Re-designing Communication and Work Distribution in Scientific Applications for Extreme-scale Heterogeneous Systems

Project Team:
Karen Tomko (PI), Ohio Supercomputer Center
Dhabeleswar K. Panda (Co-PI), Ohio State University
Khaled Hamidouche, Ohio State University
Hari Subramoni, Ohio State University
Jithin Jose, Ohio State University
Raghunathan Raja Chandrasekar, Ohio State University
Rong Shi, Ohio State University
Akshay Venkatesh, Ohio State University
Jie Zhang, Ohio State University

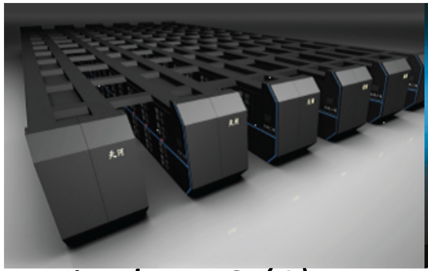
Drivers of Modern HPC System Architectures

Multi-core Processors

High Performance Interconnects

Accelerators / Coprocessors
high compute density, high performance/watt
>1 TFlop DP on a chip

- Multi-core processors are ubiquitous
- Modern interconnects have high performance features such as RDMA and support for collectives
- Accelerators/Coprocessors becoming common in high-end systems
- Pushing the envelope for Exascale computing



Tianhe – 2 (1)

Titan (2)

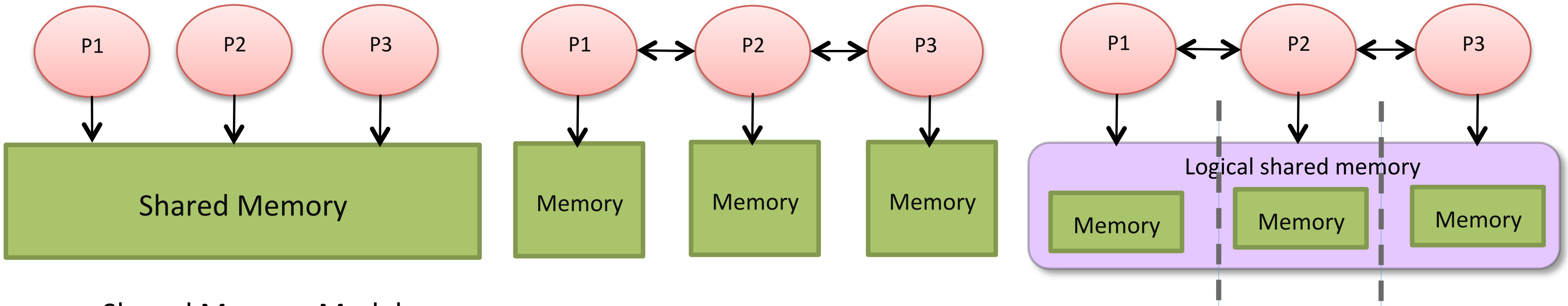
Stampede (6)

Blue Waters

Challenges for Communication Runtimes

- Complex Architecture
  - Within a node
    - Accelerators connected via PCIe,
    - NUMA shared memory
  - Interconnect feature and topology consideration
- Scaling
  - Current algorithms developed and tested with 100s to 1000s of processes
  - few systems on which to run with 10,000s to 100,000s

Parallel Programming Models Overview



Shared Memory Model

SHMEM, OpenMP

Distributed Memory Model

MPI (Message Passing Interface)

Partitioned Global Address Space (PGAS)

Global Arrays, UPC, OpenSHMEM, …

- Programming models provide abstract machine models
- Models can be mapped on different types of systems
  - e.g. Distributed Shared Memory (DSM), MPI within a node, etc.
- Many Core models
  - OpenMP, OpenACC, CUDA

Key Questions

- How do MPI collectives perform at extreme scales?
- How well do the CraySHMEM and UPC PGAS collective communications scale?
- Can both the CPU and GPU resources be leveraged effectively in a hybrid node system?

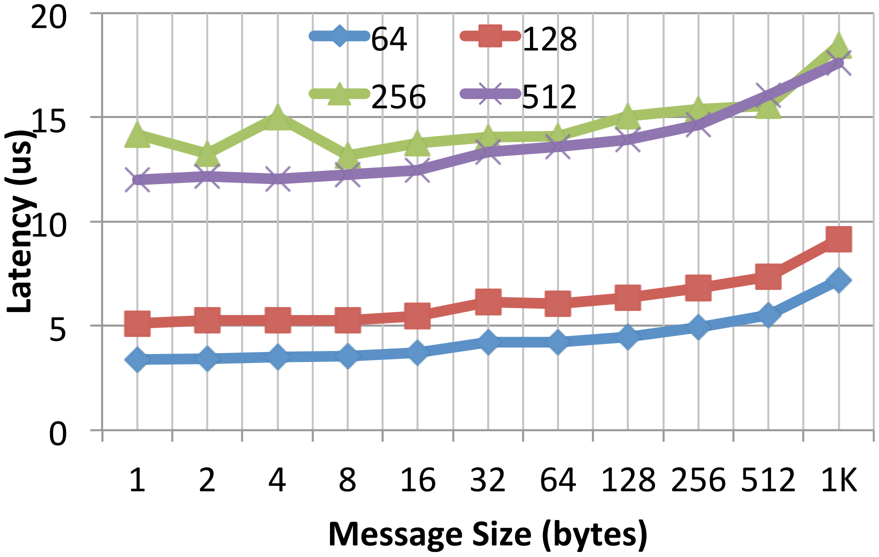
MPI on Blue Waters

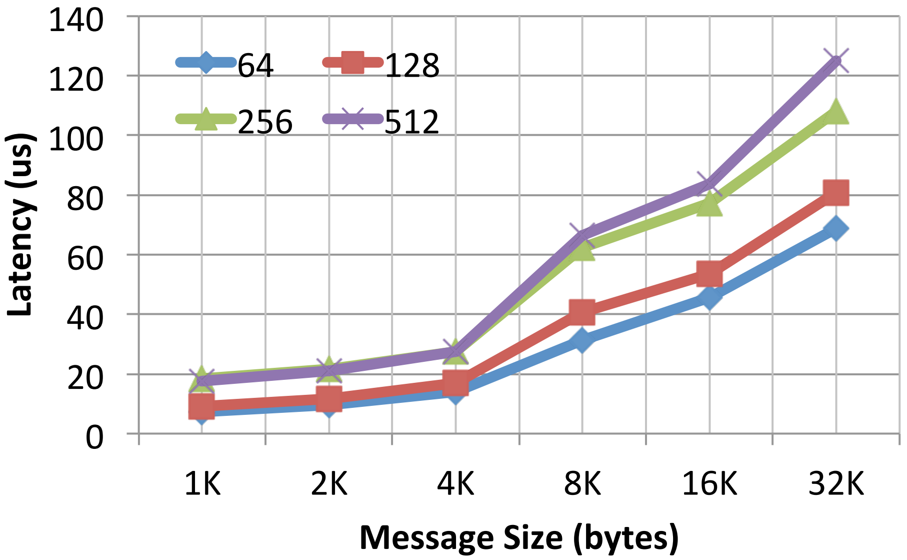
- Domain applications such weather forecasting, earthquake simulations and many more have a real requirement for large throughput capability
- MPI is the most dominant programming model for distributed memory systems
- MPI jobs in order of 1K processes becoming common
- MPI jobs in order of 1M processes is the maximum
- Blue Waters is one of the first instances that can be used to test performance of MPI jobs at a really large scale

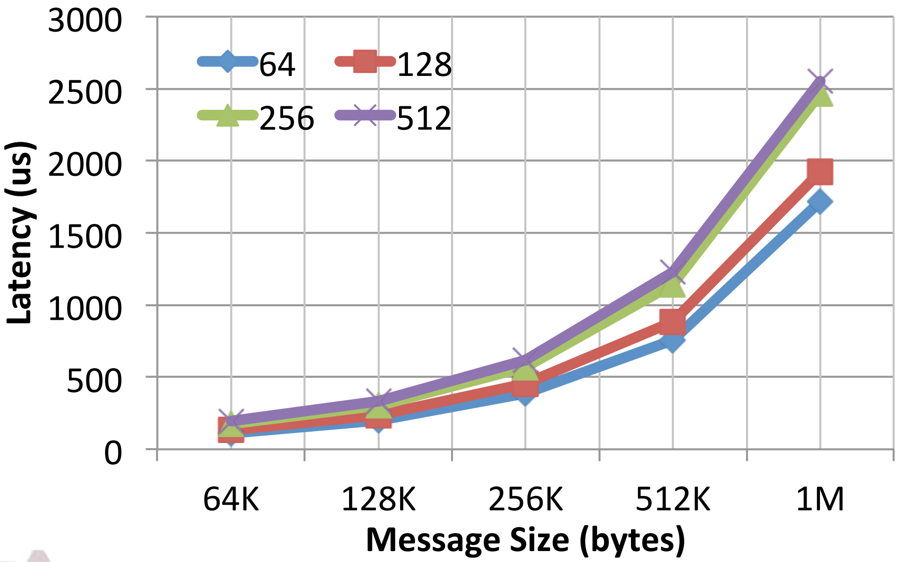
Blue Waters MPI Collective Performance

- Point-to-point operations and Collective operations determine the performance of MPI programs
- Performance of point-to-point operations involve
  - Efficient utilization of underlying interconnection hardware
  - Design of high performance protocols
- Performance of collectives additionally involves
  - Design of efficient algorithms
- We evaluate performance of common collectives such as:
  - MPI_Bcast
  - MPI_Reduce
  - MPI_Allgather

Performance of MPI_Bcast (64 – 512 Processes)

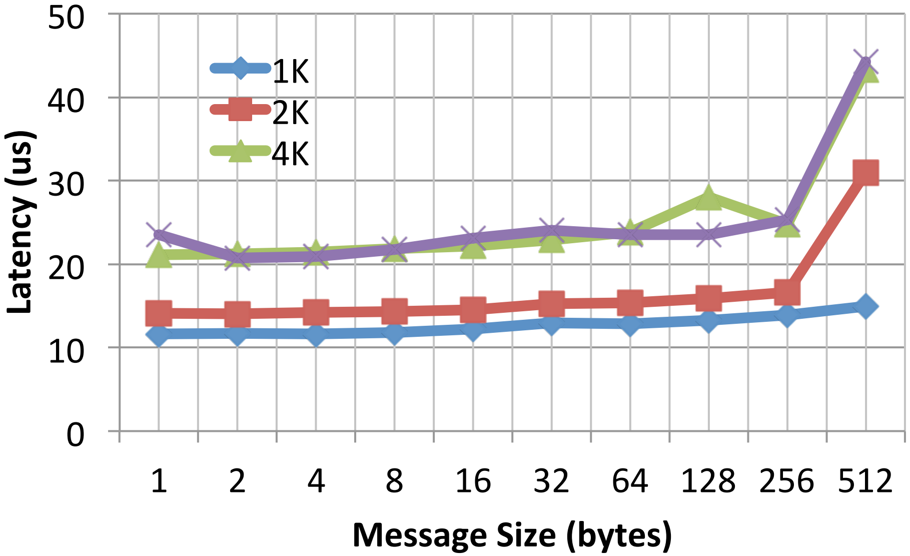


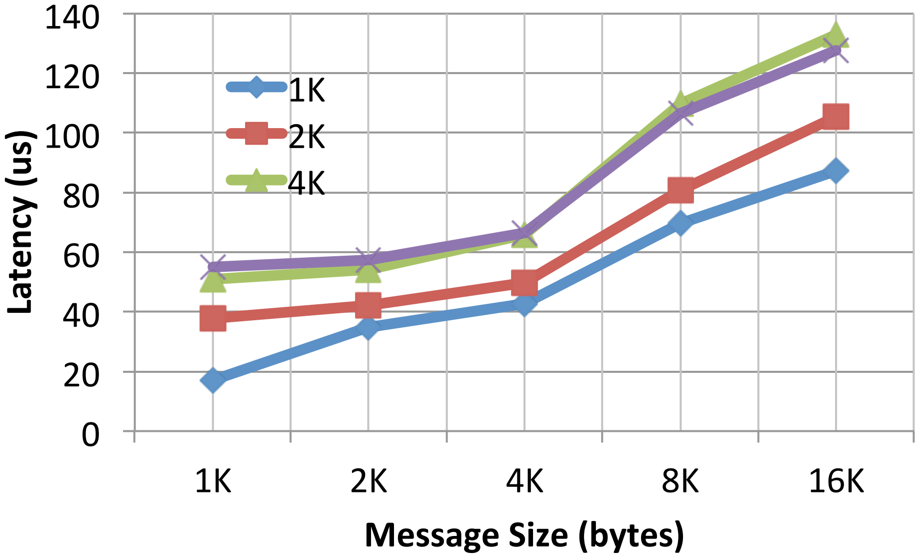


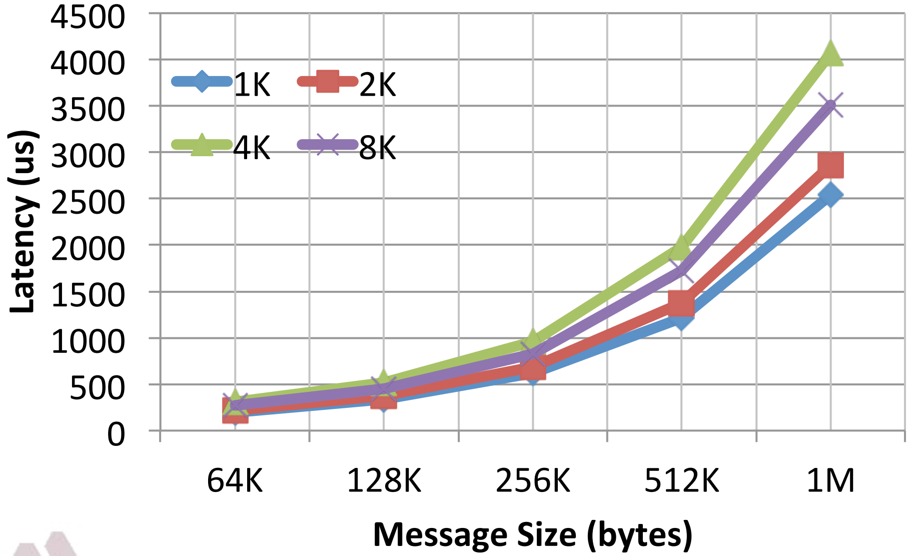


- Latency is flat in the 1 byte – 32 byte range and then starts climbing – regardless of process count
- Latency of broadcast more than doubles in the short message range going from 128 processes to 256 processes which is undesirable

Performance of MPI_Bcast (1K – 8K Processes)

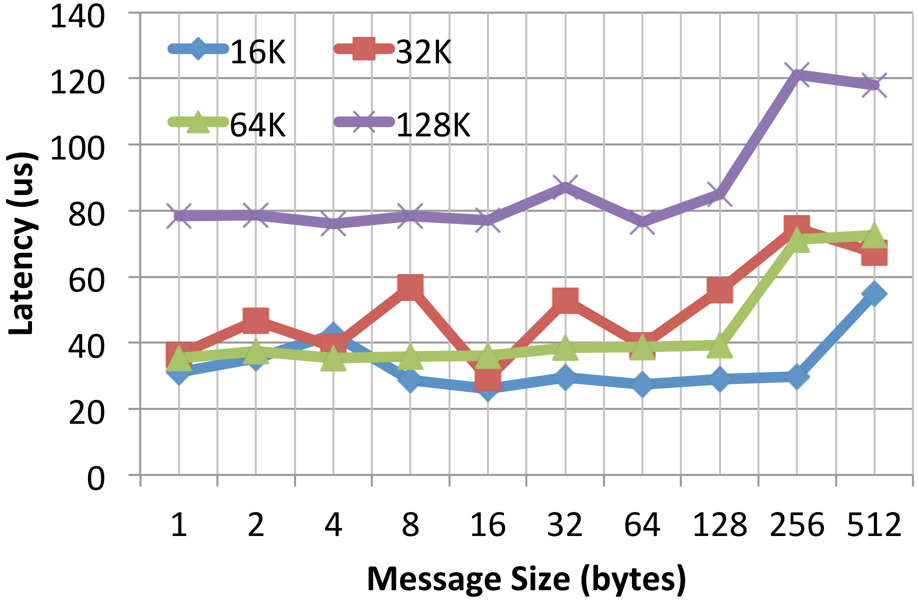


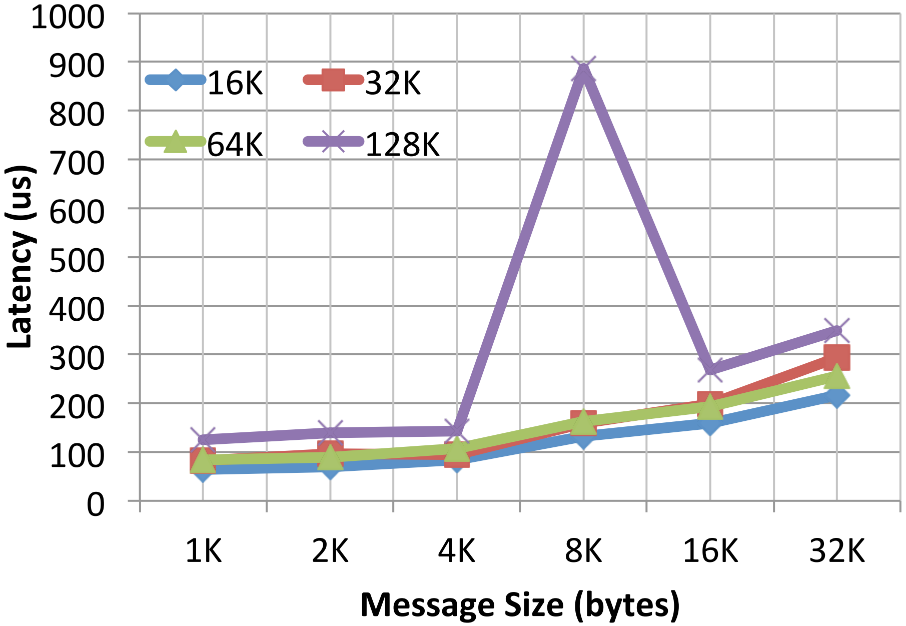


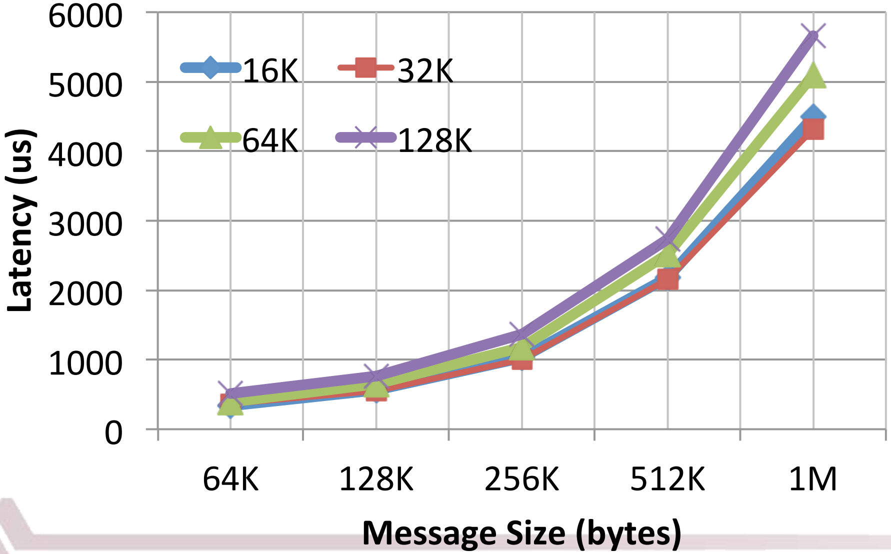


- For a process count over 1K, there is spike in latency at the 256 byte range where bandwidth available starts getting stressed

Performance of MPI_Bcast (16K – 128K Processes)

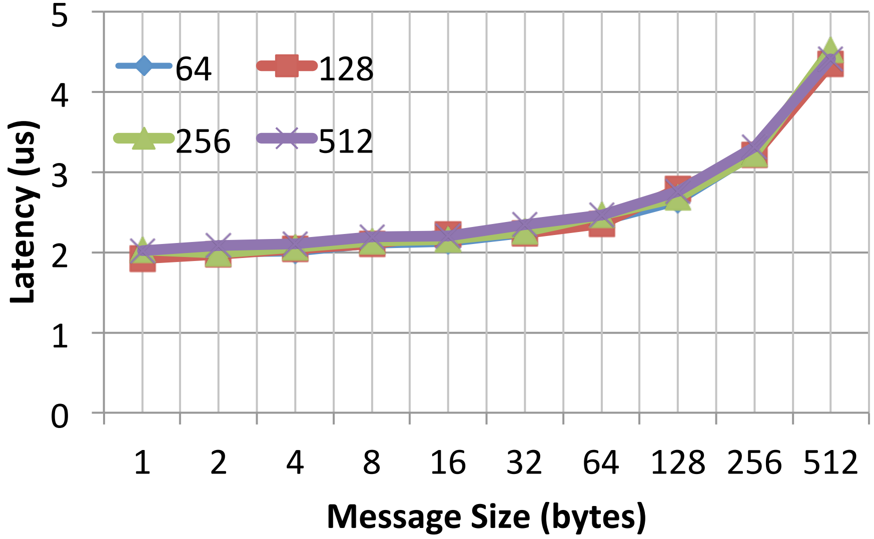




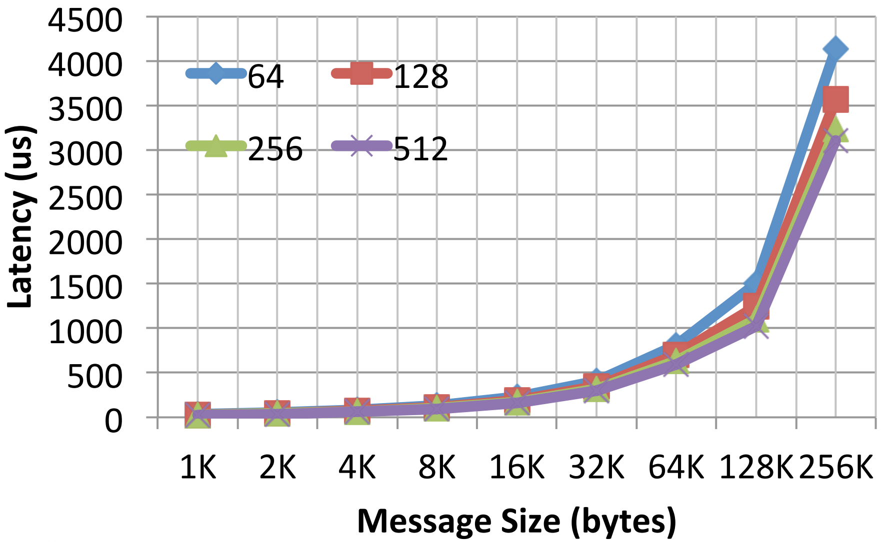


- Unlike the 64 – 8K process count there is variability – possible traffic effect
- The spike at 8K message range is indicative of algorithm selection problem

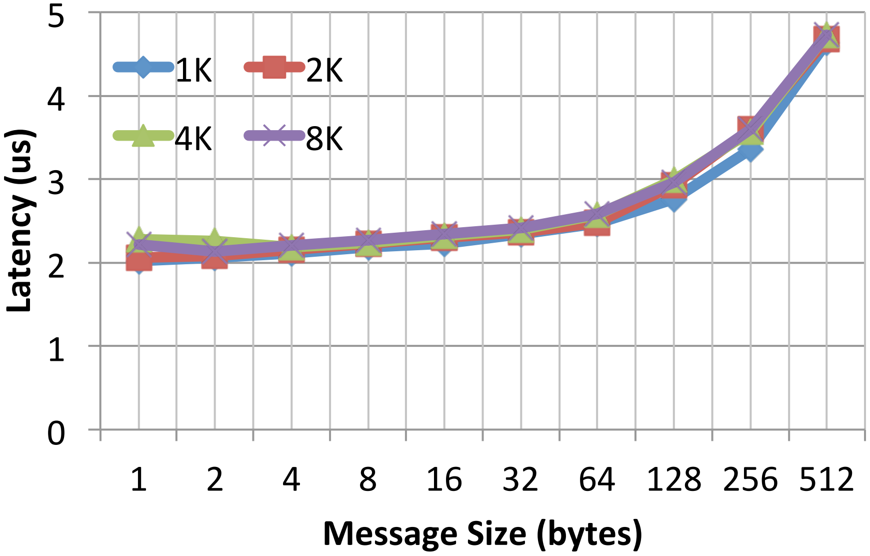
Performance of MPI_Reduce (64 – 512 Processes)

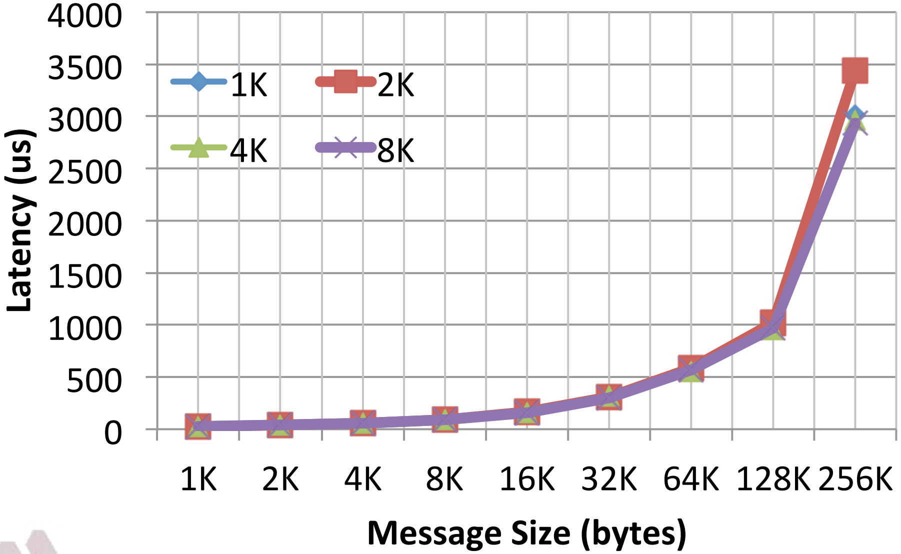


- Reduce latency is hardware accelerated and regardless of process count the latency is similar
- There does seem to be a limitation with hardware acceleration at 128K byte range



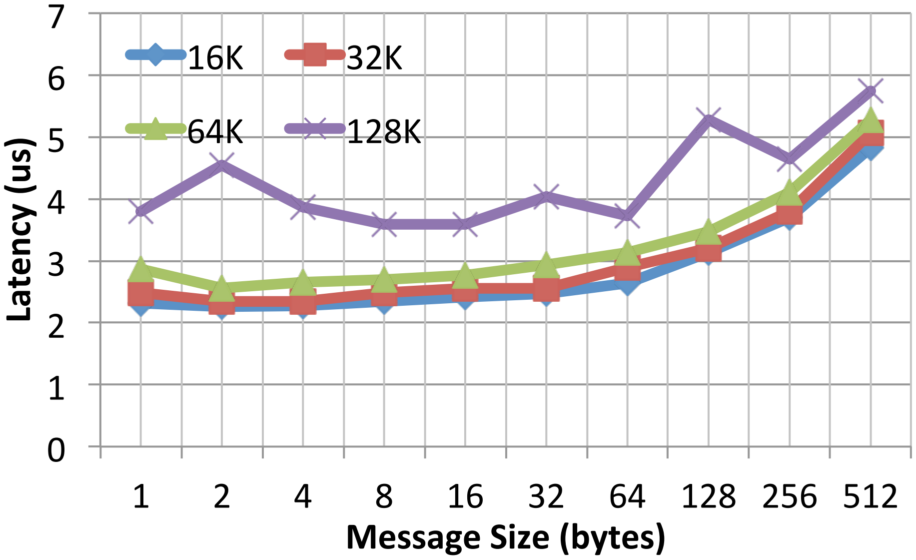
Performance of MPI_Reduce (1K – 8K Processes)

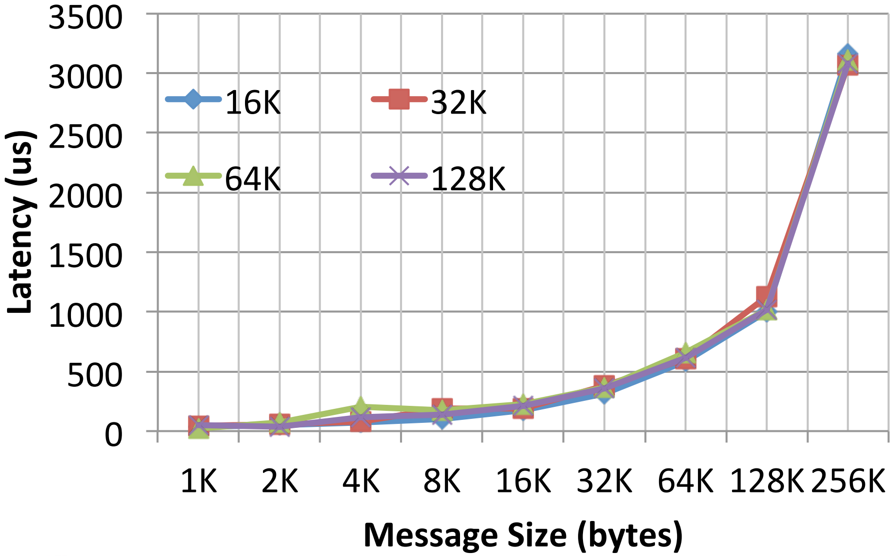




- Trends similar to smaller process count

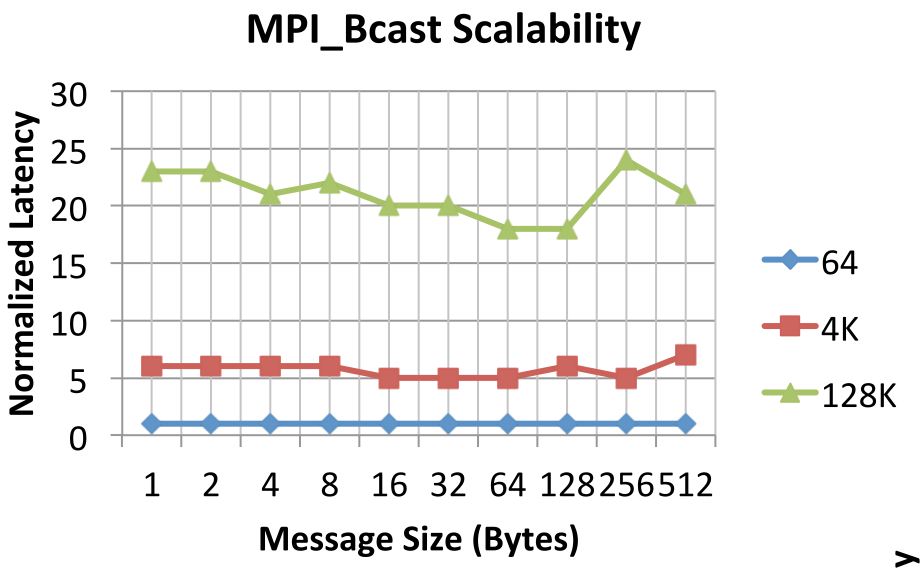
Performance of MPI_Reduce (16K – 128K Processes)



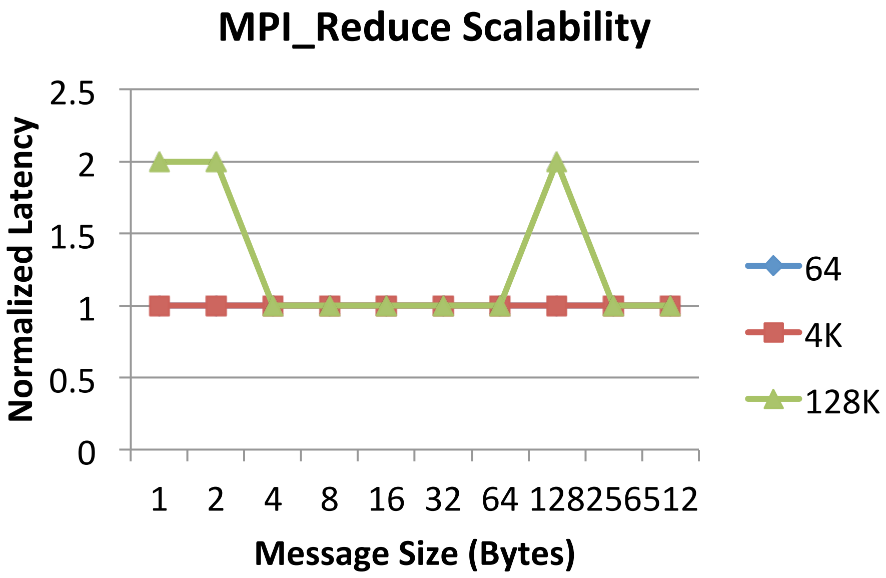


- Notable increase in latency for 128K processes in the short message range

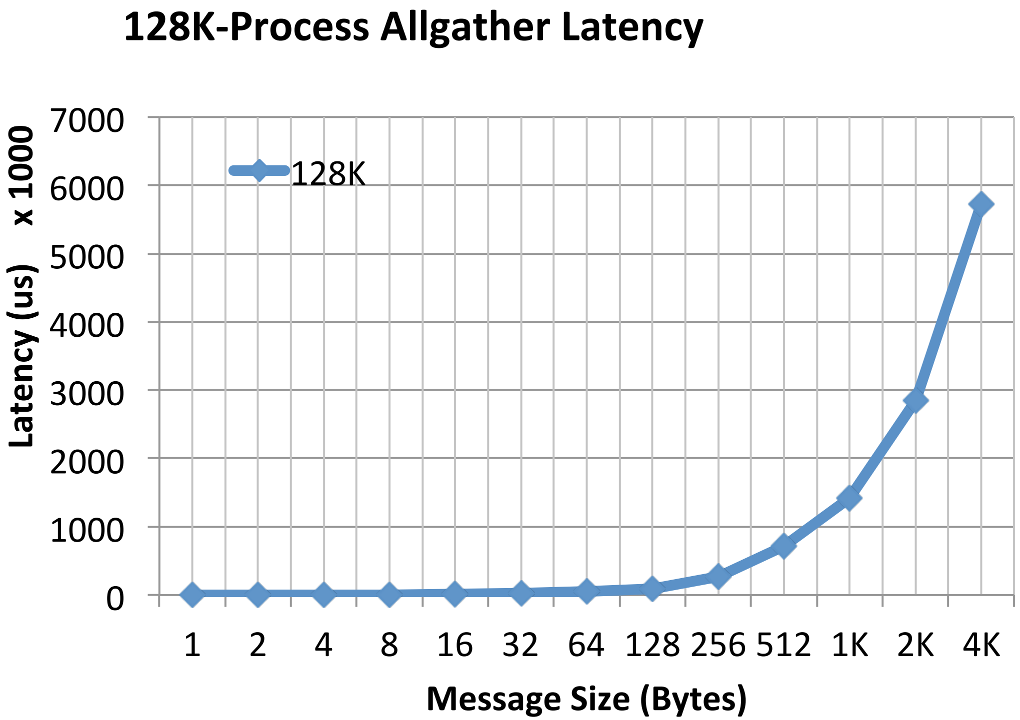
Scalability of MPI_Bcast and MPI_Reduce



- Scalability normalized to 64 process job case
- MPI_Reduce is highly scalable
- MPI_Bcast is not as scalable



Performance of MPI_Allgather (128K Processes)



- Allgather is equivalent to all processes performing broadcasts
- Bandwidth of the interconnection is tested
- Traditionally order of log (N) algorithms applicable to short message allgathers
- The above graph raises an alarm of latency growth for large scale dense collectives

Observations on MPI Collective Performance

- Performance of latency sensitive operations such as Reduce is competitive in the operational range with increasing scale
- Congestion effects, cross job traffic likely to play a role in performance of collectives as job sizes get larger (as seen in the 128K jobs)
- Performance of dense collectives like Allgather suffer from bandwidth limitations =>
  - Applications should perform such collectives in smaller communicators or using non-blocking variant of the collectives
  - Better algorithms need to be devised to overcome bandwidth limitations

Key Questions

- How do MPI collectives perform at extreme scales?
- How well do the CraySHMEM and UPC PGAS collective communications scale?
- Can both the CPU and GPU resources be leveraged effectively in a hybrid node system?

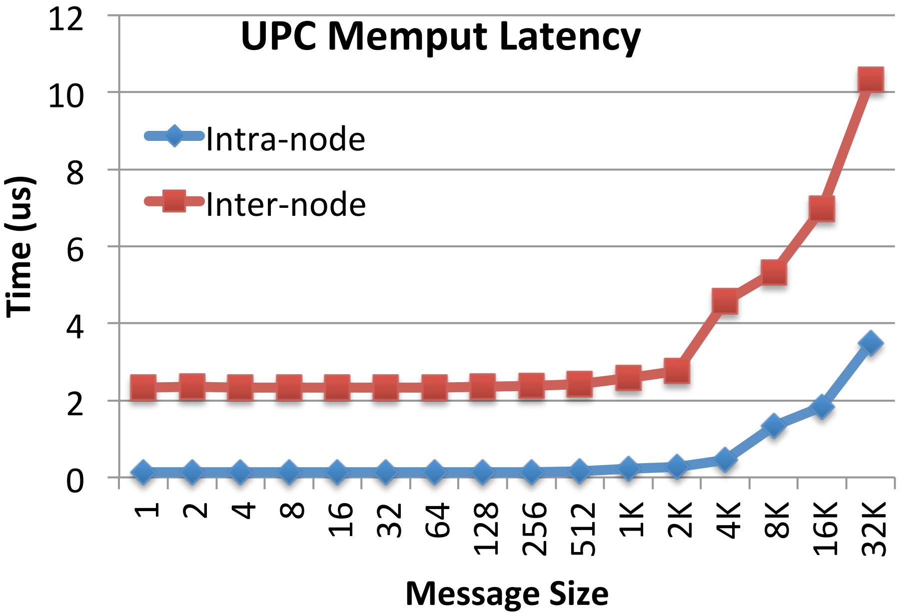
PGAS (UPC/SHMEM) on Blue Waters

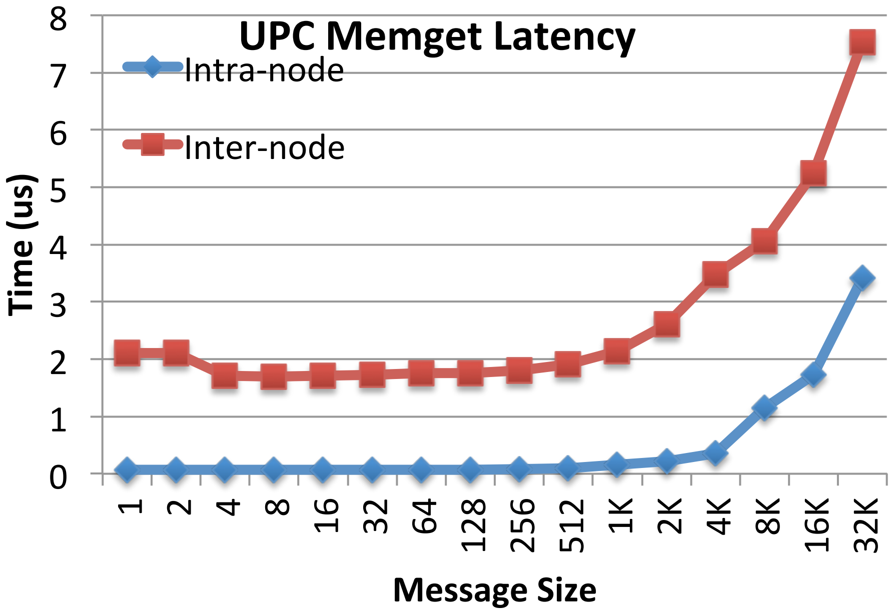
- Partitioned Global Address Space (PGAS) programming models getting more traction
  - Shared memory abstraction over distributed nodes
  - Global view of data and one-sided communication calls
  - Provides improved productivity
  - Can express irregular communication patterns easily
- Unified Parallel C (UPC) – a language based PGAS model
- SHMEM – a library based model
- Blue Waters provides a good platform to evaluate performance of UPC/SHMEM jobs at scale

Blue Waters UPC Performance Evaluations

- Point-to-point operations and Collective operations determine the performance of UPC programs
- Used Cray UPC and OSU UPC Microbenchmarks for evaluations
- Performance of point-to-point operations involve
  - upc_memput
  - upc_memget
- Performance of collectives additionally involves
  - upc_barrier
  - upc_broadcast
  - upc_reduce

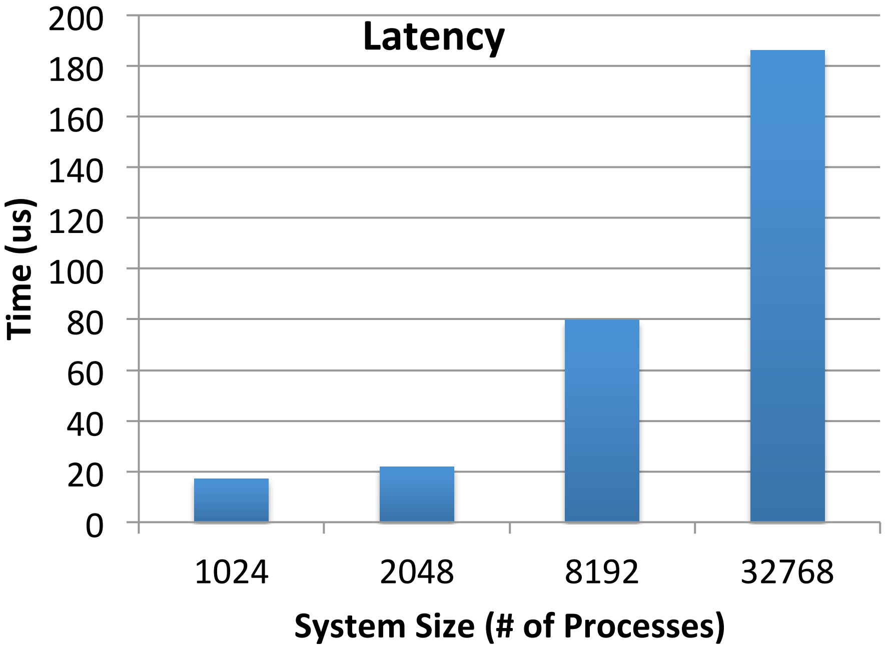
UPC Put/Get Performance

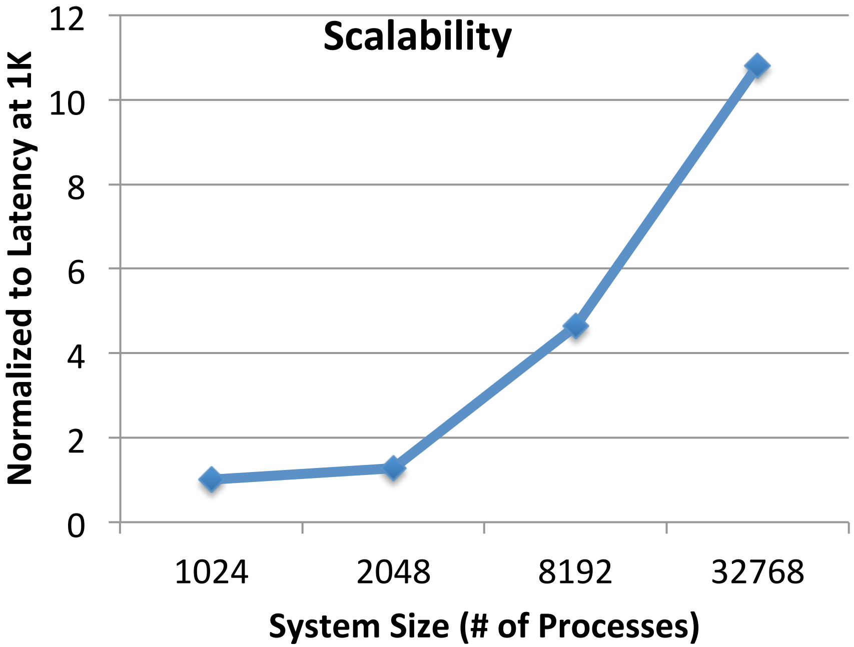




- Latency is flat in the 1 byte – 512 byte range and then starts climbing
  - Latency for UPC Put (intra/inter) for 4 byte message: **0.13/2.34 us**
  - Latency for UPC Get (intra/inter) for 4 byte message: **0.07/1.17 us**
- Higher costs for Put operation might be because of the extra synchronization operation (upc_fence) for ensuring completion

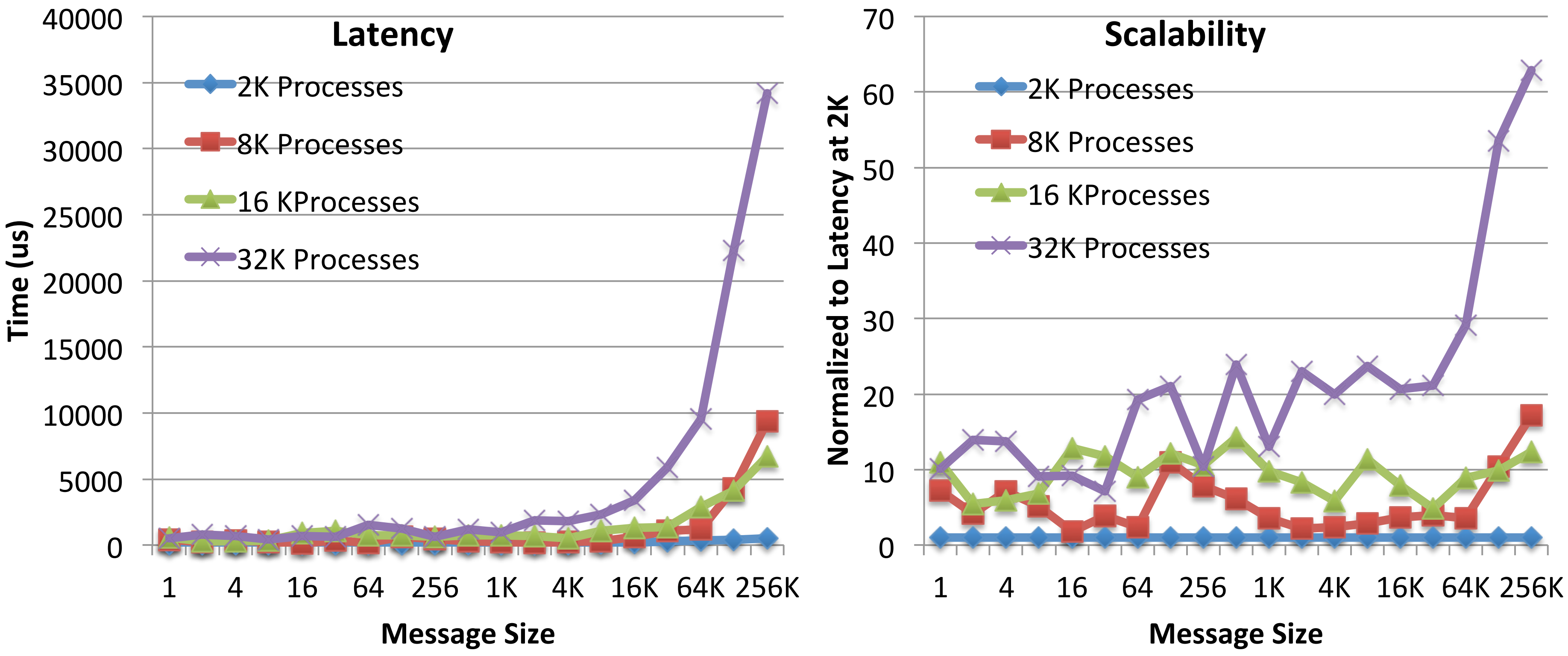
UPC Barrier Performance





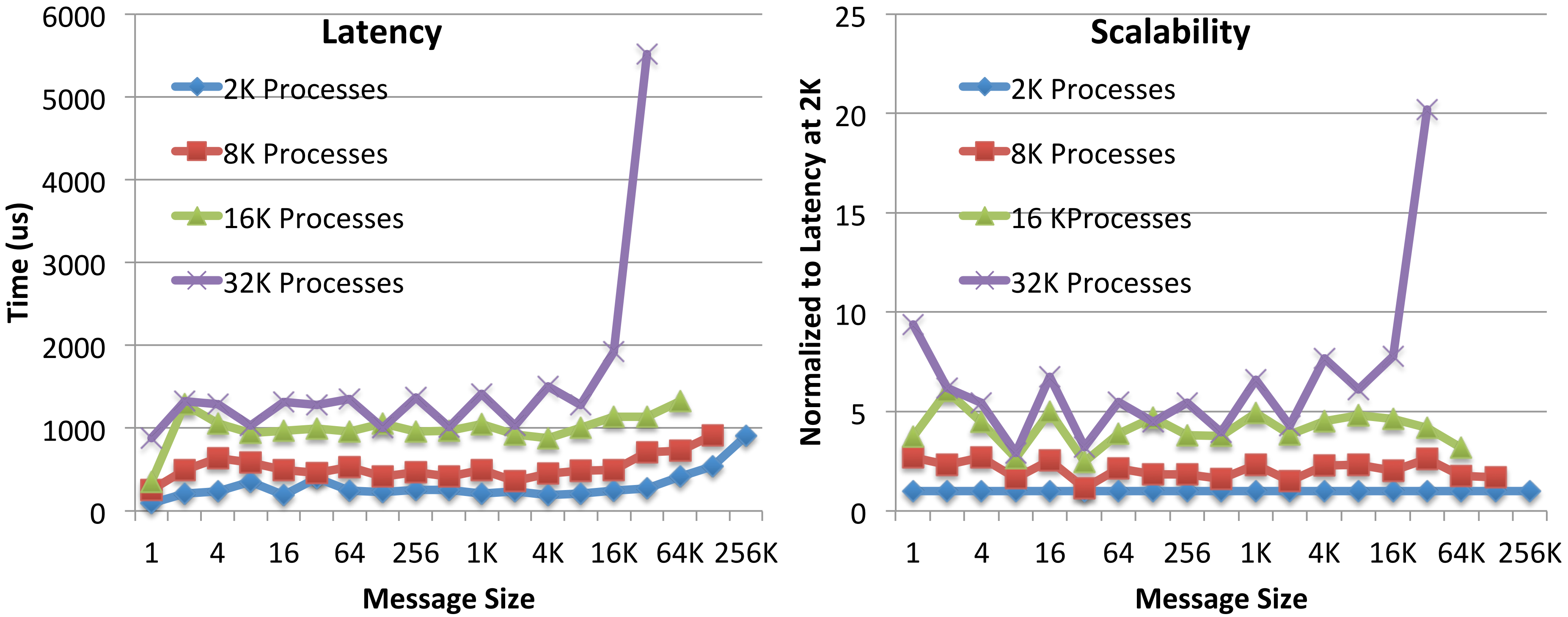
- Barrier Operation Latency at 32,768 process – **186us**
- Scalability graph shows the latency normalized to that at 1,024 processes
- Linear scalability observed for smaller system sizes

UPC Broadcast Performance



- Broadcast Latency for a 4byte message at 32,768 processes – **13us**
- Variation in latencies observed after 8192 processes, and the variation increases with scale
- Broadcast latency does not scale linearly with increase in system size

UPC Reduce Performance

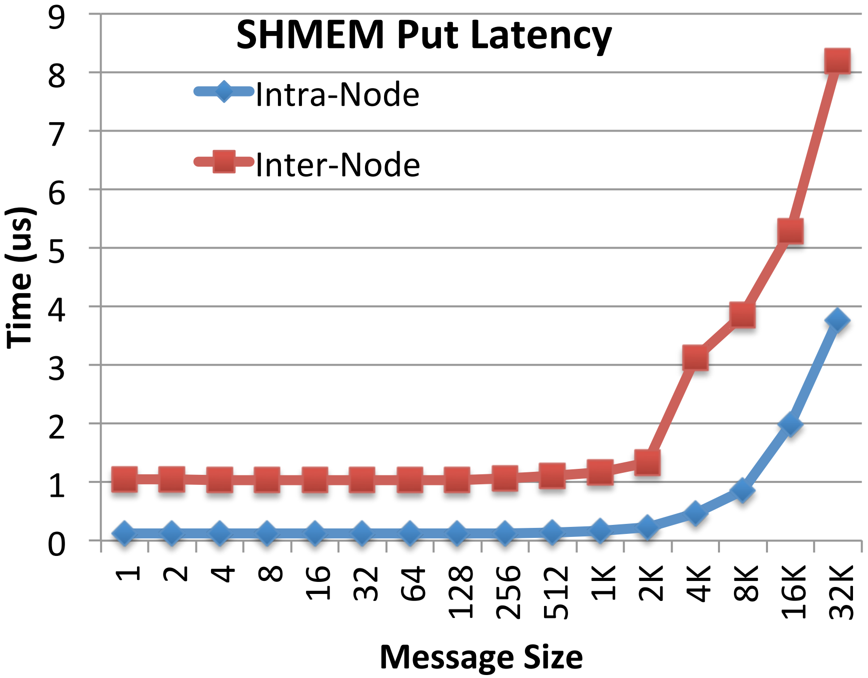


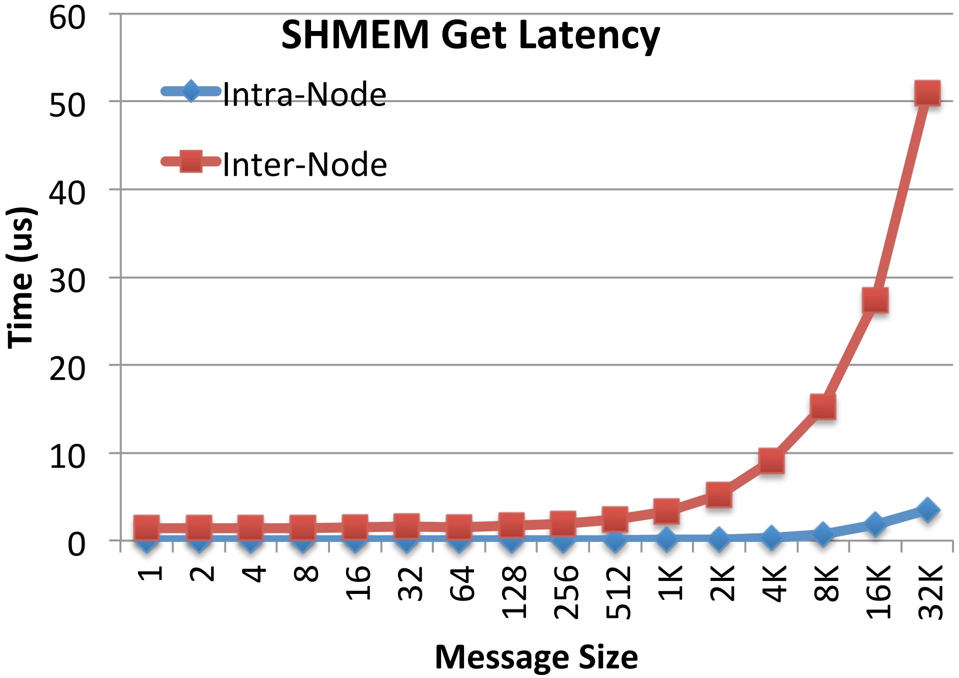
- Reduce Latency for 4 byte message at 32,768 processes – **5.4us**
- Linear scalability observed for small message range
- Variation in operation latency observed as the system size increases

Blue Waters CraySHMEM Performance Evaluations

- Point-to-point operations and Collective operations determine the performance of SHMEM programs
- Used CraySHMEM library and OSU OpenSHMEM Microbenchmarks for evaluations
- Performance of point-to-point operations involve
  - shmem_put
  - shmem_get
- Performance of collectives additionally involves
  - shmem_barrier
  - shmem_broadcast
  - shmem_reduce
  - shmem_collect

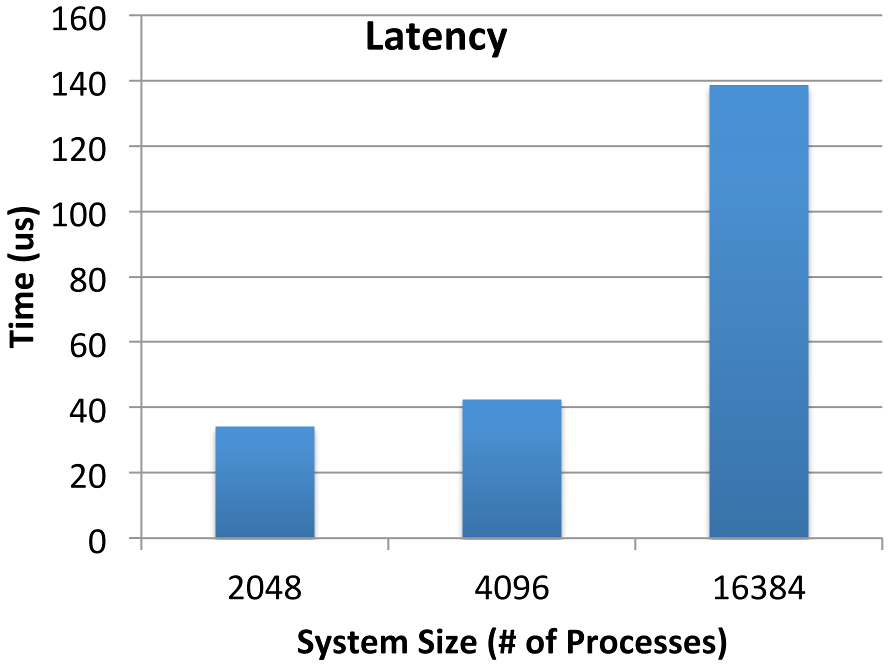
CraySHMEM Put/Get Performance

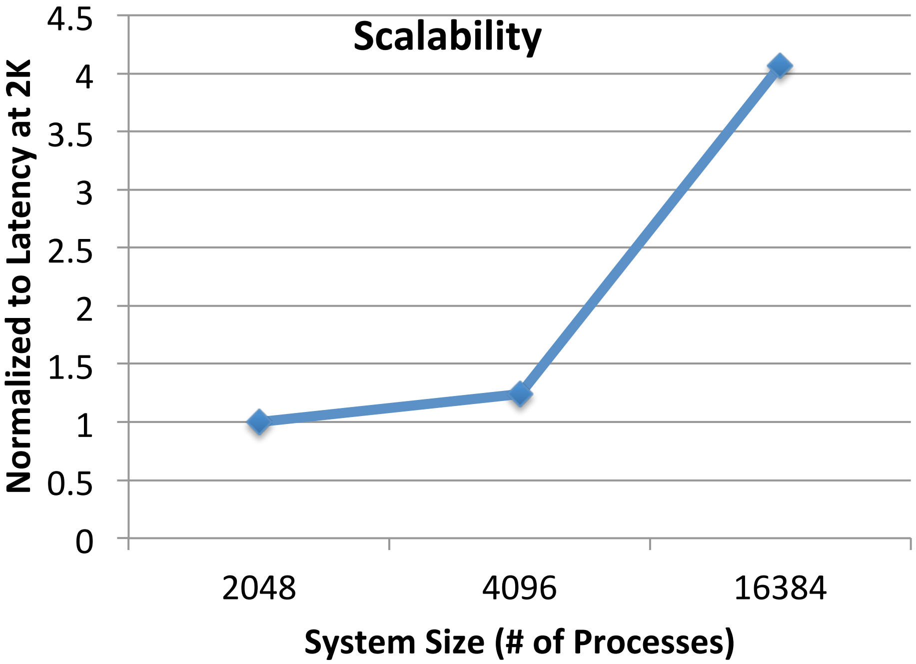




- Latency is flat in the 1 byte – 512 byte range and then starts climbing after 1K bytes
  - Latency for 4byte Put operation (intra/inter) – 0.12/1.04 us
  - Latency for 4byte Get operation (intra/inter) – 0.05/1.41 us
- Significantly higher latency observed for get operation, with increase in message size
  - Get Latency for 512K message – 763 us

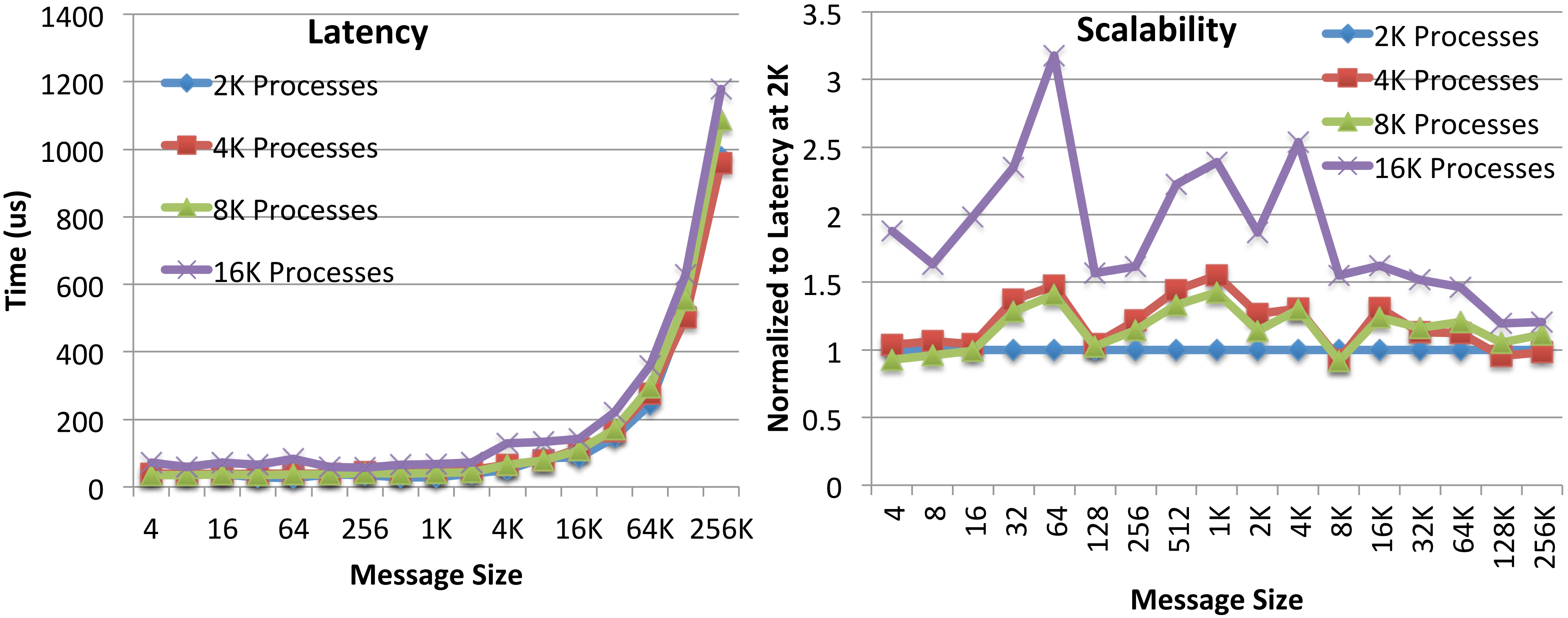
CraySHMEM Barrier Performance





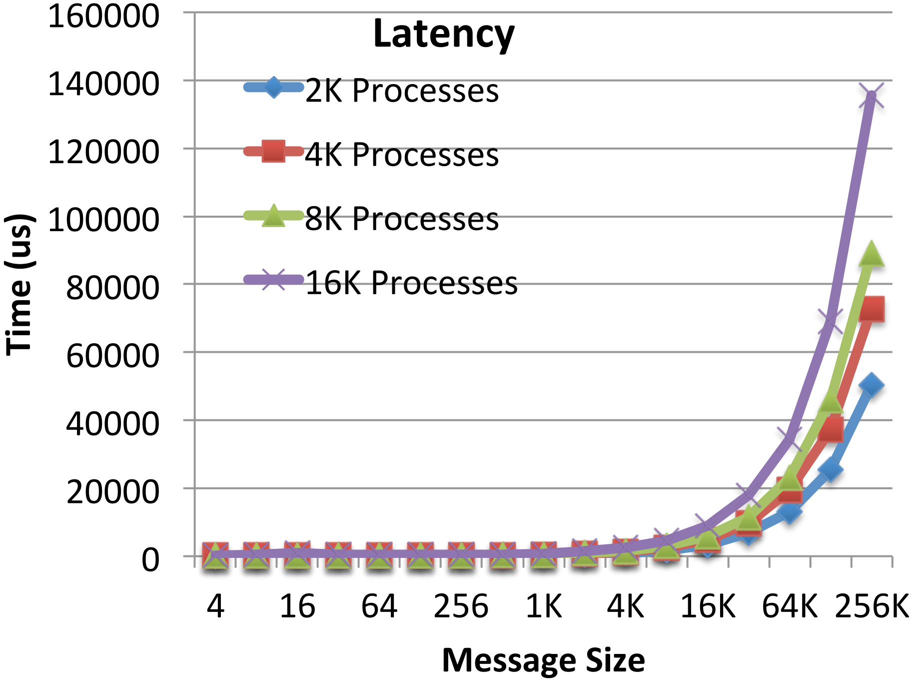
- Barrier Latency at 16,384 processes – **138.64 us**
- Similar latencies as that of UPC barrier
- Shows good scalability trends with increase in system size

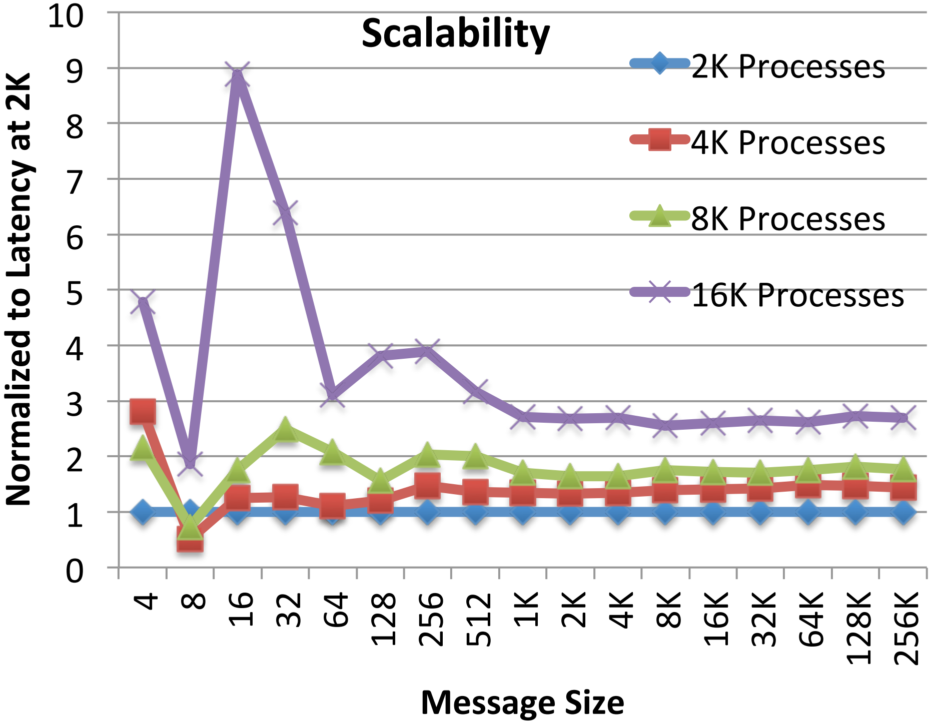
CraySHMEM Broadcast Performance



- Latency is flat in the 1 byte – 512 byte range and then starts climbing – regardless of process count
- Broadcast Latency for 4-byte message at 16,384 processes – **72.3us**
- Variation in latencies observed with increase in system size

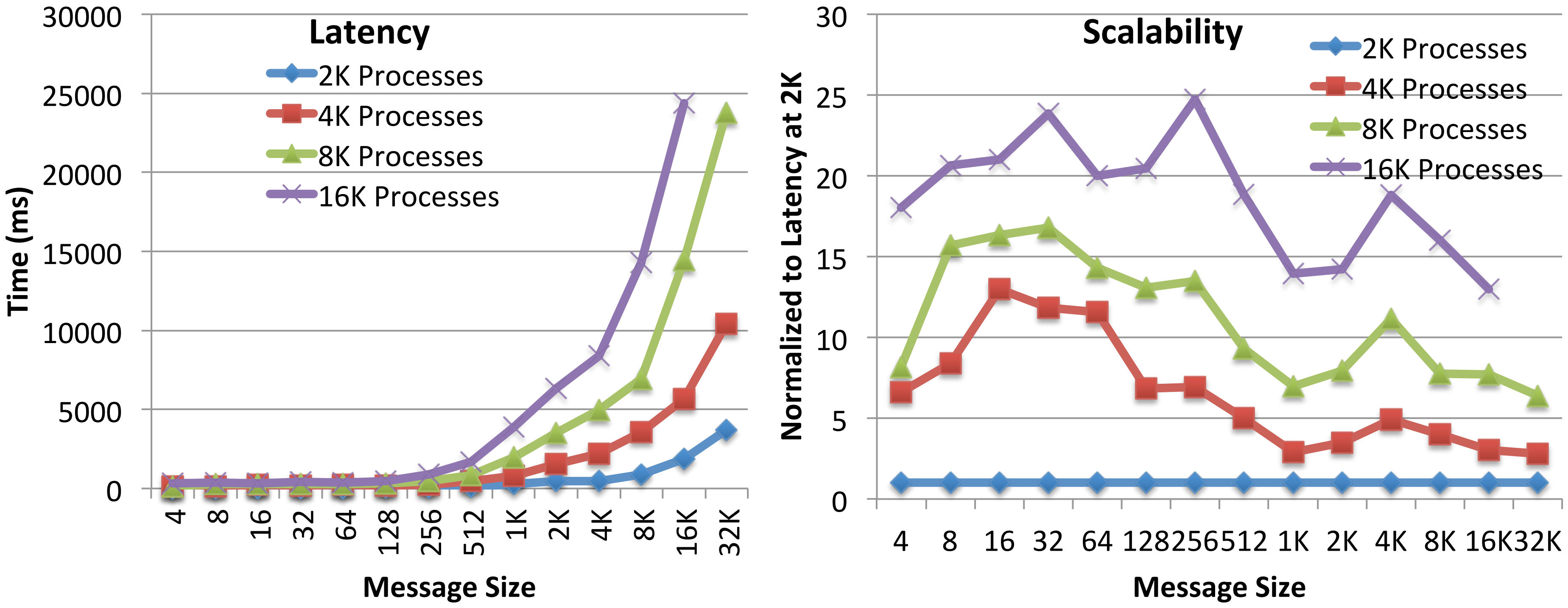
CraySHMEM Reduce Performance





- Latency for 4-byte message at 16K processes – **210 us**
- Scalability analysis shows good scalability trends with even higher system sizes as well
- Latencies smaller compared to UPC reduce operation – extra synchronization operations in UPC collective operations

CraySHMEM Collect Performance



- Latency for 4byte collect (all-gather) operation at 16K processes – 319.3 ms
- Scalability analysis shows collect operation scales well

Key Questions

- How do MPI collectives perform at extreme scales?
- How well do the CraySHMEM and UPC PGAS collective communications scale?
- Can both the CPU and GPU resources be leveraged effectively in a hybrid node system?

Current Execution of HPL on Heterogeneous GPU Clusters

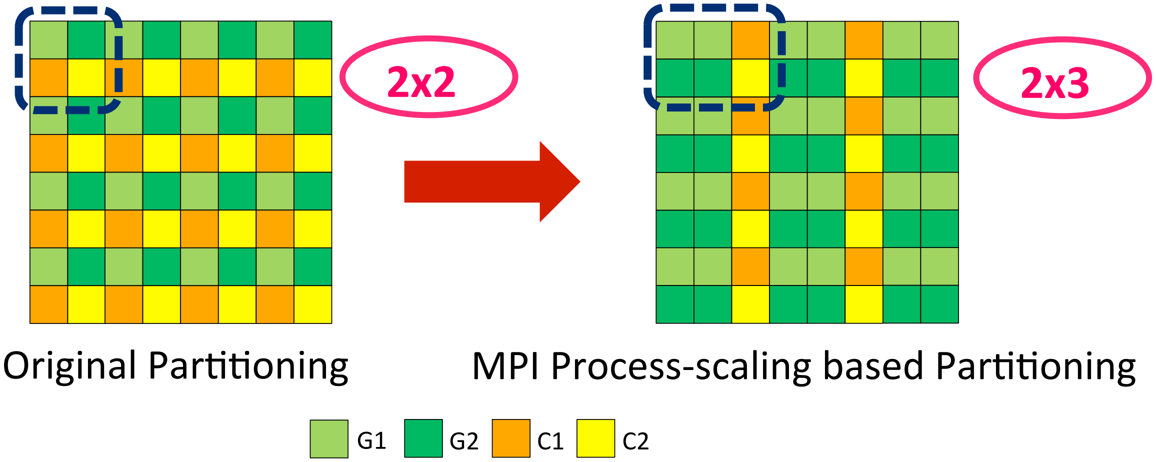
- HPL (High Performance Linpack)

  Benchmark for ranking supercomputers in the top500 list

- Current HPL support for GPU Clusters
  - Heterogeneity inside a node CPU+GPU
  - Homogeneity across nodes
- Current HPL execution on heterogeneous GPU Clusters
  - Only CPU nodes (using all the CPU cores)
  - Only GPU nodes (using CPU+GPU on only GPU nodes)
  - As the ratio CPU/GPU is higher => report the "Only CPU" runs
- Hybrid HPL support for heterogeneous systems
  - Heterogeneity inside a node (CPU+GPU)
  - Heterogeneity across nodes (nodes w/o GPUs)

R. Shi, S. Potluri, K. Hamidouche, X. Lu, K. Tomko and D. K. Panda, A Scalable and Portable Approach to Accelerate Hybrid HPL on Heterogeneous CPU-GPU Clusters, IEEE Cluster (Cluster '13), Best Student Paper Award

Two Level Workload Partitioning: Inter-node



Original Partitioning

MPI Process-scaling based Partitioning

- **Inter-node Static Partitioning**

  Original design: uniform distribution, bottleneck on CPU nodes

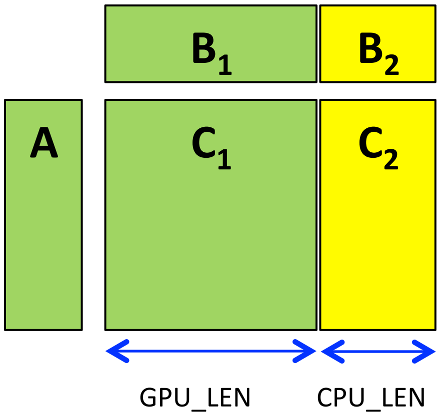
  New design: identical block size, schedules more MPI processes on GPU nodes

  MPI_GPU = ACTUAL_PEAK_GPU / ACTUAL_PEAK_CPU + β

  (NUM_CPU_CORES **mod** MPI_GPU = 0)

  Evenly split the cores

Two Level Workload Partitioning: Intra-node



- **Intra-node Dynamic Partitioning**
  - MPI-to-Device Mapping

    Original design: 1:1

    New design: M: N (M > N), N= number of GPUs/Node, M= number of MPI processes

  - Initial Split Ratio Tuning: alpha = GPU_LEN / (GPU_LEN + CPU_LEN)

    Fewer CPU cores per MPI processes

    Overhead caused by scheduling multiple MPI processes on GPU nodes

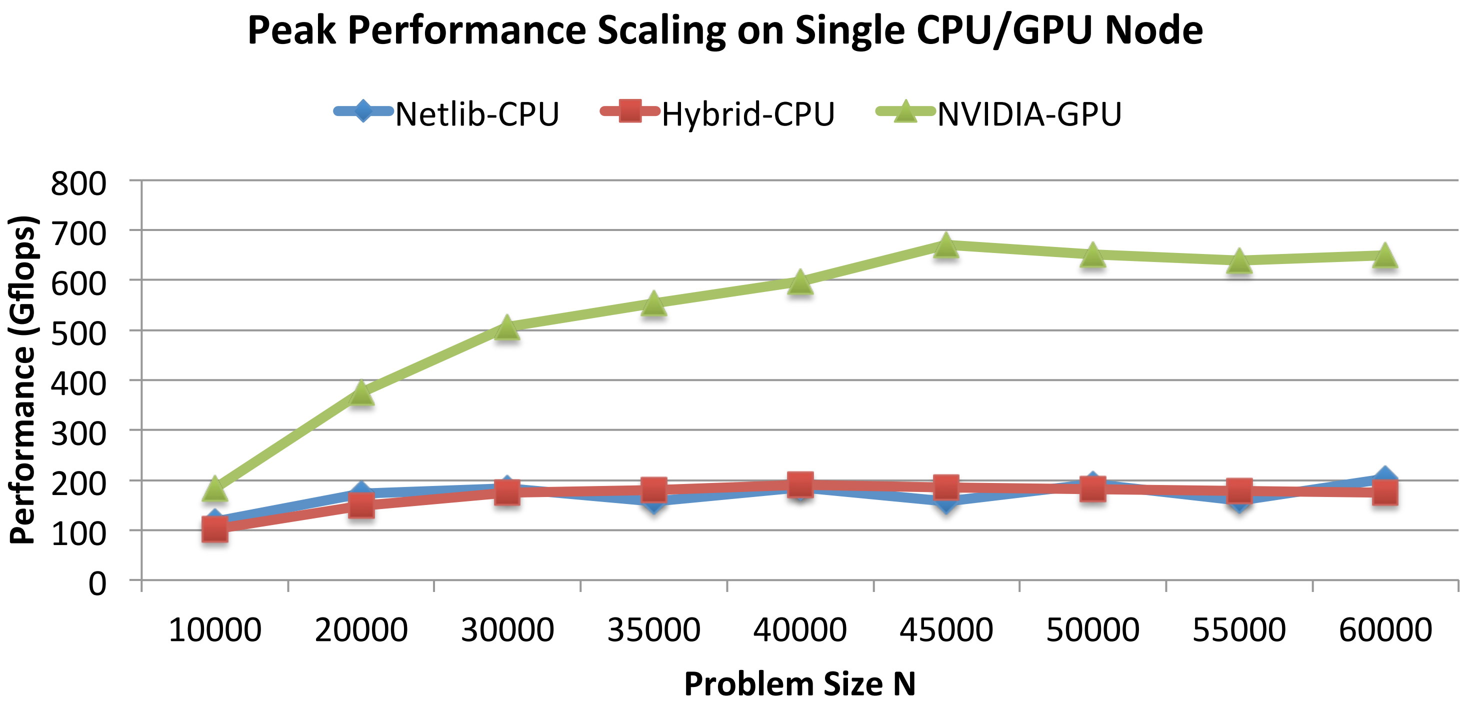
Performance Tuning of Single CPU Node and GPU Node

Netlib-CPU: Standard HPL version from Netlib (UTK)

Hybrid-CPU: Hybrid HPL version with OpenMP support

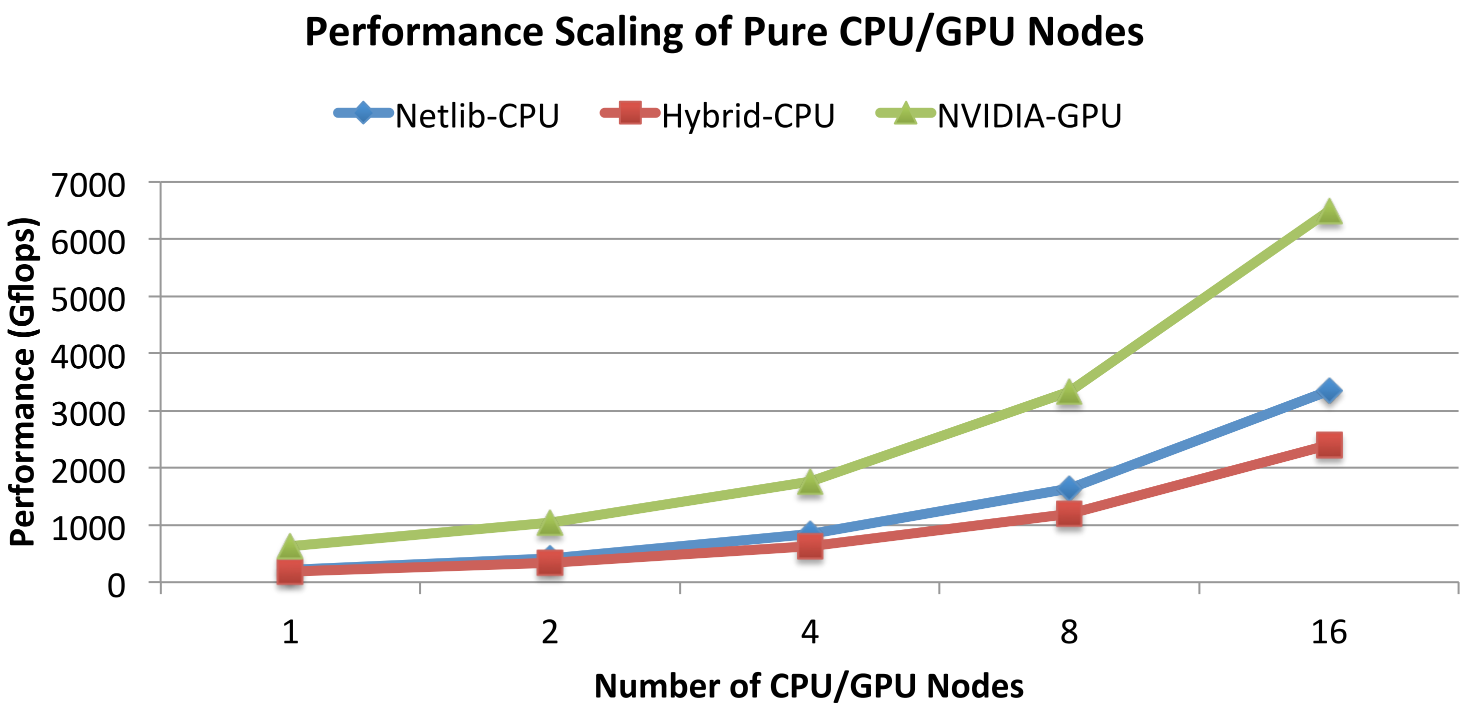
NVIDIA-GPU: NVIDIA's HPL version

* OpenBLAS Math Library is used



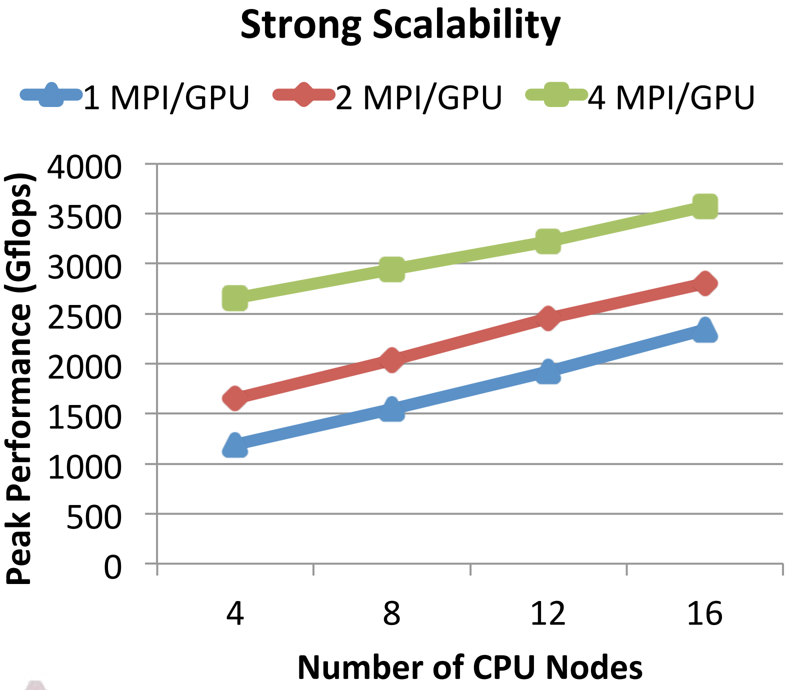
Peak Performance Scaling of Pure CPU/GPU Nodes

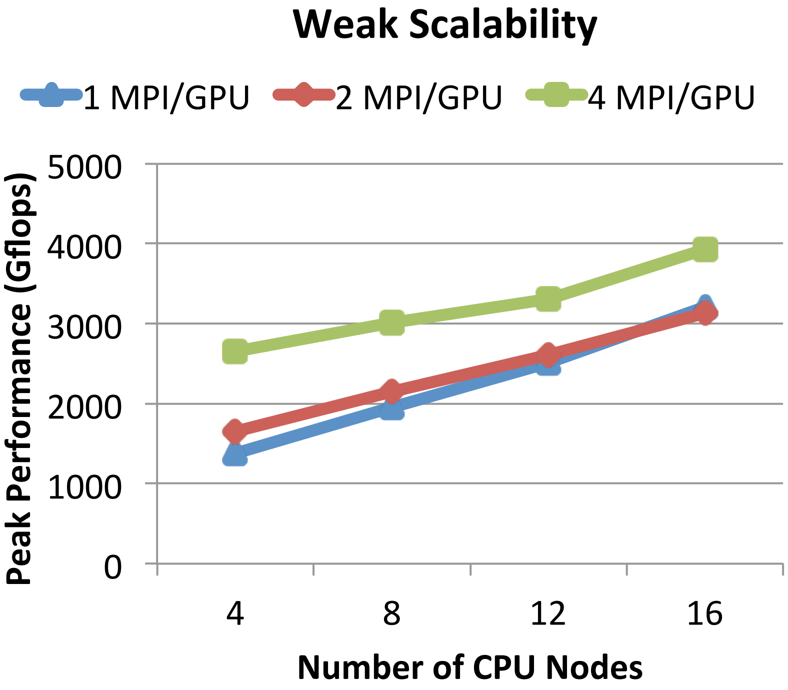
Measure the peak performance of either pure CPU Nodes or pure GPU Nodes (1, 2, 4, 8, 16)



Strong and Weak Scalability of Hybrid CPU+GPU Nodes

- Using Hybrid-HPL to measure the scalability with 4 GPU Nodes + (4, 8, 12, 16) CPU Nodes
- Launch 1 MPI process / CPU node; 1, 2 or 4 MPI processes / GPU node
- Strong Scalability: fixed problem size N for each combination of CPUs+GPUs (e.g. N=100,000 for 4 GPUs + 4 CPUs)
- Weak Scalability: fixed memory usage (~40%) on GPU nodes for all cases





Peak Performance of Hybrid CPU Nodes + GPU Nodes

Measure the peak performance of 64 CPU Nodes and 16 GPU Nodes

Launch 1 MPI process / CPU node, and 4 MPI processes / GPU node

Node Configuration	Peak Performance (Gflops)
16 GPUs	6,480
64 CPUs	13,210
16 GPUs + 64 CPUs	14,520

Peak Performance Efficiency (Hybrid-HPL)

Peak Perf. of hybrid Nodes / (Peak Perf. of CPUs + Peak Perf. of GPUs)

(e.g. 14,520 / (6,480 + 13,210) = 73.7 %

Conclusion

- The Blue Waters system provides unique opportunities
  - Communications at large scale
  - Hybrid system with XE6 and XK7 nodes
- MPI collectives study on up to 128K processes
  - Latency sensitive collectives such as reduce perform well
  - Bandwidth limitations impact dense collectives such as Allgather
- UPC and SHMEM communications study up 32K and 16K cores respectively
  - UPC and SHMEM point-to-point performance is good
  - Some collectives (UPC Scatter, SHMEM Broadcast) scale well, for others (SHMEM collect) we observed high latencies

Conclusion (continued)

- Hybrid HPL
  - Peak single CPU node performance 202 Gflops/sec
  - Peak GPU node performance 670 Gflops/sec
  - Performance efficiency of hybrid HPL compared to the sum of pure CPU and GPU nodes, above 70% efficiency with 16 GPU nodes and 64 CPU nodes.

- Contact US:

Karen Tomko
Ohio Supercomputer Center
E-mail: ktomko@osc.edu
http://www.osc.edu/~ktomko

Dhabaleswar K. (DK) Panda
The Ohio State University
E-mail: panda@cse.ohio-state.edu
http://www.cse.ohio-state.edu/~panda