely backward compatible separately as well. For example, the CMB for the NVMe functions need not be located in the NVMe-oC RAM, but may be anywhere in the processor addressing space including direct attached DRAM or CXL memory modules. Similarly, while the RAM located on the NVMe-oC module may be allocated for use as CMBs, it is just RAM what can be accessed and used like any other CXL memory module.

NVMe Over CXL Compared to Memory Semantic SSD

Memory semantic solid-state drives (MS-SSDs) are a different way to merge Flash and RAM. While, from a high-level view, MS-SSDs look similar to NVMe Over CXL, they are actually quite different. MS-SSD provides RAM accessible over CXL.mem,

[5] Byte Addressable Energy Backed Interface (BAEBI): https://www.jedec.org/system/files/docs/JESD245E.pdf

however the memory footprint of the MS-SSD is expanded by the addition of Flash on the module. MS-SSDs use target-directed cache strategies to page contents between RAM and Flash: if a CXL.mem requests comes in and the content is in RAM, the request is honored at CXL.mem speeds. However, if the content is not in RAM, the CXL.mem request is held off while the MS-SSD controller writes the contents of some RAM block out to the Flash, remaps that memory block for the CXL.mem request, and loads the RAM contents of that requested memory block from Flash into RAM before completing the request.

This highlights the primary difference between NVMe-OC and MS-SSD. The MS-SSD CXL.mem requests can be delayed at any time if the on-module cache algorithm causes a page miss. Since applications running and using this memory are not aware of these delays, they either freeze in the active application queue until the swap is complete or they lose their slot and are put back onto the pending process list.

NVMe-oC distinguishes between the filesystem access and memory access methods cleanly, so running applications do not hang waiting for resources. Even when critical word access mode is used, the delay is only until DMA of the requested word completes, not like the much longer MS-SSD cache flush and cache fill operation completes.

In short, MS-SSD cache strategies are limited to the algorithms on the MS-SSD module, whereas NVMe-oC cache strategies are host-driven and trust the host to perform optimizations best for their applications.

This distinction also highlights another fundamental difference between these approaches. NVMe-oC works with all existing applications that use filesystem access mechanisms. MS-SSD typically requires software changes to utilize direct memory access, such as memory-mapped files.

Conclusion

Storage and memory have traditionally been distinct and separated in computer architectures. However, with the advent of CXL as a unifying fabric, new possibilities have emerged in how these essential parts are treated within future data processing systems. NVMe Over CXL combines the best of storage and direct memory access methods into a holistic solution that allows existing software to use these resources without change. NVMe-oC not only provides higher performance but also reduces the system traffic over its fabric. Moreover, the inclusion of data persistence for memory enabled by CXL paves the way for a future where instant-on features, commonly found in appliances, can now be extended to data centers and other computing equipment.

NVMe Over CXL is simple in concept yet powerful in results.