

MVAPICH Collectives on Storage Class Memories: Early Experiences

Talk by: Mustafa Abduljabbar

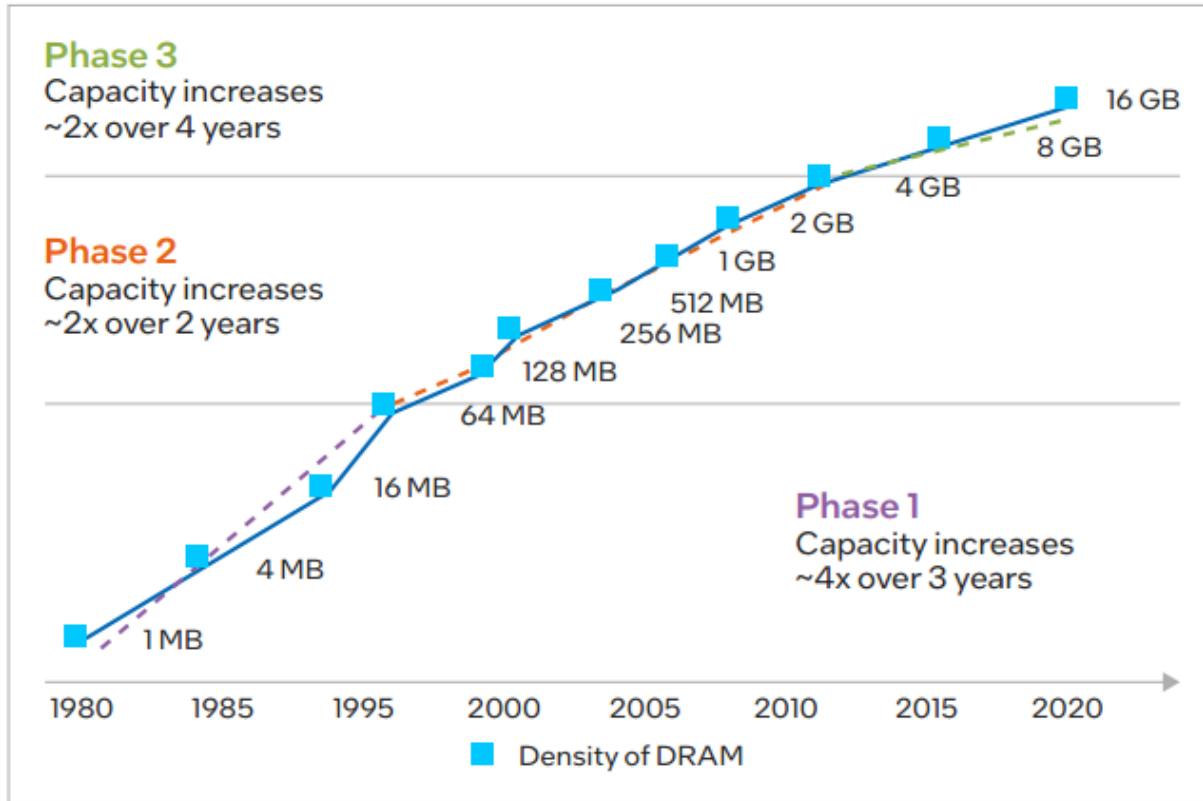
Credit to: Tran Tu and Nick Conteini

Network-based Computing Laboratory

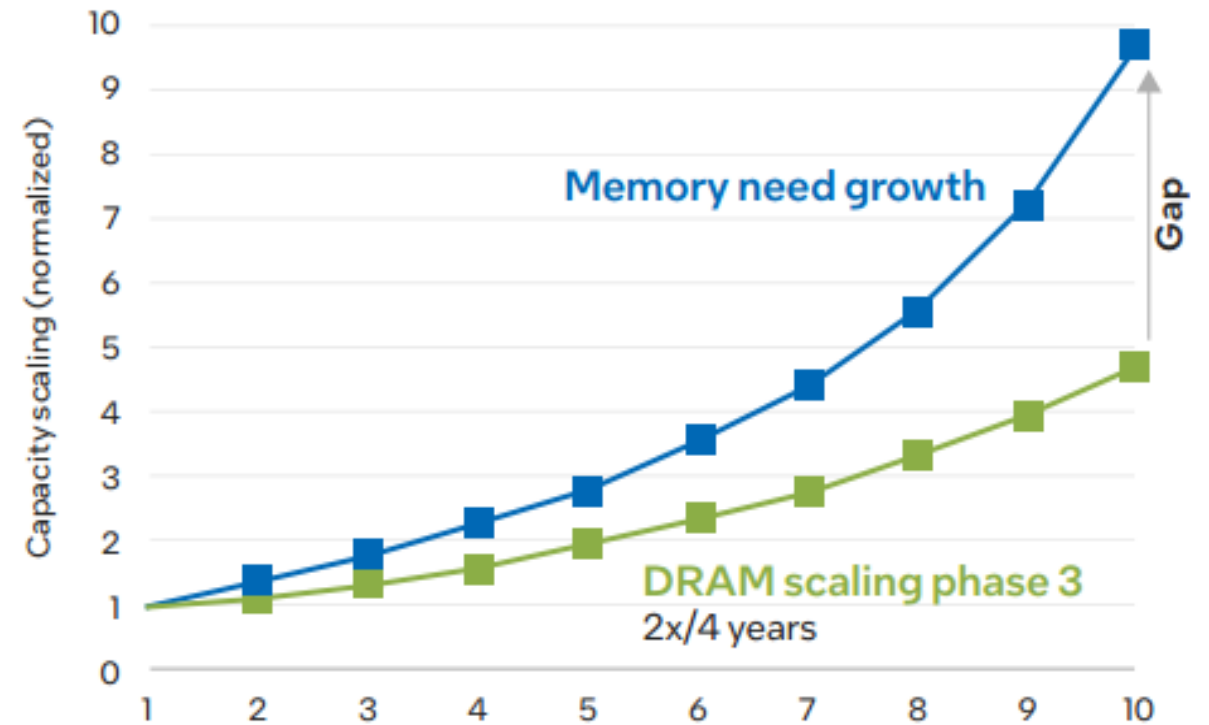
Department of Computer Science and Engineering

The Ohio State University

Memory growth trends (e.g. DRAM)



The end of the Dennard scaling made it more complex to sustain the DRAM growth



The gap between applications and growth is increasing

Overview of the MVAPICH2 Project

- High Performance open-source MPI Library
- Support for multiple interconnects
 - InfiniBand, Omni-Path, Ethernet/iWARP, RDMA over Converged Ethernet (RoCE), AWS EFA, OPX, Broadcom RoCE, Intel Ethernet, Rockport Networks, Slingshot 10/11
- Support for multiple platforms
 - x86, OpenPOWER, ARM, Xeon-Phi, GPGPUs (NVIDIA and AMD)
- Started in 2001, first open-source version demonstrated at SC '02
- Supports the latest MPI-3.1 standard
- <http://mvapich.cse.ohio-state.edu>
- Additional optimized versions for different systems/environments:
 - MVAPICH2-X (Advanced MPI + PGAS), since 2011
 - MVAPICH2-GDR with support for NVIDIA (since 2014) and AMD (since 2020) GPUs
 - MVAPICH2-MIC with support for Intel Xeon-Phi, since 2014
 - MVAPICH2-Virt with virtualization support, since 2015
 - MVAPICH2-EA with support for Energy-Awareness, since 2015
 - MVAPICH2-Azure for Azure HPC IB instances, since 2019
 - MVAPICH2-X-AWS for AWS HPC+EFA instances, since 2019
- Tools:
 - OSU MPI Micro-Benchmarks (OMB), since 2003
 - OSU InfiniBand Network Analysis and Monitoring (INAM), since 2015



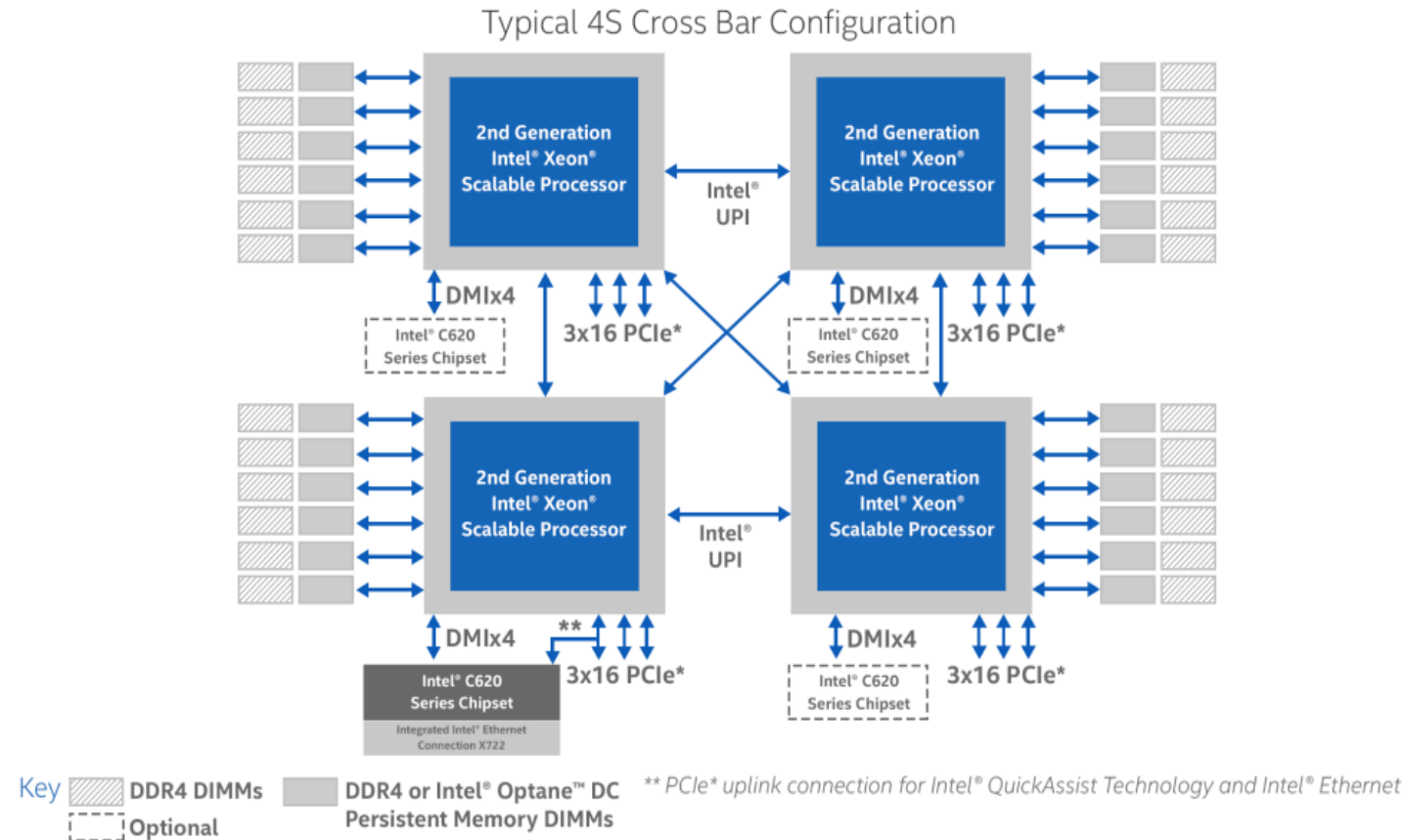
- Used by more than 3,290 organizations in 90 countries
- More than 1.63 Million downloads from the OSU site directly
- Empowering many TOP500 clusters (Nov '22 ranking)
 - 7th , 10,649,600-core (Sunway TaihuLight) at NSC, Wuxi, China
 - 19th, 448, 448 cores (Frontera) at TACC
 - 34th, 288,288 cores (Lassen) at LLNL
 - 46th, 570,020 cores (Nurion) in South Korea and many others
- Available with software stacks of many vendors and Linux Distros (RedHat, SuSE, OpenHPC, and Spack)
- Partner in the 19th ranked TACC Frontera system
- Empowering Top500 systems for more than 16 years

Why collectives?

- MPI collectives are used by many data intensive (map-reduce) workloads
- In MPI libraries, they are the heaviest in terms of computation and communication (interconnect/mem buses)
- Collective performance (such as alltoall and allreduce) is based on many factors, including but not limited to:
 - The algorithmic choice
 - The underlying pt-to-pt performance (inter-node vs intra-node)
 - The platform characteristics (e.g. CPU/Memory model)

Architecture of our experimental platform

Model	Intel Xeon Platinum 8280M ("Cascade Lake")
Total cores per CLX node:	112 cores on four sockets (28 cores/socket)
Hardware threads per core:	1 Hyperthreading is not currently enabled on Frontera
Clock rate:	2.7GHz nominal
Memory:	2.1 TB NVDIMM
Cache:	32KB L1 data cache per core; 1MB L2 per core; 38.5 MB L3 per socket. 384 GB DDR4 RAM configured as an L4 cache Each socket can cache up to 66.5 MB (sum of L2 and L3 capacity).
Local storage:	144GB /tmp partition on a 240GB SSD 4x 833 GB /mnt/fsdax[0,1,2,3] partitions on NVDIMM 3.2 TB usable local storage



The bigmem (Optane) vs smallmem node

Bigmem Node

Model	Intel Xeon Platinum 8280M ("Cascade Lake")
Total cores per CLX node:	112 cores on four sockets (28 cores/socket)
Hardware threads per core:	1, Hyperthreading is not currently enabled on Frontera
Clock rate:	2.7GHz nominal
Memory:	2.1 TB NVDIMM
Cache:	32KB L1 data cache per core; 1MB L2 per core; 38.5 MB L3 per socket. 384 GB DDR4 RAM configured as an L4 cache Each socket can cache up to 66.5 MB (sum of L2 and L3 capacity).
Local storage:	144GB /tmp partition on a 240GB SSD 4x 833 GB /mnt/fsdax[0,1,2,3] partitions on NVDIMM 3.2 TB usable local storage

Smallmem Node

Model	Intel Xeon Platinum 8280 ("Cascade Lake")
Total cores per CLX node:	56 cores on two sockets (28 cores/socket)
Hardware threads per core:	1, Hyperthreading is not currently enabled on Frontera
Clock rate:	2.7GHz nominal
Memory:	192GB (2933 MT/s) DDR4
Cache:	32KB L1 data cache per core; 1MB L2 per core; 38.5 MB L3 per socket. Each socket can cache up to 66.5 MB (sum of L2 and L3 capacity).
Local storage:	144GB /tmp partition on a 240GB SSD

STREAM bandwidth with PMEM memory modes

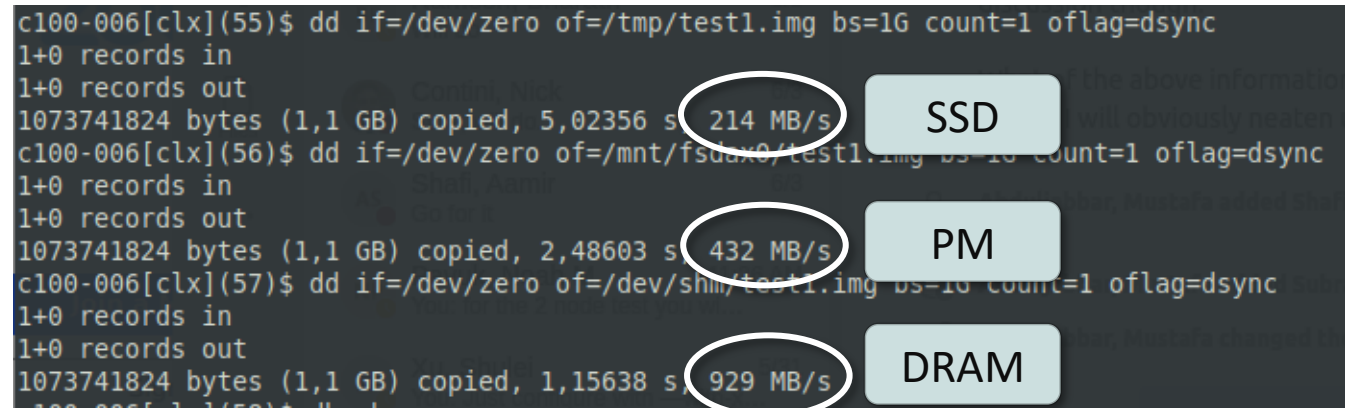
- NOWLAB system: MRI
 - Core(s) per socket: 28
 - Socket(s): 2
 - L2 cache: 1280K (70MB in total)
 - L3 cache: 43008K (84MB in total)
 - DRAM: 256GB
 - Optane: 991GB

BW (MB/s)	Memory mode	App direct	Memory mode	App direct
Total memory required (GB)	4.6	4.6	457.8	457.8
Copy	156246.3	2311.2	20249.0	2453.7
Scale	174390.6	2411.8	16097.6	2202.2
Add	194115.5	3831.0	17414.0	2910.5
Triad	197335.6	3854.9	22558.8	2776.7

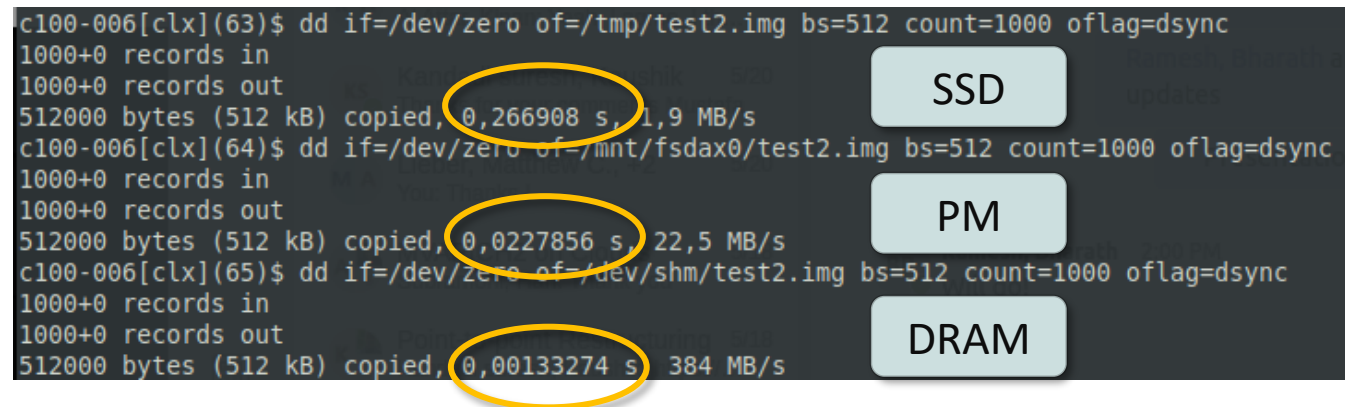
Quick I/O sanity check shows 3 different BW/latency on the Intel-Optane node

- Single core, hence, not maximizing streaming BW usage
- SSD -> PM -> DRAM
 - 2x BW increase on each step
 - 10x latency decrease on each step

```
c100-006[clx](55)$ dd if=/dev/zero of=/tmp/test1.img bs=1G count=1 oflag=dsync
1+0 records in
1+0 records out
1073741824 bytes (1,1 GB) copied, 5,02356 s, 214 MB/s
c100-006[clx](56)$ dd if=/dev/zero of=/mnt/fsdax0/test1.img bs=1G count=1 oflag=dsync
1+0 records in
1+0 records out
1073741824 bytes (1,1 GB) copied, 2,48603 s, 432 MB/s
c100-006[clx](57)$ dd if=/dev/zero of=/dev/shm/test1.img bs=1G count=1 oflag=dsync
1+0 records in
1+0 records out
1073741824 bytes (1,1 GB) copied, 1,15638 s, 929 MB/s
```

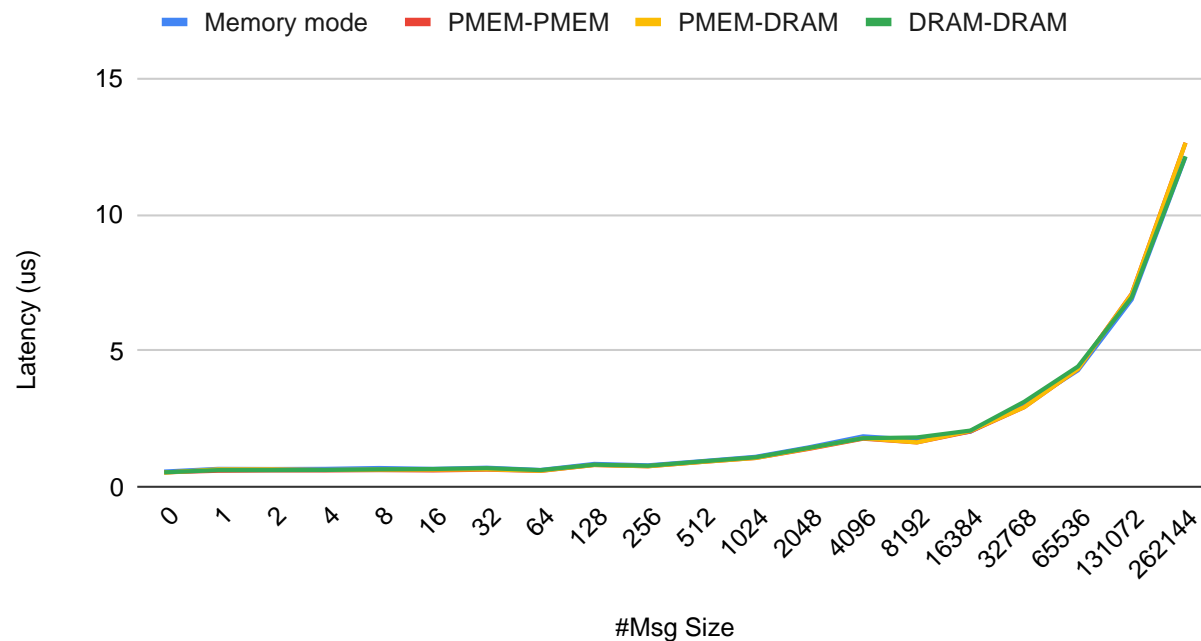


```
c100-006[clx](63)$ dd if=/dev/zero of=/tmp/test2.img bs=512 count=1000 oflag=dsync
1000+0 records in
1000+0 records out
512000 bytes (512 kB) copied, 0,266908 s, 1,9 MB/s
c100-006[clx](64)$ dd if=/dev/zero of=/mnt/fsdax0/test2.img bs=512 count=1000 oflag=dsync
1000+0 records in
1000+0 records out
512000 bytes (512 kB) copied, 0,0227856 s, 22,5 MB/s
c100-006[clx](65)$ dd if=/dev/zero of=/dev/shm/test2.img bs=512 count=1000 oflag=dsync
1000+0 records in
1000+0 records out
512000 bytes (512 kB) copied, 0,00133274 s, 384 MB/s
```

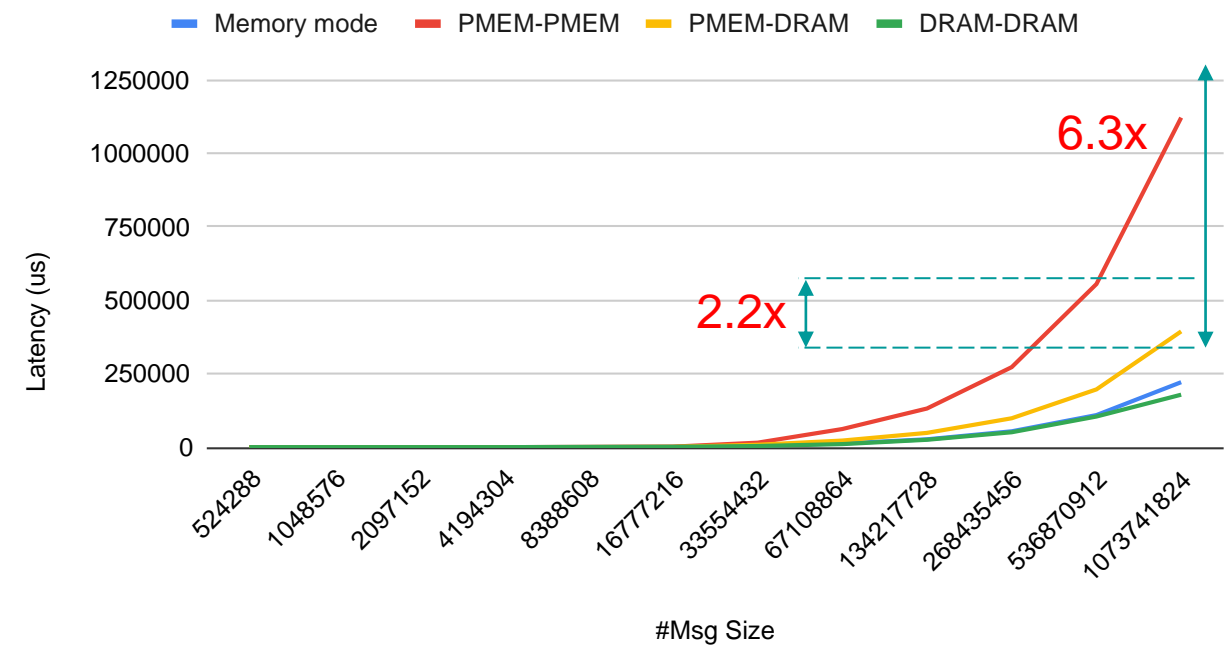


MV2 Pt-to-Pt based with PMEM memory modes

P2P performance with different buffer allocations

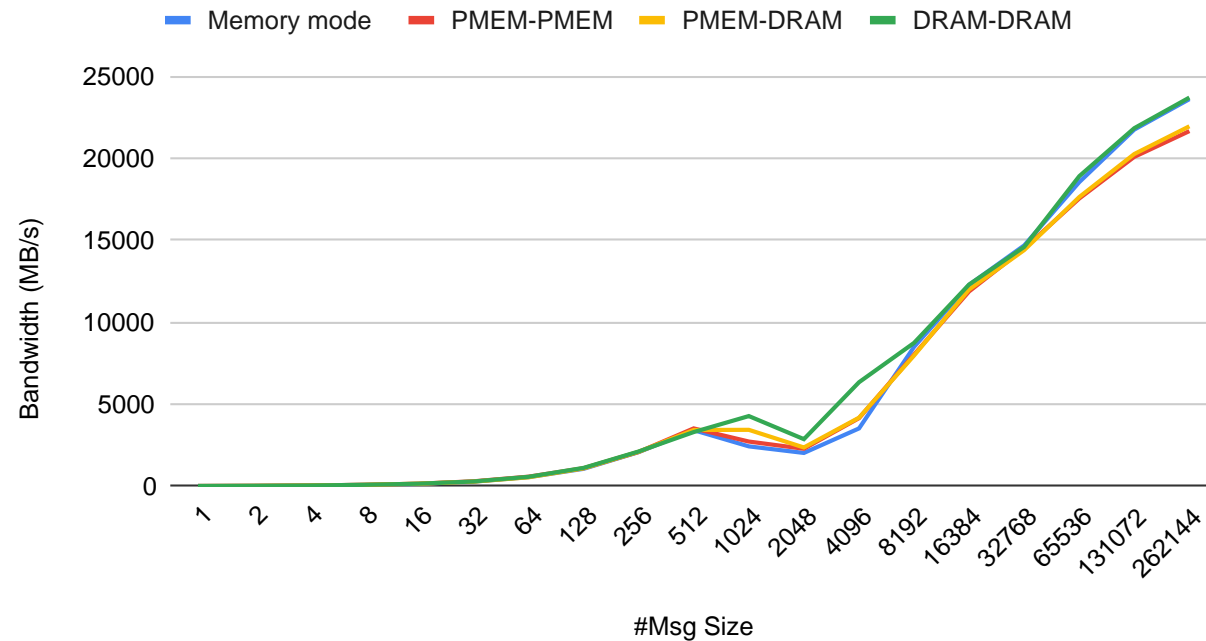


P2P performance with different buffer allocations

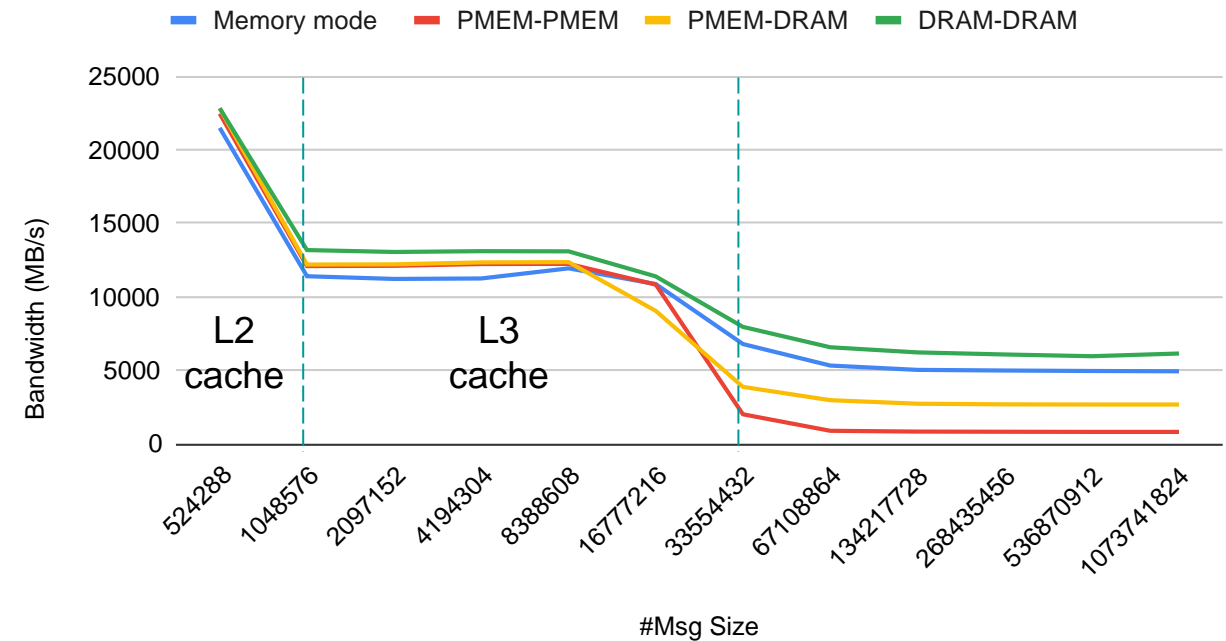


MV2 Pt-to-Pt based with PMEM memory modes

P2P performance with different buffer allocations

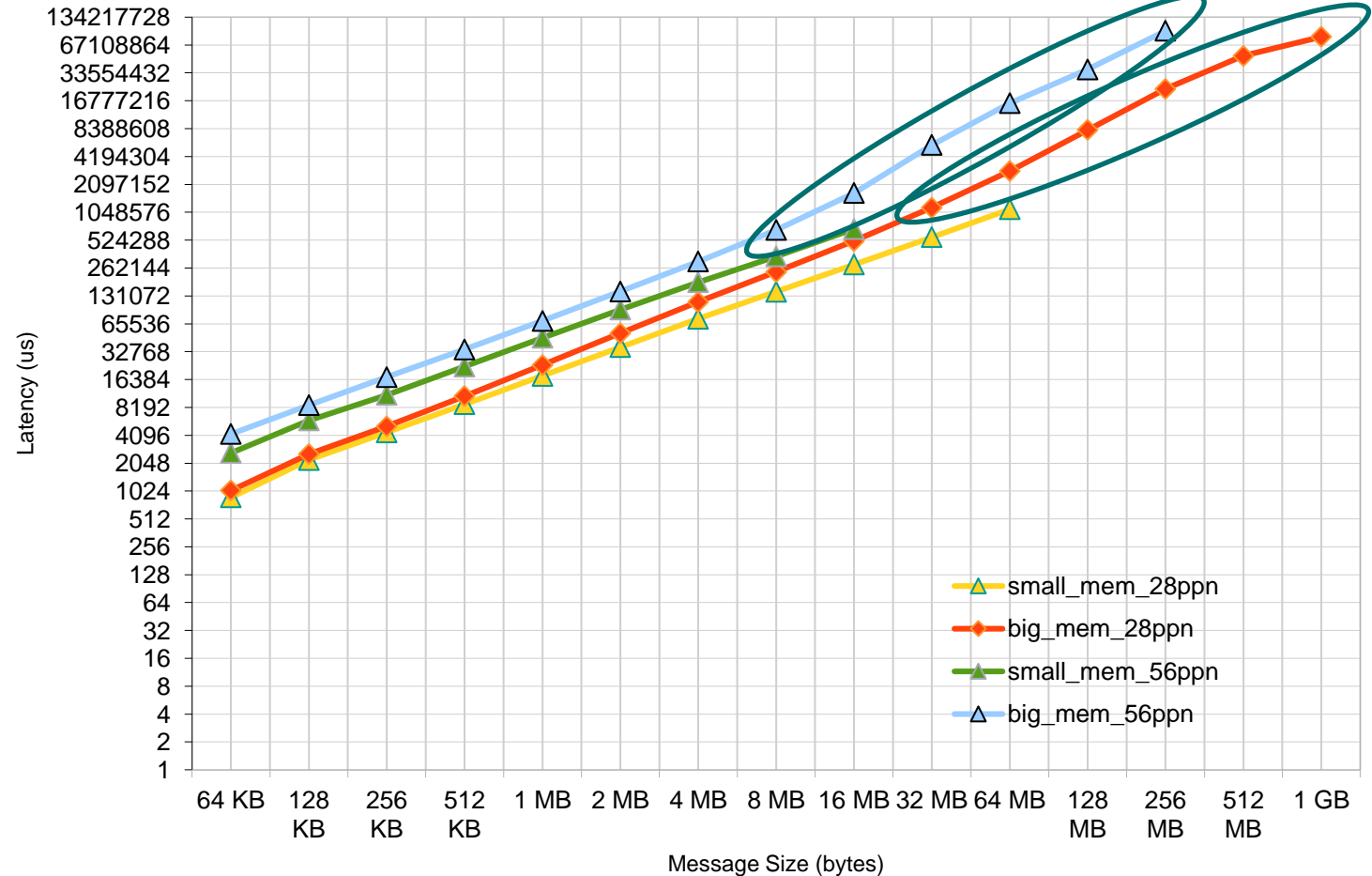


P2P performance with different buffer allocations



Alltoall collective behavior

Average Latency of Single Node AlltoAll on Frontera
"Small Mem" vs "Big Mem"



Around 1 TB of alltoall exchange
without drop in scaling

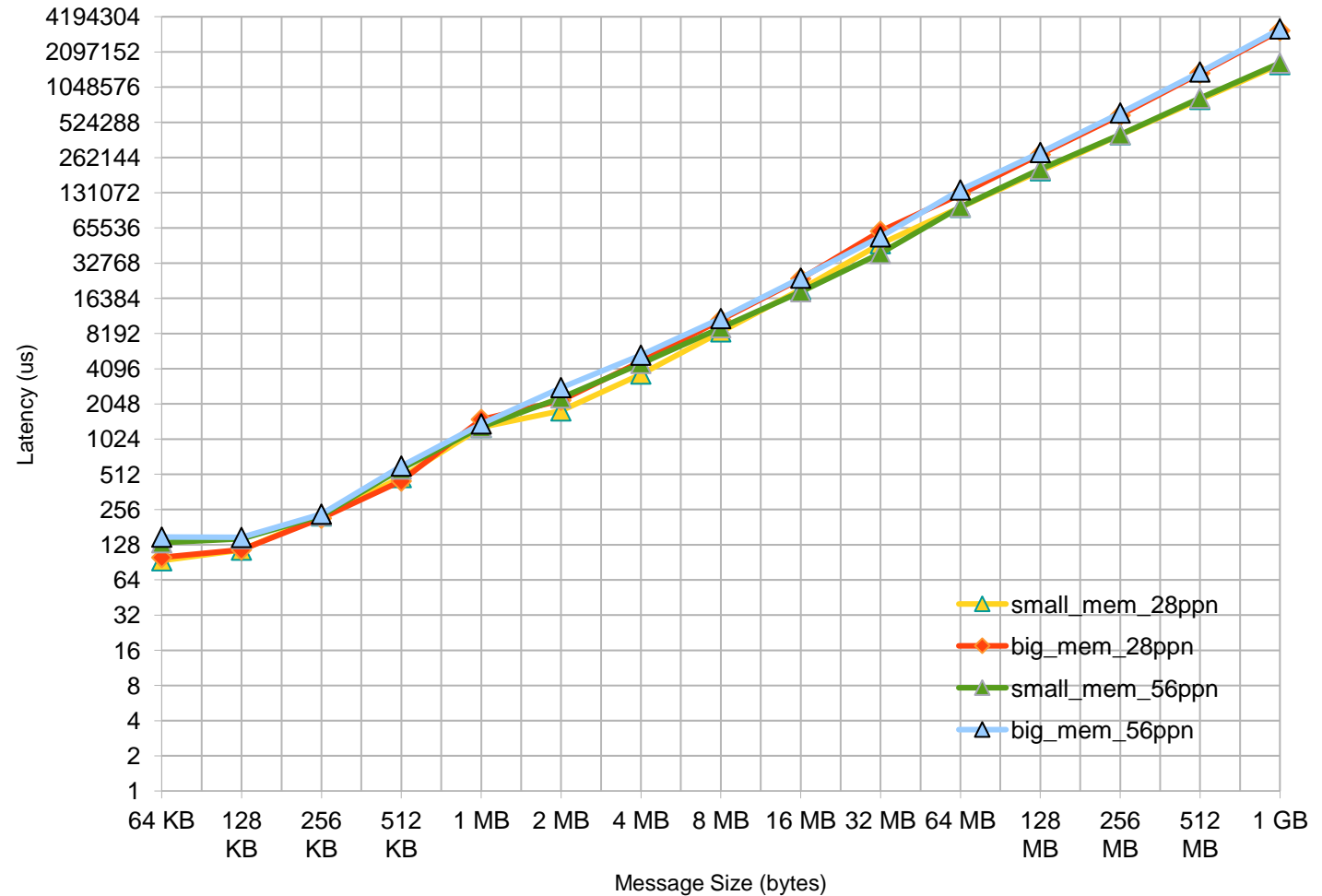
Proper collective tuning choice and
techniques to push the boundaries of
effective resource usage

Observed increase in latency due to
higher latency

Allreduce collective behavior

- Similar performance up to L2 (1MB), and smallmem outperforms bigmem beyond L3 size (32M)
- However, lower degradations because allreduce is more computationally heavy
- On par with DRAM with the small range (< 1MB message size)

Average Latency of Single Node Allreduce on Frontera
"Small Mem" vs "Big Mem"



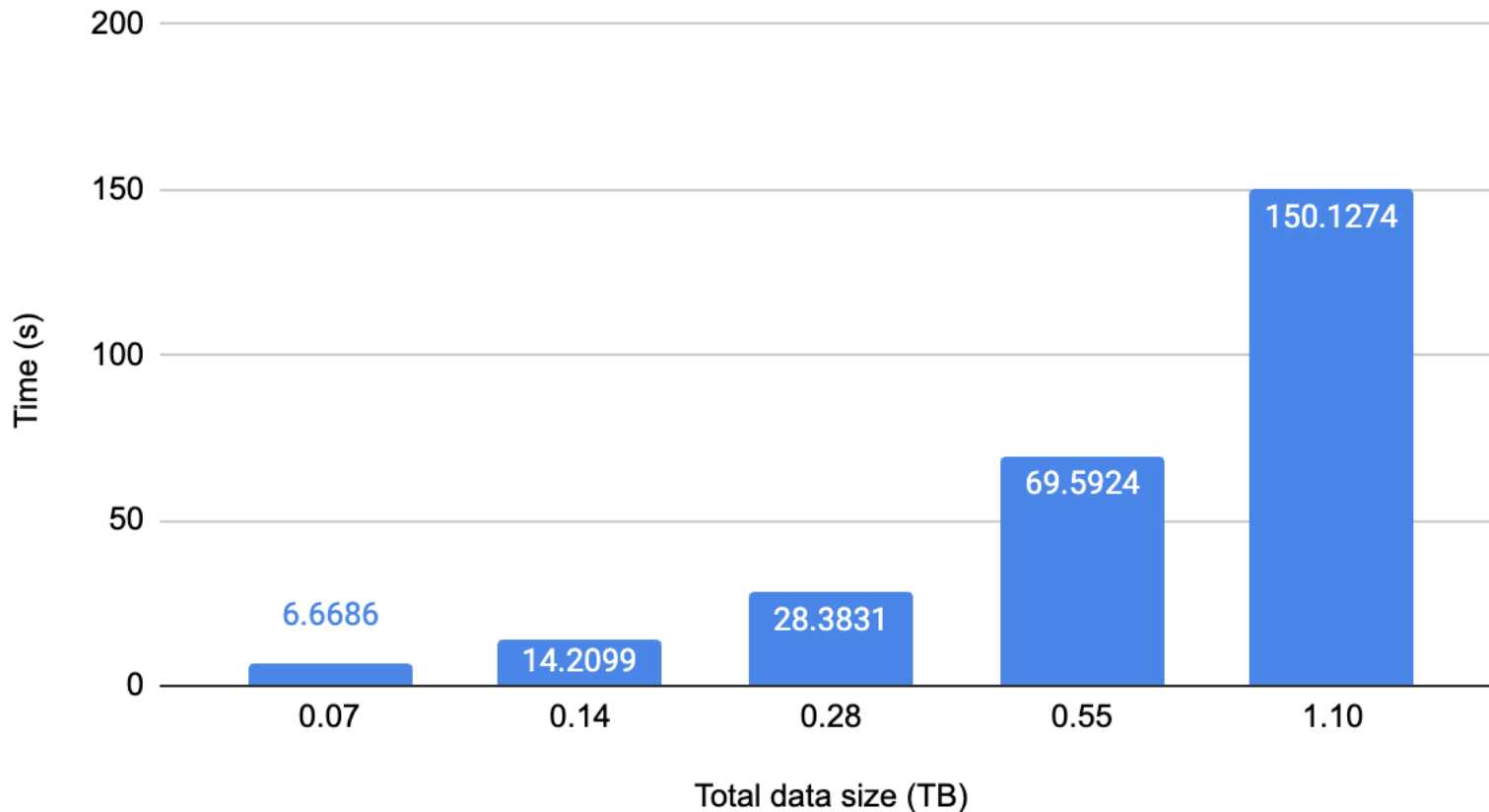
Using PMEMs for out-of-RAM sorting

- We use usort [GitHub - hsundar/usort: Fast distributed sorting routines using MPI and OpenMP](#)
- Compare and evaluate the performance of **one PMEM node** with **multiple DRAM ones** when data cannot fit into a single DRAM at application level

Configuration	DRAM node	PMEM node
Core(s) per socket	28	28
Socket(s)	2	4
L2 cache	1024K	1024K
L3 cache	39424K	39424K
DRAM	186G	186G
Optane	N/A	1.9T

Behavior of distributed sorting (i.e. complexity)

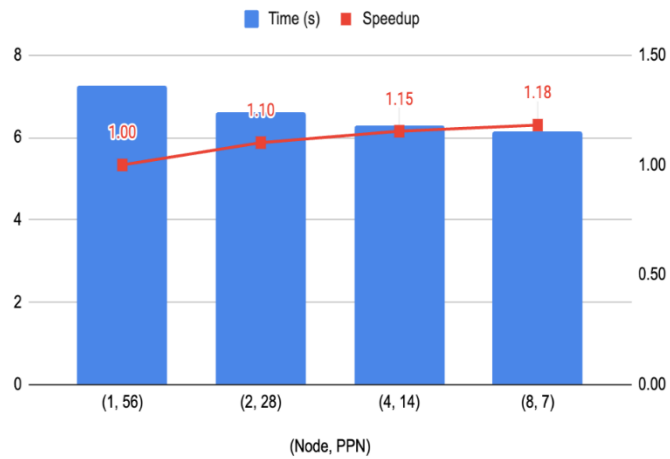
PMEM performance with different data size (1 node 56 processes)



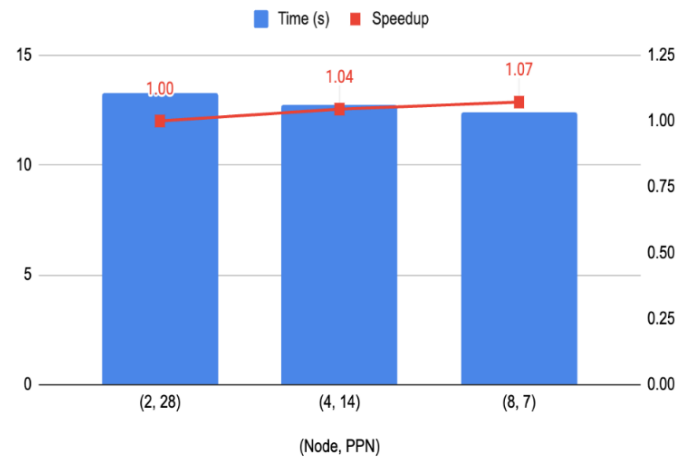
- We used the usort (disk-to-disk sorting) as a use-case for large collectives (alltoall and allreduce)
- The scale-up mode is scalable according to our tests sorting 1.1 TB of keys on a single PMEM node
- Generally speaking, this approach is cost effective (i.e. in terms of performance, operation and energy)

Scale-out vs Scale-up (usort)

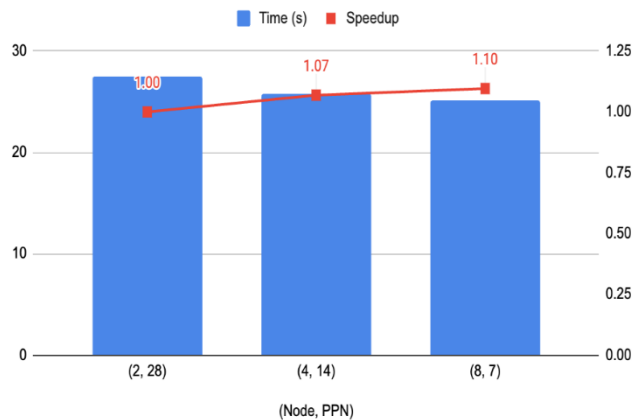
Multiple DRAM performance on sorting 0.07 TB of data



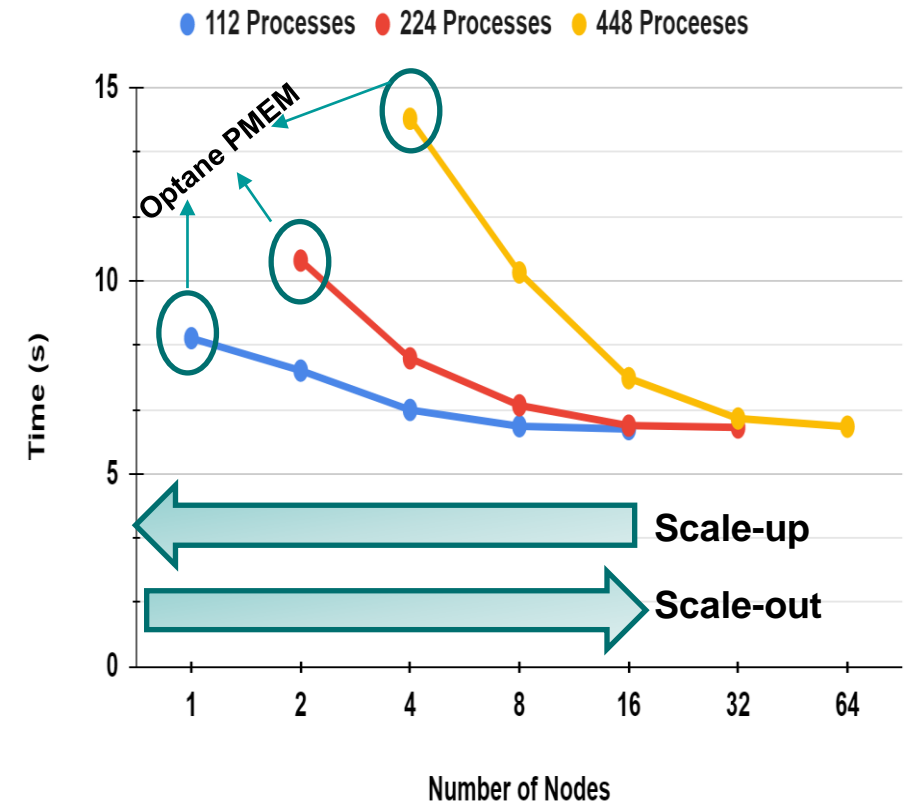
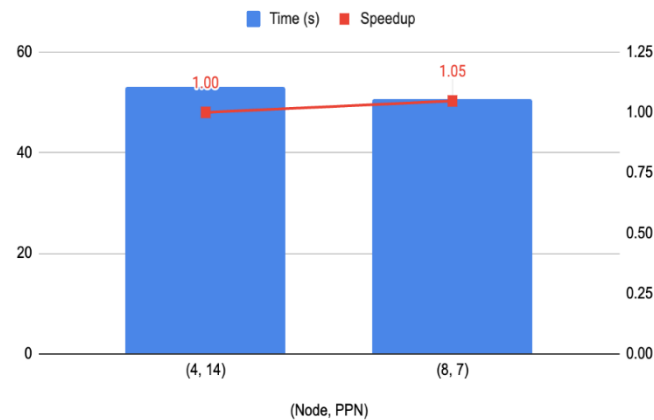
Multiple DRAM performance on sorting 0.14 TB of data



Multiple DRAM performance on sorting 0.28 TB of data



Multiple DRAM performance on sorting 0.55 TB of data



Observations on the Multi-tiered Approach

- Intel PMEMs do not function as HBMs, however, they provide for a cost-effective approach for scale-up and scale-out performance compared to multiple expensive DRAM, HBM nodes
- For volatile usage, use in Memory mode (DRAM as an L4 cache)
- Intel is discontinuing this series to up their game in advanced interconnects (CXL, CCIX)
- Using more DRAMs has only a small benefit of 10 – 18% speedup in the 4-node case
 - More cache capacity
 - More memory channels
 - Higher cost

Can we close the performance gap?

- To recap, performance is impacted by
 - Lower aggregate memory bandwidth
 - In the scale-up approach, lower number of core resource (i.e. caches)
- We are collaborating with ETRI to close this gap
 - Scale-out and process near-memory

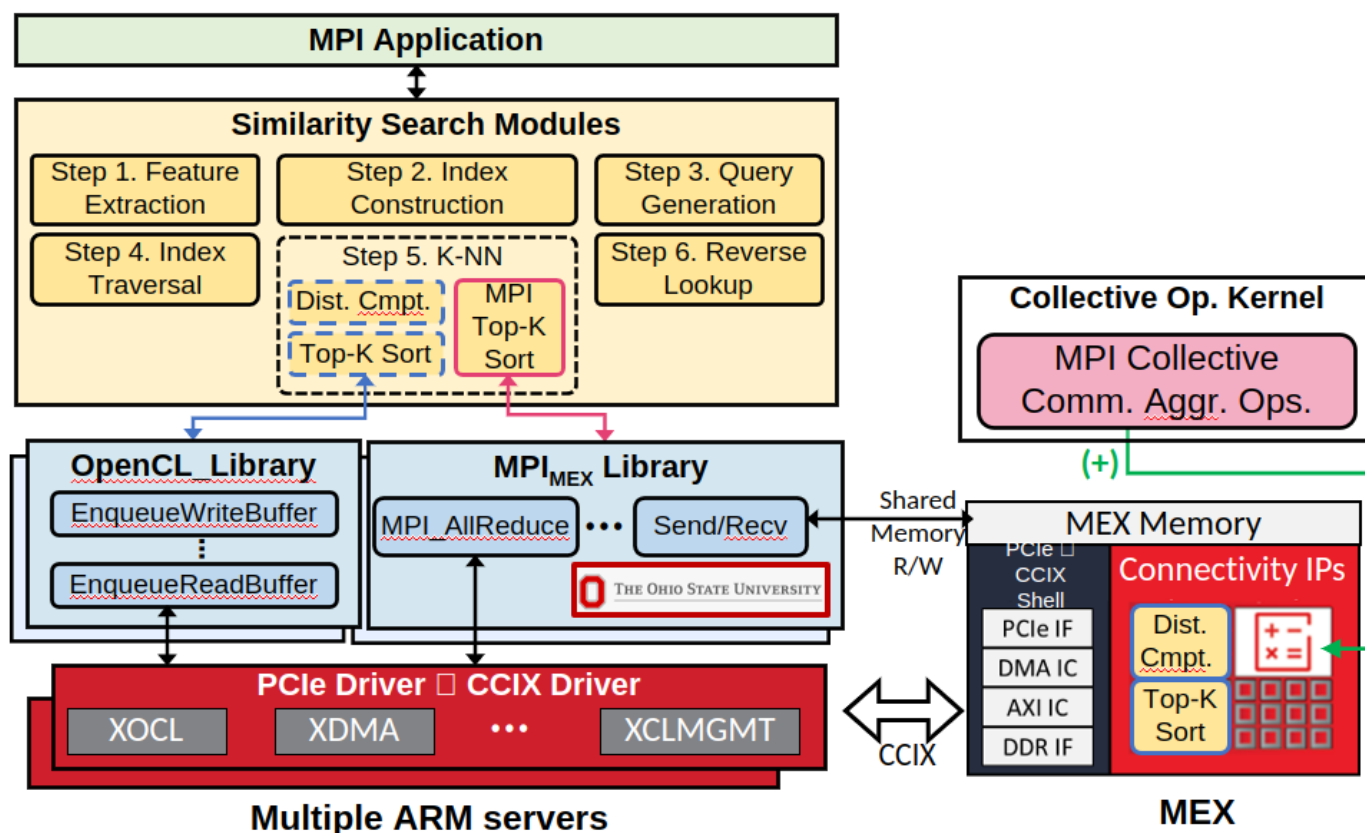
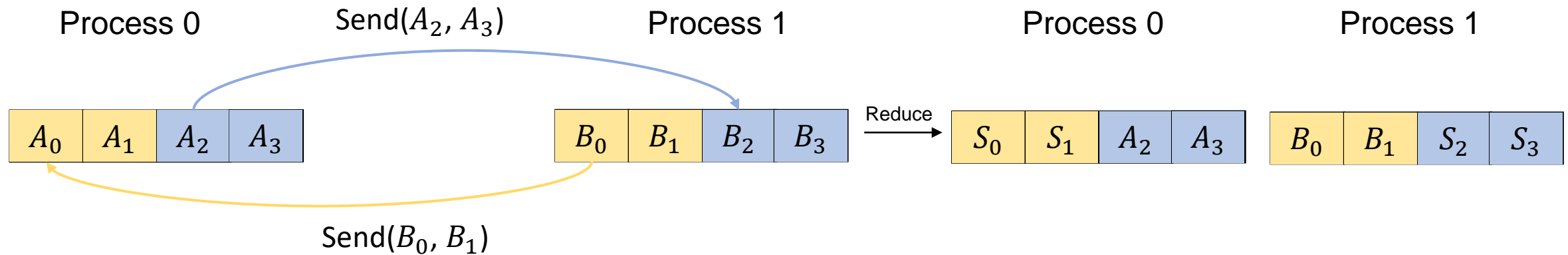


Figure: showing the interaction between MVAPICH and the MEX hardware (in development) - courtesy to ETRI

Designs for MPI_Allreduce : Overview

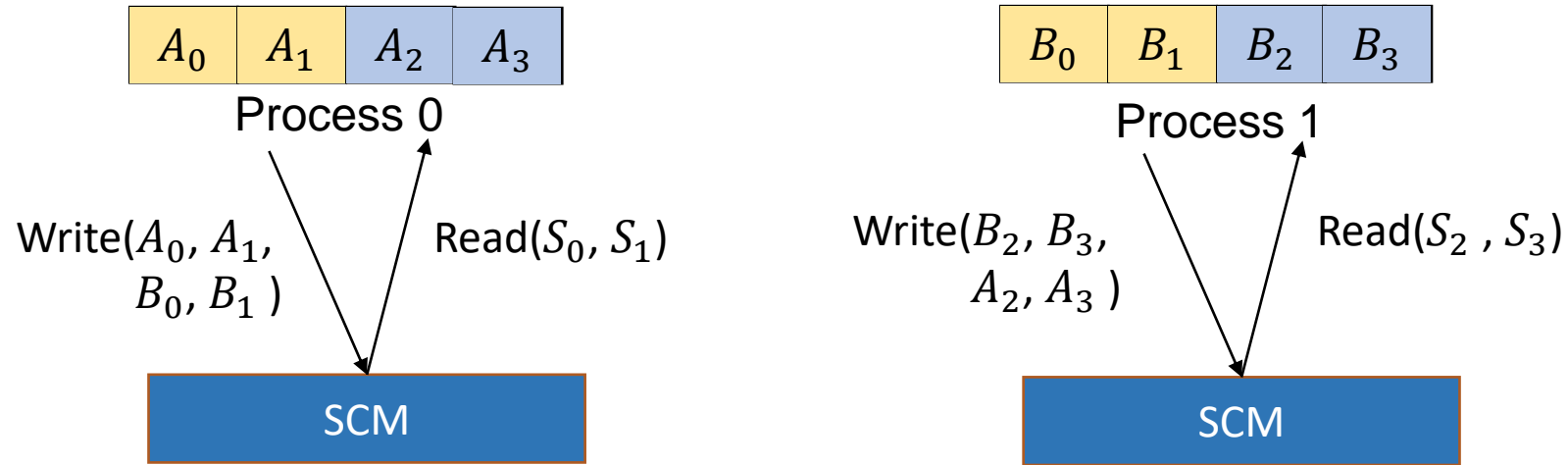
- Take a reduce-scatter allgather algorithm as a case-study
 - Very widely used for long vector reductions ($\geq 256\text{KB}$)
 - Implement a staging-based emulated design and study on two processes as a starting point
 - Use MPI_SUM as the sample MPI operation (applicable to others as well)
- Two phase algorithm
 - Reduce-scatter phase results in every process having one “chunk” of the final reduced buffer
 - Every step in the reduce-scatter involves a communication operation, followed by compute (example : sum)
 - Allgather phase involves communication only and ensures that every process has all reduced chunks in the receive buffer
- All send/recv operations are on DRAM buffers
- Staging is done between DRAM and Optane

Designs for MPI_Allreduce : Reduce scatter



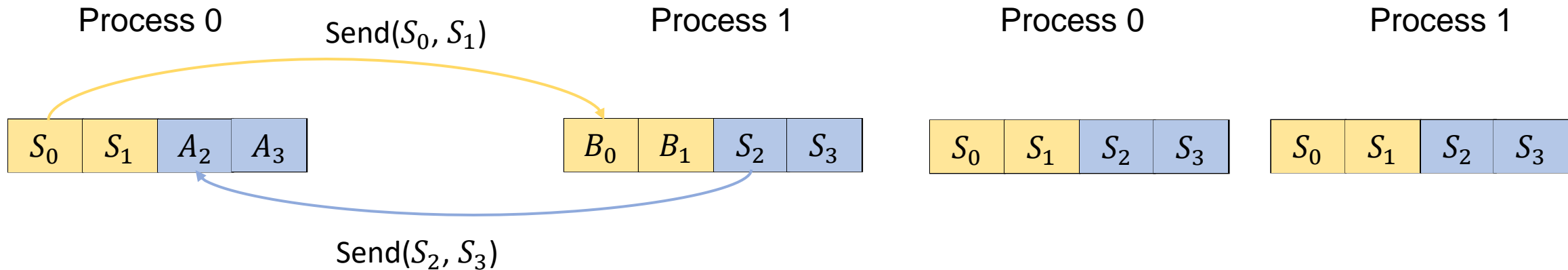
- First phase involves exchanging one half of the buffer with a partner process
 - Process on the left/right are responsible for reduction of first/second half respectively
 - Process on the left computes $S_0 = A_0 + B_0, S_1 = A_1 + B_1$
 - Process on the right computes $S_2 = A_2 + B_2, S_3 = A_3 + B_3$
 - In the case of multiple processes, every process has a partner, and these steps are divided into sub-steps recursively

Designs for MPI_Allreduce : Emulate FPGA



- Start compute after staging operation is complete
 - Measure “round-trip” time i.e., the staging overhead (to/from the SCM memory/optane)
 - Measure compute time on the CPU
 - To emulate FPGAs, we simply divide the measured CPU compute time in Allreduce by the expected speedup from the FPGAs
 - In a real scenario, compute happens on the FPGA and the reduced buffer is read from the SCM memory on completion of compute --- the experiment above gives a close estimate

Designs for MPI_Allreduce : Allgather



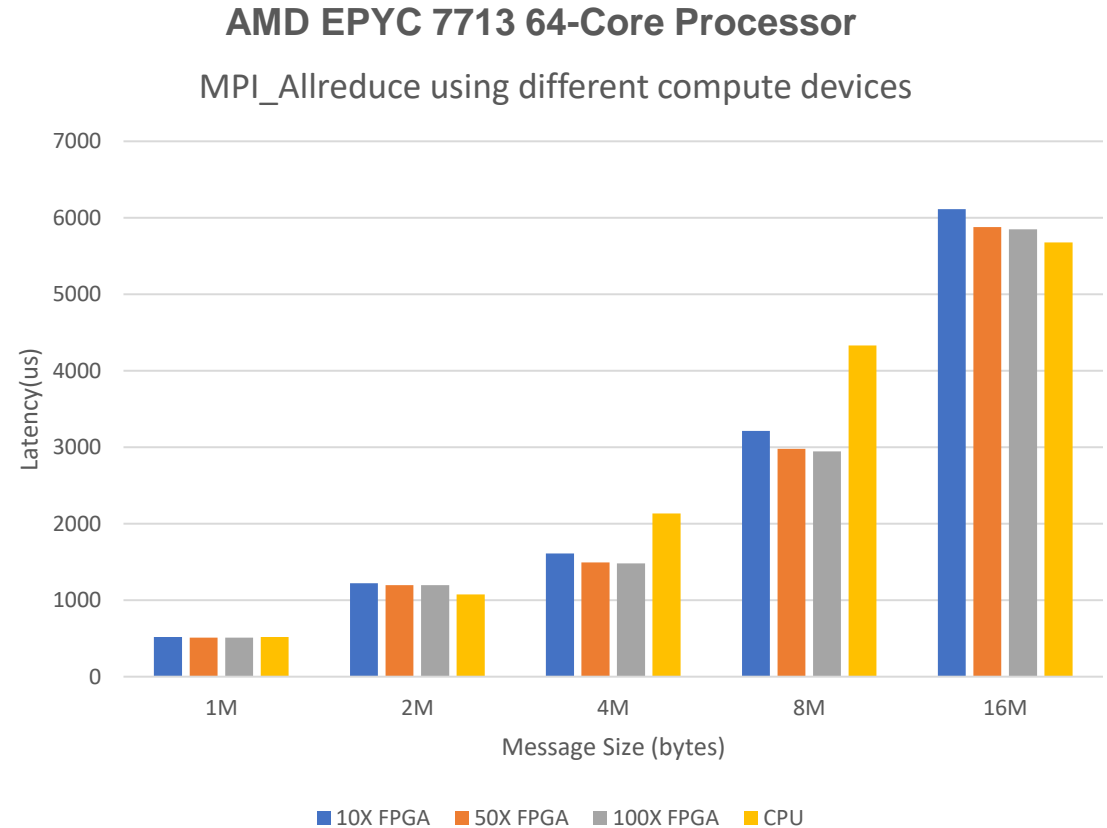
- Final phase involves an “allgather” to exchange sums
 - This adds to the total communication time for allreduce
 - Pure comm time = $T(\text{allgather}) + T(\text{reduce-scatter-comm})$
 - Compute time = $T(\text{compute})/\text{expected_compute_improvement} + \text{staging_overhead}$
 - For CPU only runs, $\text{staging_overhead} = 0$, $\text{expected_compute_improvement} = 1$

Experimental results

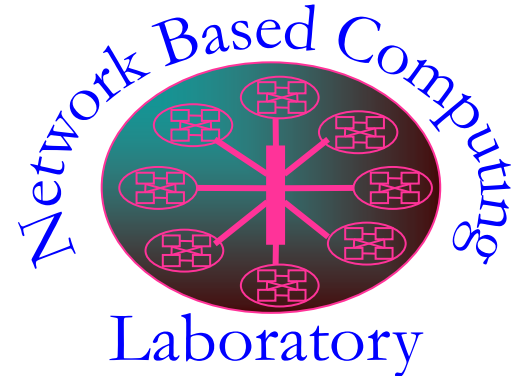
- We run 1 process per node, 2 nodes on two different platforms with the Infiniband HDR-200 interconnect
 - AMD EPYC 7713 64-Core Processor
- Since AMD does not support Optane, we only measure compute/communication time on the platform and use staging overheads obtained on the intel platform
- We compare CPU-only algorithms with emulated FPGAs
 - Each 10X, 50X and 100X faster than the CPU respectively
- Use representative long vector message range : 1M – 16M

Experimental results : Overall latency

- Staging to FPGA performs up to ~30% better than CPU based algorithm on AMD platforms for 4M and 8M messages
- We are working closely on lowering host-to-SCM/FPGA latency and coming up with better near-memory reduction designs
- A prototype HW implementation is underway



THANK YOU!



Network-Based Computing Laboratory
<http://nowlab.cse.ohio-state.edu/>



The High-Performance MPI/PGAS
Project
<http://mvapich.cse.ohio-state.edu/>



The High-Performance Big Data
Project
<http://hibd.cse.ohio-state.edu/>

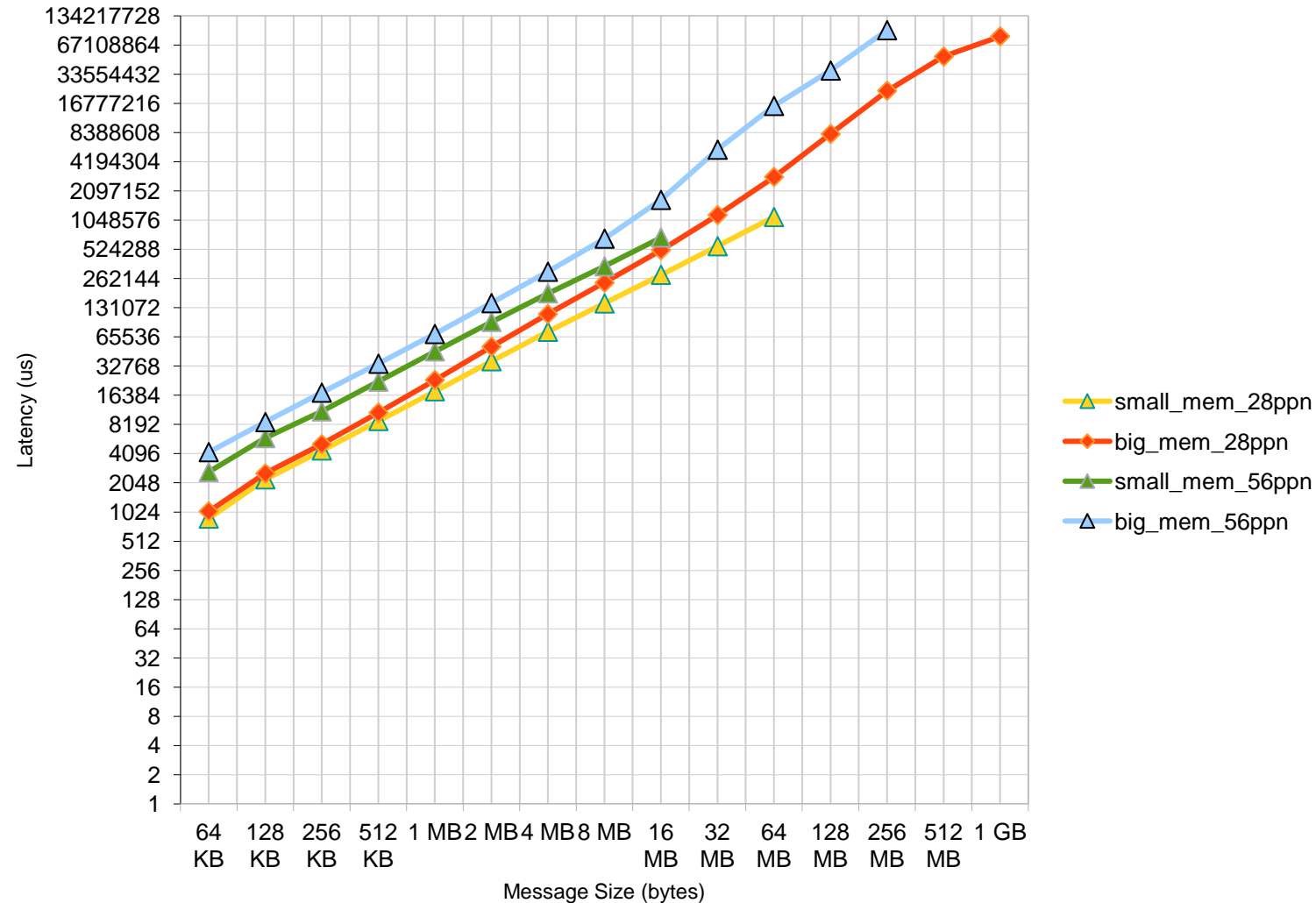


The High-Performance Deep Learning
Project
<http://hidl.cse.ohio-state.edu/>

Experimental results - Alltoall

- Run Alltoall up to 1GB on both nodes to form a baseline
- Bigmem runs for larger messages due to higher capacity
- Total memory consumed exceeds DRAM capacity
 - Smallmem_28ppn
 - Per process memory req. = $128 * 2 = 7168\text{MB}$
 - Total memory req. = $7168 * 28 = 200\text{GB}$
 - 200GB exceeds DRAM size

Average Latency of Single Node AlltoAll on Frontera
"Small Mem" vs "Big Mem"



Experimental results - Allgather

- Run Allgather up to 1GB on both nodes to form a baseline
- Bigmem runs for larger messages due to higher capacity
- Total memory consumed exceeds DRAM capacity
 - Smallmem_28ppn
 - Per process memory req. = $256 * 28 \sim 7168\text{MB}$
 - Total memory req. = $7168 * 28 \sim 200\text{GB}$
 - 200GB exceeds DRAM size

Average Latency of Allgather on Frontera – “Small Mem” Vs. “Big Mem”

