

CMS Workloads and Benchmarks

Version 1.0
Date: Oct 17, 2023

Author (s) :
Nara Vijay, Meta
Vikrant Soman, Uber

Reviewer (s):
Anjaneya Reddy Chagam, Intel Corporation
Manoj Wadekar, Meta

Executive Summary

This document defines the **Open Compute Project (OCP) Composable Memory Systems (CMS)** Workloads and Benchmark proposal. CMS is an emerging paradigm that facilitates dynamic and unified memory management across diverse memory technologies, interconnects, and hierarchies. To evaluate the performance of these emerging technologies across industries relevant to real-world applications, standard mechanisms are essential. This document proposes workloads and benchmarks that are representative of the use cases to enable evaluation of various HW/SW technologies for Composable Memory systems solutions. These benchmarks will allow researchers, developers, and industry to evaluate the performance of Composable Memory systems solutions.

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. Workloads, Benchmarks, Metrics and Performance Targets **6**
Component Benchmarks **7**
Memory Latency Checker (MLC) 7
Benchmark recipe 7
How to run the benchmark 7
System Benchmarks **8**
SPECCPU2017 8
Benchmark recipe 8
How to Run the benchmark: 10
What to measure: monitoring/metrics 12
Memcache system benchmark - Cachebench 13
Benchmark recipe 14
How to Run the benchmark 16
What to measure: Monitoring and Metrics 16
4. Conclusion **20**
5. References **21**
6. License **22**
7. About Open Compute Foundation **23**

1. Compliance with OCP Tenets

1.1 Openness

The OCP CMS Workloads and Benchmarks were developed in dialogue with the entire OCP CMS community membership and the resulting document represents the initial set of workloads and benchmarks for developing composable memory solutions. CMS workloads/benchmarks are nascent, so this document will evolve over time, and that this document represents the wisdom of the collective ecosystem for creating CXL based composable memory systems.

1.2 Efficiency

The OCP CMS Workloads and Benchmarks document lays the foundation for developing and validating hardware/software co-designed composable memory solutions to meet cloud service provider needs.

1.3 Impact

Having a common set of Workloads and Benchmarks helps us in guiding CMS workstream priorities for developing open composable memory solutions to meet cloud server provider needs.

1.4 Scale

CMS Workloads and Benchmarks enable interoperable solutions using open eco-system software, hardware and best practices to achieve faster time to market.

1.5 Sustainability

CMS Workloads and Benchmarks are designed to optimize performance and resource utilization for underlying physical memory technologies thereby demonstrating 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 and capacity needs. Memory technologies must be able to support the growing amount of data and compute needs. The industry is currently developing diverse new and evolving memory technologies, interconnects & hierarchies, which provides Total Cost of Ownership (TCO) optimized memory capacity, latency, throughput, sharing, and scalability for data intensive applications. These hardware innovations merged with appropriate usage provides better performance and optimum resource utilization of the underlying physical memories. Some applications continue to demand higher memory capacity (driven by constant increase in compute performance). Other 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/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, and enable pooled memory with unified namespaces.

During Global OCP 2022, the Open Compute Project Foundation (OCP) announced that formal work has been started under the Composable Memory Systems (CMS) Sub Project. In this white paper we will explore workloads and benchmarks that help evaluate various HW and SW technologies for Composable Memory systems solutions.

3. Workloads, Benchmarks, Metrics and Performance Targets

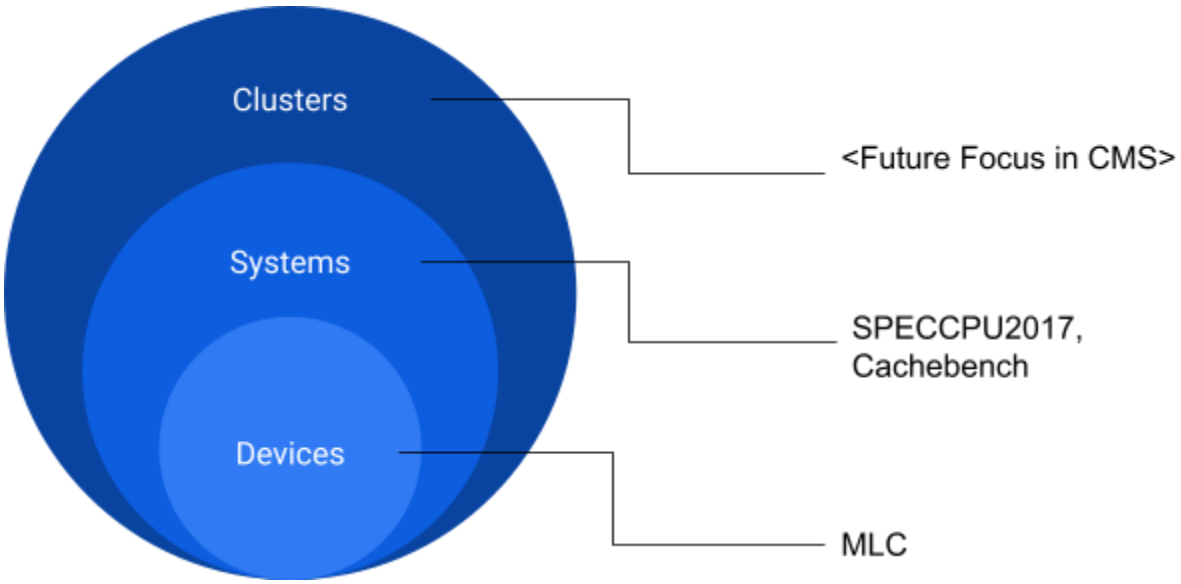


Figure 1: Benchmark Categories

Benchmarks have been divided into three categories from setup and test focus perspective.

Component benchmarks: They focus on a specific component's behavior without considering stress or other components of the system. For example, in this document the "memory latency checker" focuses solely on memory performance without considering CPU, Storage, or Network utilization.

System benchmarks: These focus on a part of the system that the application uses in a node level implementation. This expands the focus on all the system components that application is expected to use when deployed in single node instantiation. For example, the Cachebench benchmark discussed in this document uses Cachelib, a real world application that represents single system instantiation. However this is not taking network into consideration that will be an important factor when Cachelib is used by Memcache in a clustered fashion.

Cluster benchmarks: These bring benchmarks to near practical deployment scenarios, where multiple machines are tested in a networked environment and the whole cluster performance is considered. A future example, while not included in this document, might be Memcached tested with multiple servers and clients.

Component Benchmarks

Component benchmarks allow testing of memory for throughput and latency for varying access sizes and mix (read/write).

Memory Latency Checker (MLC)

Intel® Memory Latency Checker (MLC) is a tool used to measure memory latencies, bandwidth and how these change with increasing load on the system. It also provides several options for more fine-grained investigation where bandwidth and latencies from a specific set of cores to caches or memory can be measured as well. The mlc v3.10 also added a new feature to track the latency of each request and provide a histogram of latency distributions. This can be found here.

Benchmark recipe

For a system with Far Memory enabled with CXL, it shows up as a remote NUMA memory to the operating system. MLC program is used for getting the performance data for this Far memory. (Numactl lets administrators run a process with a specified scheduling or memory placement policy. Numactl can also set a persistent policy for shared memory segments or files, and set the processor affinity and memory affinity of a process).

How to run the benchmark

Example Commands:

1. **Max Bandwidth**: `numactl --cpubind=<numa_node> --membind=<numa_node> ./mlc --max_bandwidth`
2. **Idle Latency**: `numactl --cpubind=<numa_node> --membind=<numa_node> ./mlc --idle_latency`
3. **Idle Latency with histogram:** `numactl --cpubind=<numa_node> --membind=<numa_node> ./mlc -idle_latency -x0 -f3`
4. **Loaded Latency**: `numactl --cpubind=<numa_node> --membind=<numa_node> ./mlc --loaded_latency -b102400 -K1 -W2 -U`
   `[100MB buffer size, 64B load/store lines, random access 2:1 read/write ratio]`

What to measure: monitoring and metrics

	Local NUMA	CXL NUMA
Idle Latency (ns)	100	250
Max Read Bandwidth (GB/s)	300	27
Max 2:1 Read Bandwidth (GB/s)	270	24
STREAM Triad Bandwidth (GB/s)	261	21
Loaded Latency 0 inject delay (ns)	400	1200

Table 1: mlc measurement (Example data)

System Benchmarks

System benchmarks cover as many components in the system as possible (E.g. CPU, Memory, SSD, HDDs etc.).

SPECCPU2017

We can use well known industry standard benchmark components within the SPEC CPU 2017 suite allowing us to quickly create various templates that exercise the memory subsystem in a given host. The **memory and cache bound** subcomponents would be relevant for studying the memory pooling solution. Based on the published studies for various SPEC CPU 2017, we can shortlist the following subcomponents that target specific resources of the cache and memory hierarchy.

Subcomponent	**Memory component stressed**	**Characteristics**
x264	L1$	High IPC and low backend bound
xalancbmk	L2/L3$	Medium IPC and backend bound
omnetapp	Main memory (DRAM DIMM)	Low IPC and high backend bound

Table 2: Memory and cache bound Subcomponents

Benchmark recipe

There are 2 main goal of the benchmarking recipe

1. Come up with a fixed set of templates using SPECCPU2017 subcomponents to mimic container behavior on a single machine and stressing different parts of the cache/memory subsystem concurrently. Ex - some containers might stress the L1 local cache while another container colocated on an adjacent core might stress the DRAM main memory. See figure 3 below.
2. Measure the relative performance metric differences (score/latency/BW) for these templates when pinned to local memory vs pinned to CXL remote memory.

The following templates should cover the requirements mentioned above

	Template 1 (L1 cache)		**Template 2 (L2/L3 cache)**		**Template 3 (DRAM)**	
	Core 0 …… Core N		**Core 0 ….. Core N**		**Core 0 ….. Core N**	
	SMT0	**SMT 1**	**SMT 0**	**SMT 1**	**SMT 0**	**SMT 1**
Local only	x264	x264	xalancbmk	xalancbmk	omnetapp	omnetapp
Remote CXL memory only	x264	x264	xalancbmk	xalancbmk	omnetapp	omnetapp

Table 3

	Template 4 (L1 + L2/L3 cache)		**Template 5 (L2/L3 cache + DRAM)**		**Template 6 (L1 cache + DRAM)**	
	Core 0 …… Core N		**Core 0 ….. Core N**		**Core 0 ….. Core N**	
	SMT0	**SMT 1**	**SMT 0**	**SMT 1**	**SMT 0**	**SMT 1**
Local only	x264	xalancbmk	xalancbmk	omnetapp	x264	omnetapp
Remote CXL memory only	x264	xalancbmk	xalancbmk	omnetapp	x264	omnetapp

Table 4

Visually the above stencils can be outlined as

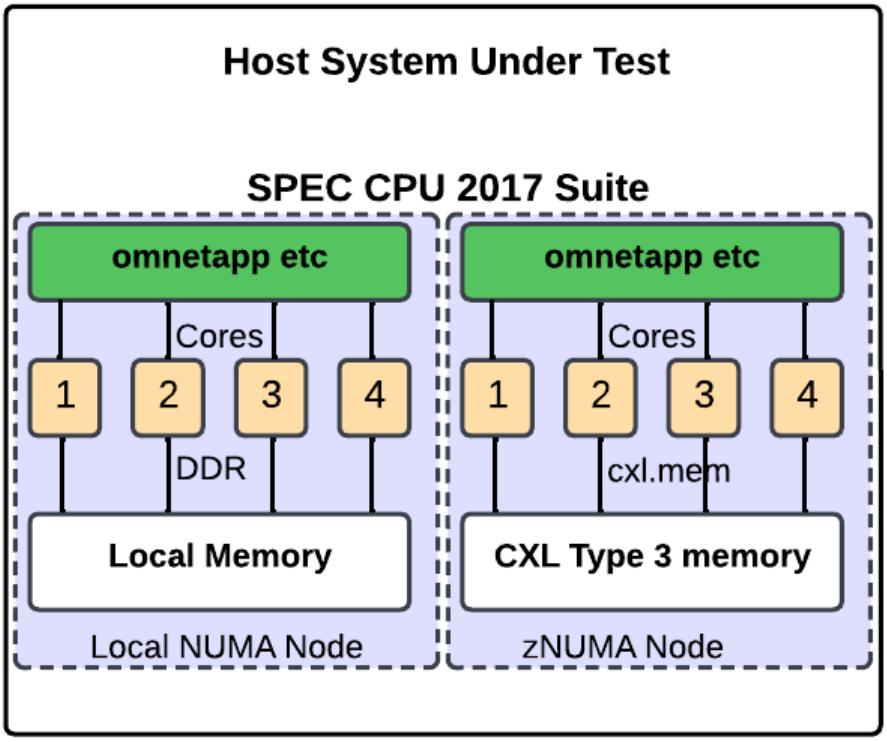


Figure 3

How to Run the benchmark:

Sample commands:

A. Running 48 copies of omnetapp

```
runcpu --configfile combo_0.cfg --noreportable --action onlyrun --iterations 3
--copies 48 --nopower --runmode rate --tune base  --size=refrate --nobuild
omnetpp_r
```

B. Running 24 copies of omnetapp + 24 copies of x264

```
runcpu --configfile combo_0.cfg --noreportable --action onlyrun --iterations 3
--copies 24 --nopower --runmode rate --tune base  --size=refrate --nobuild
x264

runcpu --configfile combo_0.cfg --noreportable --action onlyrun --iterations 3
--copies 24 --nopower --runmode rate --tune base  --size=refrate --nobuild
omnetpp_r
```

1. Enable SMT on the system if applicable
2. For template 1,2,3

a. set up number of copies for each subcomponent equal to the number of vcpus in the socket
b. Pin a copy to either SMT0 or SMT1 (since its the same workload should not matter)
c. Pin memory usage to local DRAM
d. Start the benchmark run and record scores and metrics mentioned in the section below
e. Once the benchmark ends pin memory usage to CXL remote memory
f. Start the benchmark run and record scores and metrics mentioned in the section below
g. Compare scores and metrics in steps d and f to assess CXL memory vs local memory performance differences

3. For template 3,4,5
    a. set up a number of copies of x264, omnetapp and xalancbmk such that each one equals 50% of the total virtual cores. Ex - in a 100 core machine, we will setup 50 copies of x264 and 50% xalancbmk
    b. Pin all copies of a subcomponent to SMT0 or SMT1 of all cores (as described in the table)
    c. Pin memory usage to local DRAM
    d. Start both subcomponents and end the run when the first subcomponent finishes. Capture the reported scores and metrics mentioned in the section below
    e. Now pin memory usage to CXL remote memory
    f. Start both subcomponents and end the run when the first subcomponent finishes. Capture the reported scores and metrics mentioned in the section below
    g. Compare scores and metrics in steps d and f to assess CXL memory vs local memory performance differences

What to measure: monitoring/metrics

Along with SPECCPU2017 score, some of the metrics that are emitted by the Intel telemetry collector (marked in asterisk) can be applicable for this benchmark.

Metric	Local Memory	CXL type 3 device	Local & CXL memory (OS tiering)
SPECCPU2017 int rate			
CPI*			
Average LLC data read miss latency (ns)*			
Memory Read Bandwidth (MB/s)*			
Memory Write Bandwidth (MB/s) *			
Memory Total Bandwidth (MB/s)*			

Table 5: Example table for SPEC CPU 2017 run

Memcache system benchmark - Cachebench

Caching use case has published an open source benchmark to test Cachelib. It's called Cachebench.

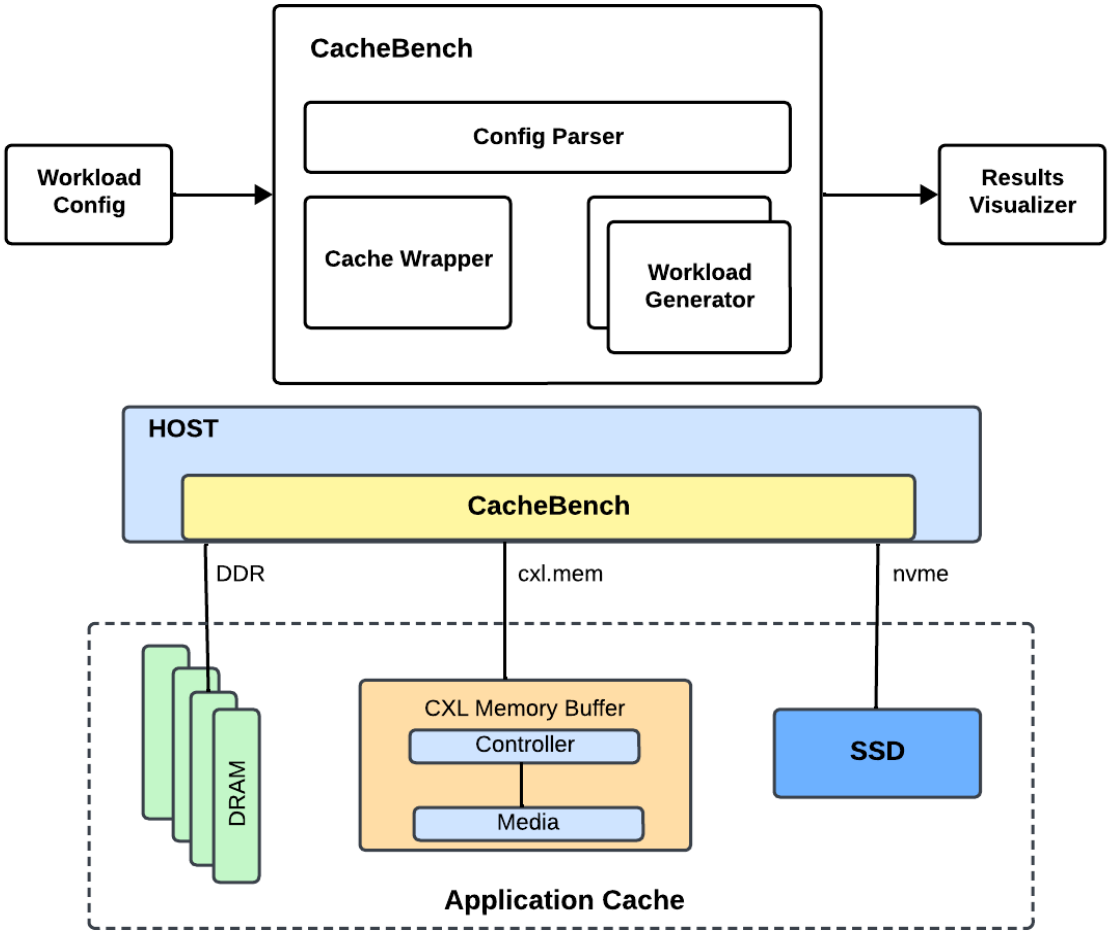


Figure 3

CacheBench is a benchmark and stress testing tool to evaluate cache performance with real hardware and real cache workloads. CacheBench takes in a configuration that describes the cache workload and the cache configuration and simulates the cache behavior by instantiating a CacheLib cache. It runs the workload and emits results periodically and at the end. The results include metrics such as hit rate, evictions, write rate to flash cache, latency, etc. The workload configs can be hand-written by a human, produced by a workload analyzer, or backed by raw production cachelib traces. The main customization points into CacheBench are through writing workload configs or custom workload generators.

Benchmark recipe

DUT HW Configuration format:

	System Configuration	Example
CPU	Physical Core(Virtual Cores), L3 Cache size, Core freq, Turbo enabled freq	20 Cores(40 vcores), 30MB L3 Cache, 1.8GHz (3GHz Turbo) Optional: Intel, AMD, ARM etc.
System Memory	Memory Capacity (Channels, DIMM Quantity * DIMM size), Memory Technology	128GB (4*32GB), 4800MTs DDR5
CXL Memory	Memory Capacity (Channels, DIMM Quantity * DIMM size)	128GB (4*32GB), 3200MTs DDR4
Storage	Storage capacity used for Cachebench if any, Storage technology	900GB NVMe SSD, PCIe Gen5x4

Table 6: System configuration reporting

DUT SW Configuration:

- Operating Systems
  - Linux: CentOS 5.15+ kernel
- Device Drivers
  - Linux Auto-tiering and DAX device enabled modes (<Example pointer to TPP>)
- BIOS support
  - Memory Interleaving BIOS options
  - Interleaving granularity optimized for performance
- NOTE: Cachebench config by default supports only 256GB Cache capacity allocation

Cachebench Workload Profiles

We can configure different workload profiles such as graph cache (leader and follower), key value cache, flat cache, etc. with Cachebench using configuration files. The workload characteristics can be modified or adjusted in the configuration file.

Example of the configuration file with Cache and NVM Cache:

```
{
 "cache_config": {
 "cacheSizeMB": 104857,
 "rebalanceStrategy": "tail-age",
 "allocFactor": 1.08,
 "maxAllocSize": 524288,
 "minAllocSize": 64,
 "htBucketPower": 27,
 "navyReaderThreads": 32,
```

```
"navyWriterThreads": 32,
"nvmCacheSizeMB": 131072,
"nvmCachePaths": ["/dev/nvme1n1p1"],
"writeAmpDeviceList": [
"nvme1n1"
],
"navyBigHashSizePct": 60,
"navyBigHashBucketSize": 4096,
"navyBlockSize": 4096,
"navyParcelMemoryMB": 6048,
"enableChainedItem": true,
"moveOnSlabRelease": true,
"poolRebalanceIntervalSec": 5
},
"test_config": {
"enableLookaside": true,
"generator": "online",
"numKeys": 10000000000,
"numOps": 5000000,
"numThreads": 384,
"poolDistributions": [
{
"addChainedRatio": 0.0,
"delRatio": 0.0,
"getRatio": 0.9845,
"keySizeRange": [
16,
255
],
"keySizeRangeProbability": [
1.0
],
"loneGetRatio": 1.0426679936986586e-05,
"loneSetRatio": 0.00317,
      "popDistFile": "pop.json",
"setRatio": 0,
"valSizeDistFile": "sizes.json"
}
],
"opDelayNs": 0,
"opDelayBatch": 25
}
```

How to Run the benchmark

1. Use the Installation | CacheLib instructions to install cachebench on the system under test
2. In order to standardize the data output, please use the following cachebench command format to run the benchmark:

```
./cachebench --json_test_config {workload file.json} --progress_stats_file {output_file.log} --report_api_latency
```

What to measure: Monitoring and Metrics

It is essential to capture the necessary configuration and metrics before the test execution.

CacheBench Metrics to Capture

Metric	Description	Example
Cachebench Revision	Revision of Cachebench used and any modifications to the revision	v2023.09.25.00
Cache size in GB	Local DRAM allocated size for Cachebench	100GB
CXL Cache Size in GB	CXL module or DRAM allocated size for Cachebench	100GB
NVM Cache size in GB	SSD cache allocated size if any	60GB
Number of Keys	To adjust the working set size of the cache, you can increase or decrease the numKeys in the cache	1000000
Number of Operations	Number of operations, To run cachebench operation for longer, increase the numOps appropriately in the config file	5000000
Number of Threads	Number of benchmark threads, Running with more threads should increase throughput until you run out of cpu or hit other bottlenecks	64

Table 7: Important Metrics

CacheBench Metrics to Monitor

- CPU Utilization
- Memory Utilization (Native and CXL)
- Memory Bandwidth Utilization (Native and CXL)
- Storage SSD Bandwidth Utilization

CacheBench Reporting Format

The following measurements are essential to be captured:

Measurement	**Description**	**Example**
Hit Ratio	Cache Hit ratio	58%
Number of Keys	Number of keys stored/processed in the cache	1billion
Total Ops	This is usually the product of numOps * numThreads from config file that will be part of the output log	1920 Million
Get throughput	Overall Cache read throughput, this include local, cxl and nvm cache if used	12 million ops/s
Set throughput	Overall Cache write throughput, this include local, cxl and nvm cache if used	4 million ops/s
Del throughput	Overall Cache delete operation throughput if any	1.2 million op/s
Cache Find P99 latency	DRAM and/or CXL memory P99 read latency	300 us
Cache allocate P99 latency	DRAM and/or CXL memory P99 write latency	1200 us
NVM Read P99 latency	NVM cache (SSD) P99 read latency	2 ms
NVM Write P99 latency	NVM cache (SSD) P99 write latency	7 ms

Table 8: Measurements to Report

Example Output:

```
01:55:01       6.72M ops completed. Hit Ratio  33.72% (RAM  31.71%, NVM
2.95%)
== Allocator Stats ==
Items in RAM  : 3,133,710
Items in NVM  : 1,442,247
Alloc Attempts: 4,595,599 Success: 100.00%
Evict Attempts: 1,461,720 Success: 99.98%
RAM Evictions : 1,461,376
Fraction of pool 0 used : 1.00
Cache Gets    : 6,699,076
Hit Ratio     :  33.72%
Cache Find API latency p50      :    3018.00 ns
Cache Find API latency p90      :   14019.00 ns
Cache Find API latency p99      :  383262.00 ns
Cache Find API latency p999     : 1051519739.00 ns
Cache Find API latency p9999    : 4079075148.00 ns
Cache Find API latency p99999   : 5056456047.00 ns
Cache Find API latency p999999  : 5154194137.00 ns
Cache Find API latency p100     : 12535803063.00 ns
Cache Allocate API latency p50      :    8989.00 ns
Cache Allocate API latency p90      :   30009.00 ns
Cache Allocate API latency p99      :  596658.00 ns
Cache Allocate API latency p999     : 14616067.00 ns
Cache Allocate API latency p9999    : 43741472.00 ns
Cache Allocate API latency p99999   : 50133597.00 ns
Cache Allocate API latency p999999  : 50772809.00 ns
Cache Allocate API latency p100     : 71162600.00 ns
RAM Hit Ratio :  31.71%
NVM Hit Ratio :   2.95%
RAM eviction rejects expiry : 0
RAM eviction rejects clean : 0
NVM Read  Latency    p50      :     124.00 us
NVM Read  Latency    p90      :     416.00 us
NVM Read  Latency    p99      :    2091.00 us
NVM Read  Latency    p999     :    7425.00 us
NVM Read  Latency    p9999    :    8916.00 us
NVM Read  Latency    p99999   :    9407.00 us
NVM Read  Latency    p999999  :    9456.00 us
NVM Read  Latency    p100     :       0.00 us
NVM Write Latency    p50      :     762.00 us
NVM Write Latency    p90      :     822.00 us
```

```
NVM Write Latency    p99      :    7044.00 us
NVM Write Latency    p999     :   11273.00 us
NVM Write Latency    p9999    :   15943.00 us
NVM Write Latency    p99999   :   17384.00 us
NVM Write Latency    p999999  :   17528.00 us
NVM Write Latency    p100     :       0.00 us
NVM bytes written (physical)  :  48.18 GB
NVM bytes written (logical)   :  47.43 GB
NVM bytes written (nand)      :   0.00 GB
NVM app write amplification   :   1.02
NVM dev write amplification   :   0.00
NVM Gets      :       4,574,896, Coalesced :   0.01%
NVM Puts      :       1,461,504, Success   : 100.00%, Clean   :   0.00%,
AbortsFromDel   :         0, AbortsFromGet   :         1
NVM Evicts    :                0, Clean     :   0.00%, Unclean :       0,
Double          :         0
NVM Deletes   :       4,461,193 Skipped Deletes:  76.18%
Total eviction executed 1
== Hit Ratio Stats Since Last ==
Cache Gets    : 6,699,075
Hit Ratio     :  33.72%
RAM Hit Ratio :  31.71%
NVM Hit Ratio :   2.95%
== Throughput Stats ==
Total Ops : 6.72 million
Total sets: 4,461,279
get       :   111,584/s, success   :  33.72%
couldExist:         0/s, success   :   0.00%
set       :    74,311/s, success   : 100.00%
del       :         0/s, found     :   0.00%
```

4. Conclusion

A performance evaluation of the use cases for composable memory solutions is imperative. By using standard workloads and benchmarks, we can better understand the advantages of new memory interconnects such as CXL, enabling the industry to explore and validate new technologies. Furthermore, we should invest in optimizing the mentioned benchmarks with various software and kernels as well as adding relevant AI use case benchmarks that are driving memory and bandwidth requirements. Therefore, we intend to focus on the following for next revision of the white paper:

1. Cachebench
    a. Gather performance insights with TPP kernel
    b. Performance analysis with different CXL memory modules
    c. Publish Test Results
2. AI benchmark
    a. Develop and add relevant AI benchmarks to the document
3. Device power
    a. Add another measurement metric for power/performance efficiency

5. References

1. Intel® Memory Latency Checker v3.9a
2. SPEC CPU® 2017
3. CacheLib
4. CacheLib/BENCHMARKS.md at main
5. [2206.02878] TPP: Transparent Page Placement for CXL-Enabled Tiered-Memory

6. 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.

7. 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.