High Performance Middlewares for Next Generation Architectures: Challenges and Solutions

Sourav Chakraborty, Dhabaleswar K Panda (Advisor), The Ohio State University

Overview

Current Trends in HPC

- Tremendous increase in system and job sizes
- Dense multi-/many-core CPUs (Xeon/KNL/Power)
- High-performance Interconnects (InfiniBand, Omni-Path)
- MPI and hybrid MPI+PGAS models very popular

Major Challenges

Job-startup performance

- Inefficient address exchange mechanisms
- Does not utilize High-performance interconnects
- Poor memory scalability due to data duplication

Collective communication

- Kernel-level contention limits performance
- Exacerbated by many-core CPUs

Point-to-point communication

- Inefficient usage of resources and lack of overlap
- Does not consider the overall communication pattern
- Lack of dynamic and adaptive protocols

Fault tolerance

- Inefficient failure detection and propagation
- Lack of scalable recovery mechanisms

Proposed Solutions

Improving job-startup performance[1-4]

- Exchange addresses over high-performance networks and use more efficient algorithms
- Overlap address exchange with other tasks
- Co-design middleware and launcher to expose information through shared memory regions

Avoiding kernel-level contention[5]

- Propose contention-aware kernel-assisted collectives

Cooperative rendezvous protocols[6]

- Dynamic and adaptive rendezvous protocols
- Consider overall communication pattern and CPU load
- Improve overlap of intra-node and inter-node communication through message chunking

Scalable fault-tolerance and recovery[7]

- Co-design middleware and resource manager to efficiently detect and recover from process and node failures

Key Results

Job-startup performance

- 10x and 30x improvement in startup time of MPI and OpenSHMEM respectively (16K processes, 1K nodes)
- 4GB memory saved per node (64 ppn, 1M processes)

Contention-aware Collectives

- Up to 5x, 4x, and 14x faster on KNL, Broadwell, and OpenPOWER respectively

Cooperative rendezvous protocols

- 19%, 16%, and 10% improvement with Graph500, CoMD, and MiniGhost with 896 processes

Scalable fault-tolerance and recovery

- Up to 4x improvement in application recovery time from process failures with 4K processes

References

[1] On-demand Connection Management for OpenSHMEM and OpenSHMEM+MPI. (S. Chakraborty et al, HIPS '15)
[2] PMI Extensions for Scalable MPI Startup. (S. Chakraborty et al, EuroMPI/Asia '14)
[3] Non-blocking PMI Extensions for Fast MPI Startup. (S. Chakraborty et al, CCGrid '15)
[4] SHMEMPMI – Shared Memory based PMI for Improved Performance and Scalability. (S. Chakraborty et al, CCGrid '16)
[5] Contention-Aware Kernel-Assisted MPI Collectives for Multi-/Many-core systems. (S. Chakraborty et al, Cluster '17) [Best Paper Finalist]
[6] Cooperative Rendezvous Protocols for Improved Performance and Overlap (S. Chakraborty et al, to be presented in SC '18) [Best Student Paper Finalist]
[7] EReinit: Scalable and Efficient Fault-Tolerance for Bulk-Synchronous MPI Applications (S. Chakraborty et al, ExaMPI '17)

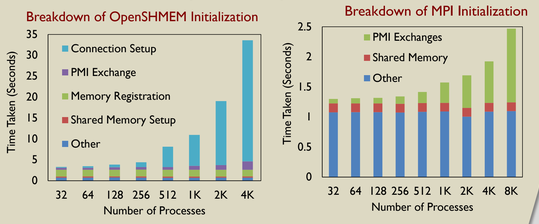
More Information

Most of the designs have been incorporated into the MVAPICH2 MPI library.
It is publicly available for download from: http://mvapich.cse.ohio-state.edu
Homepage: http://cse.osu.edu/~chakrabs
Publications: http://nowlab.cse.ohio-state.edu/member/chakrabs
Email: chakraborty.52@osu.edu

Challenges

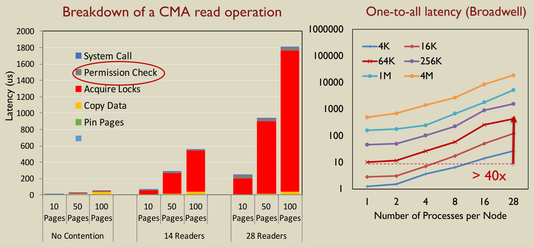
Job Startup

- RDMA operations require exchanging information about memory segments registered with the HCA
- Existing PMI (Process management Interface) primitives used to achieve this are inefficient
- Does not use High-performance networks
- Address information duplicated on all processes - increased memory consumption



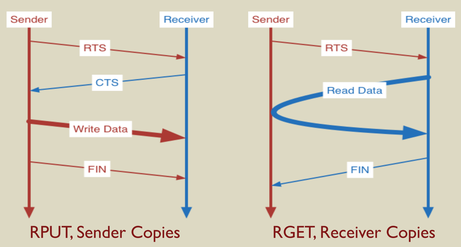
Collective Communication

- Shared memory copies require two copies
- Kernel assisted transfers require single copy
- CMA (Cross-Memory Attach) is commonly used by MPI libraries to read/write data from peers
- Multiple processes accessing the same source/target process leads to kernel-level lock contention



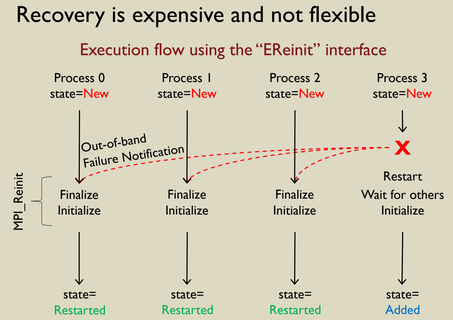
Point-to-Point Communication

- Existing rendezvous protocols use only the sender or the receiver CPU to copy data
- Does not utilize all available resources
- Static protocols do not consider the overall communication pattern
- Low overlap between intra-node and inter-node communication



Fault Tolerance

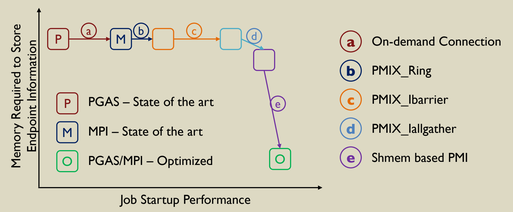
- MPI has limited fault-tolerance (FT) support
- Existing FT solutions are not suitable for Bulk-Synchronous Processing (BSP) applications
- Failure detection and notification is difficult
- Recovery is expensive and not flexible



Solution

Job Startup

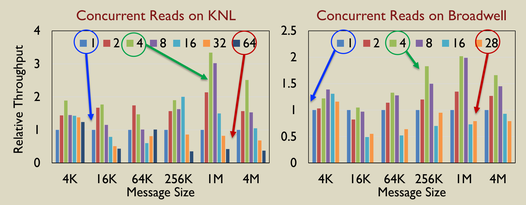
- On-demand connection management for MPI, PGAS, and MPI+PGAS models
- Proposed efficient and non-blocking PMI primitives (PMIX_Ring, PMIX_Ibarrier, PMIX_Iallgather)
- Utilize high-performance networks for bootstrapping
- Share memory between job-launcher and processes to reduce data duplication and memory consumption



Collective Communication

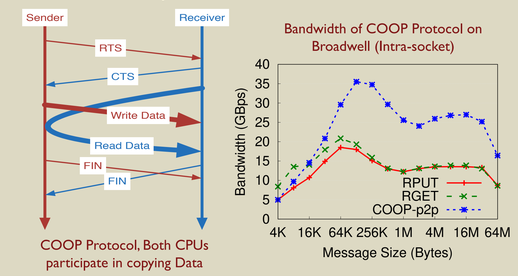
- Propose contention-aware collective designs
- Limit number of concurrent read/writes on the same source/target process
- Avoid control message exchange overhead

- High concurrency ⇒ High Contention ⇒ Low Throughput
- No concurrency ⇒ Low utilization ⇒ Low Throughput
- Medium concurrency ⇒ Best of Both ⇒ High Throughput



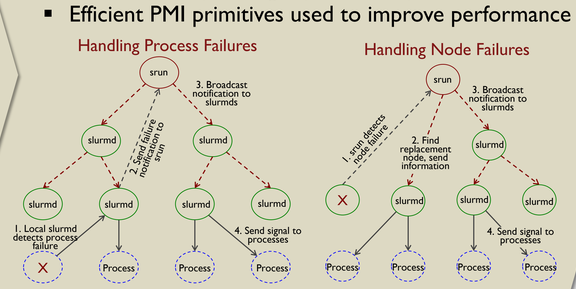
Point-to-Point Communication

- Propose novel "Cooperative" rendezvous protocols
- Utilize both sender and receiver CPUs
- Dynamically select the best rendezvous protocol based on the overall communication pattern
- Process control messages while copying data for better overlap of different communications



Fault Tolerance

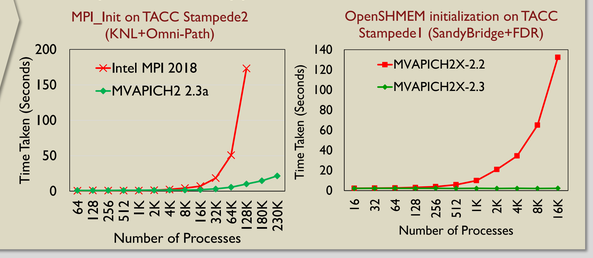
- Proposed EReinit (Efficient Reinitialization) interface
- Co-design MPI runtime and resource manager
- Resource manager (Slurm) provides failure detection and propagation using daemon processes
- Recovery by spawning replacement processes
- Efficient PMI primitives used to improve performance



Results

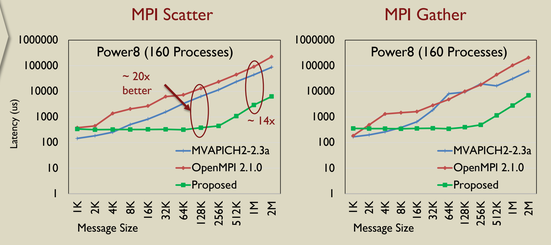
Job Startup

- 10x and 30x improvement in MPI and OpenSHMEM startup on 16K processes (1K nodes)
- MPI initialization under 22s on 230K processes on 3.5K KNL nodes with 64 processes per node
- 8.8x faster than Intel MPI with 128K processes
- Memory footprint reduced by 4GB per node with 1M processes, 64 ppn



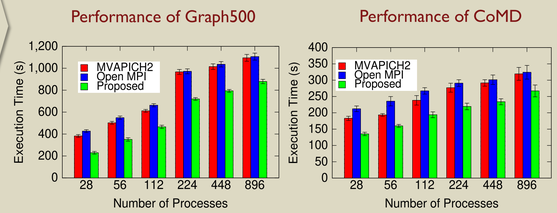
Collective Communication

- Up to 14x improvement with Scatter and Gather
- Up to 4x improvement for Allgather and Alltoall
- Applicable to KNL, Broadwell, and OpenPOWER
- Also improves performance of OpenSHMEM, UPC, and UPC++ collectives



Point-to-Point Communication

- Up to 2x improvement in latency and bandwidth
- Better load balancing across CPU cores
- Up to 10% improvement with 3dstencil benchmark
- Up to 19%, 16%, and 10% improvement with Graph500, CoMD, and MiniGhost on Broadwell+EDR with 896 processes



Fault Tolerance

- Recovers from single process and single node failure in 2.25 and 4.41 seconds respectively
- Can recover from parallel file system as well as in-memory checkpoints
- Up to 4x faster than job restart with 4K processes on 256 nodes