NETWORK-BASED COMPUTING LABORATORY

Optimizing Collective Communication in UPC

Jithin Jose, Khaled Hamidouche, Jie Zhang,
Akshay Venkatesh, and Dhabaleswar K. (DK) Panda

Network-Based Computing Laboratory
Department of Computer Science and Engineering
The Ohio State University, USA

OHIO STATE

Outline

- Introduction & Motivation
- Problem Statement
- Design
- Performance Evaluation
- Conclusion & Future work

Introduction

- MPI - the de-facto programming model for scientific parallel applications
- Offers attractive features for High Performance Computing (HPC) applications
  - Blocking and non blocking pt-to-pt, Collectives, One sided, etc.
- MPI Libraries (MVAPICH2, OpenMPI, IntelMPI) over InfiniBand optimized to the hilt
- Emerging Partitioned Global Address Space (PGAS) models - Unified Parallel C (UPC), OpenSHMEM - offer better programmability

Unified Parallel C (UPC)

- PGAS Models
  - Shared memory abstraction over distributed systems
  - Easier to express irregular communication patterns
  - Better programmability
- UPC (https://upc-lang.org/)
  - Compiler based PGAS model
  - Parallel Extensions to the C standard
  - UPC Specifications and Standards:
    - UPC Language Specifications, v1.2 (June 2005)
    - UPC Language Specifications, v1.3 (November 2013)

Collective Communication Operations in MPI/PGAS

- Collective communication primitives offer a flexible, portable way to implement group communication operations
- Used across various scientific applications
- Supported across both MPI and PGAS models.
- High-performance MPI implementations have incorporated optimizations for modern architectures
  - Optimized using multi-core-aware, network-aware, kernel assisted, mechanisms and optimized algorithms

Collective Communication in UPC

- UPC provides global view of data
  - Each thread can read global data and operate on it
  - Requires additional synchronizations for user-mode collective operations
- UPC standard defines various collective operations
- Earlier research has shown performance limitations for UPC collectives compared to MPI collectives
  - There have been studies to improve UPC collectives
  - Needs improvement in performance and scalability
- Can we leverage the entire gamut of designs that are available in high performance MPI implementations?

Outline

- Introduction & Motivation
- Problem Statement
- Design
- Performance Evaluation
- Conclusion & Future work

Problem Statement

- Can we improve UPC collective operations by efficiently mapping them on to MPI collectives?
- Can we design a light-weight and scalable interface to improve the UPC collectives performance through leveraging MPI-level designs?
- What are performance benefits of our proposed approach across various micro-benchmarks and UPC applications?

Outline

- Introduction & Motivation
- Problem Statement
- Design
- Performance Evaluation
- Conclusion & Future work

Collective Operations in UPC

- Relocation Operations:
  - Relocates global data based on the collective operation
  - upc_all_broadcast, upc_all_scatter, upc_all_gather, upc_all_gather_all, upc_all_exchange, upc_all_permute
- Computational Operations
  - Exchange global data, and operate on it, as defined by the operation and data types
  - upc_all_reduceT, upc_all_prefix_reduceT
- Synchronization Mode
  - UPC_IN_XSYNC | UPC_OUT_YSYNC
  - X and Y can be NO, MY, ALL

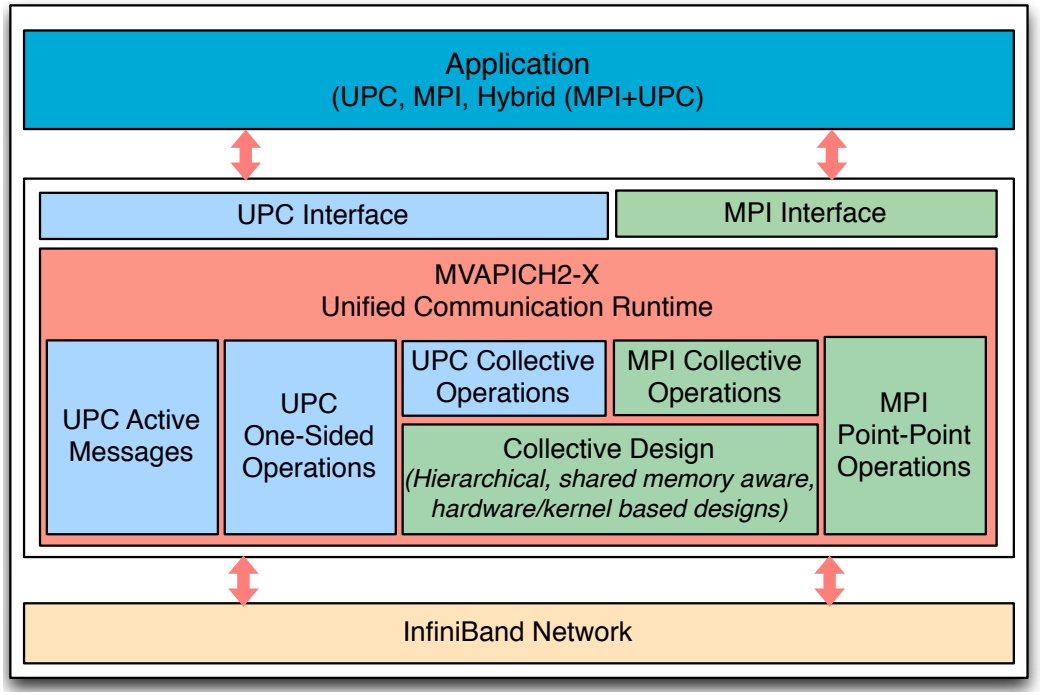
UPC Collectives: Scope of this study

- `upc_all_broadcast`
  - Copy a block of data from one process to all other processes (equivalent to MPI_Bcast)
- `upc_all_scatter`
  - Scatters the source buffer into all other processes, as indicated by offset (equivalent to MPI_Scatter)
- `upc_all_gather, upc_all_gather_all`
  - Inverse of scatter; gathers data from other processes into one/all processes (equivalent to MPI_Gather, MPI_Allgather)
- `upc_all_exchange`
  - All processes exchange data with every other process (similar to MPI_Alltoall)

MVAPICH2/MVAPICH2-X Software

- High Performance open-source MPI Library for InfiniBand, 10Gig/iWARP, and RDMA over Converged Enhanced Ethernet (RoCE)
  - MVAPICH (MPI-1), MVAPICH2 (MPI-2.2 and MPI-3.0), Available since 2002
  - MVAPICH2-X (MPI + PGAS), Available since 2012
  - Support for GPGPUs and MIC
  - **Used by more than 2,150 organizations (HPC Centers, Industry and Universities) in 72 countries**
  - **More than 212,000 downloads from OSU site directly**
  - Empowering many TOP500 clusters
    - 7th ranked 519,640-core cluster (Stampede) at TACC
    - 11th ranked 74,358-core cluster (Tsubame 2.5) at Tokyo Institute of Technology
    - 16th ranked 96,192-core cluster (Pleiades) at NASA and many others
  - Available with software stacks of many IB, HSE, and server vendors including Linux Distros (RedHat and SuSE)
  - http://mvapich.cse.ohio-state.edu
- Partner in the U.S. NSF-TACC Stampede System

Design Overview



- MPI Collectives in MVAPICH2/MVAPICH2-X highly optimized
  - Hierarchical, shared-memory aware, topology-aware, and hardware/kernel based designs
- Unified Communication Runtime enables UPC collectives to make use of advanced MPI collectives designs

UPC Collectives in MVAPICH2-X

- UPC Broadcast, Scatter, Gather
  - Two-level hierarchical algorithms
  - Intra and inter node levels
  - Typically use k-nomial algorithms for inter node transfers ($O(\log_k N)$ time)
  - Hardware-assisted multicast designs (when available)
  - Shared Memory/LiMIC designs for intra node transfers
- UPC Allgather, All-Exchange
  - Communication Intensive
  - Recursive Doubling/Brucks Algorithms (O(logN) time)
  - Hierarchical Leader based Algorithms
- Designs tuned based on platform

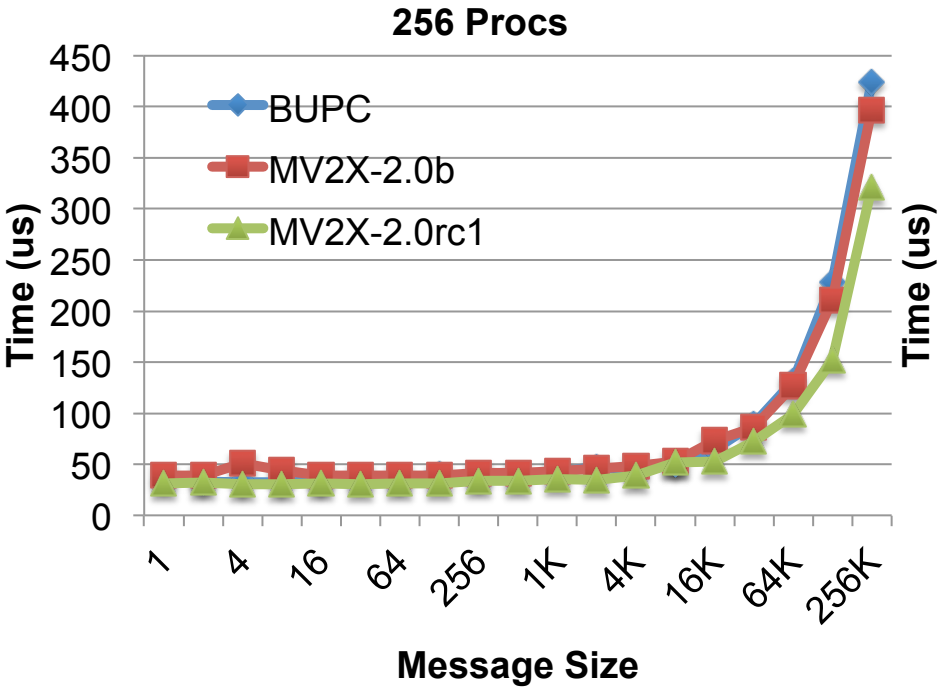
Outline

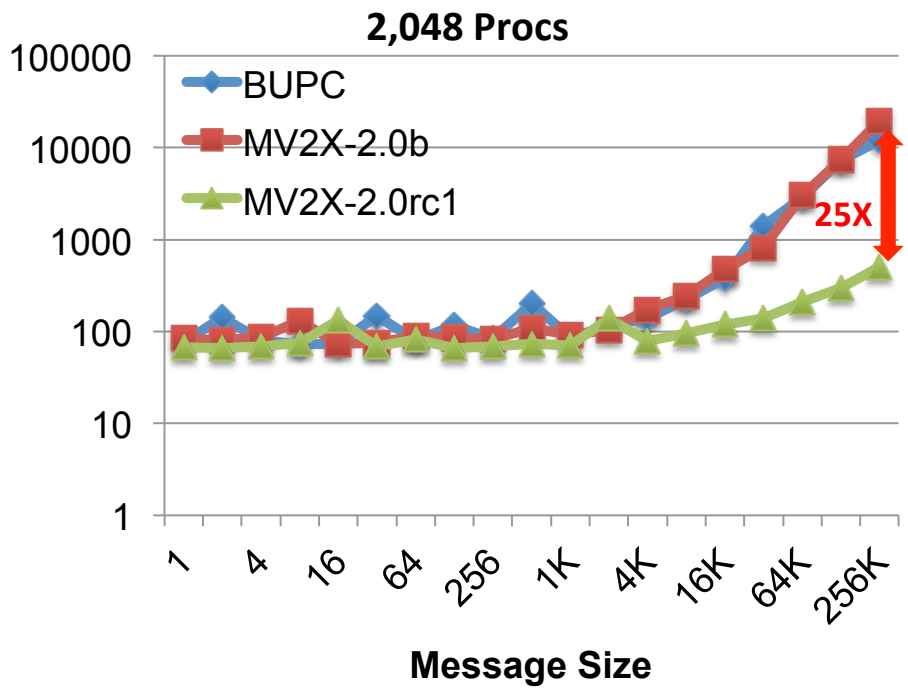
- Introduction & Motivation
- Problem Statement
- Design
- Performance Evaluation
  - Experiment Setup
  - Microbenchmark Results
  - Application Evaluations
- Conclusion & Future work

Experiment Setup

- Cluster A (TACC Stampede)
  - Intel Sandybridge series of processors using Xeon dual 8 core sockets (2.70GHz) with 32GB RAM
  - Each node is equipped with FDR ConnectX HCAs (54 Gbps data rate) with PCI-Ex Gen3 interfaces
- Cluster B (OSU Cluster)
  - Intel Westmere Dual quad-core processor (2.67GHz) with 12GB RAM
  - Each node is equipped with QDR ConnectX HCAs (32Gbps data rate) with PCI-Ex Gen2 interfaces
- Software Stacks
  - MVAPICH2-X UPC v2.0b (denoted as MV2X-2.0b)
  - Berkeley UPC v2.18.0 (denoted as BUPC)
  - MVAPICH2-X UPC + Proposed designs (denoted as MV2X-2.0rc1)
    - **Proposed designs are already available in MVAPICH2-X 2.0rc1**

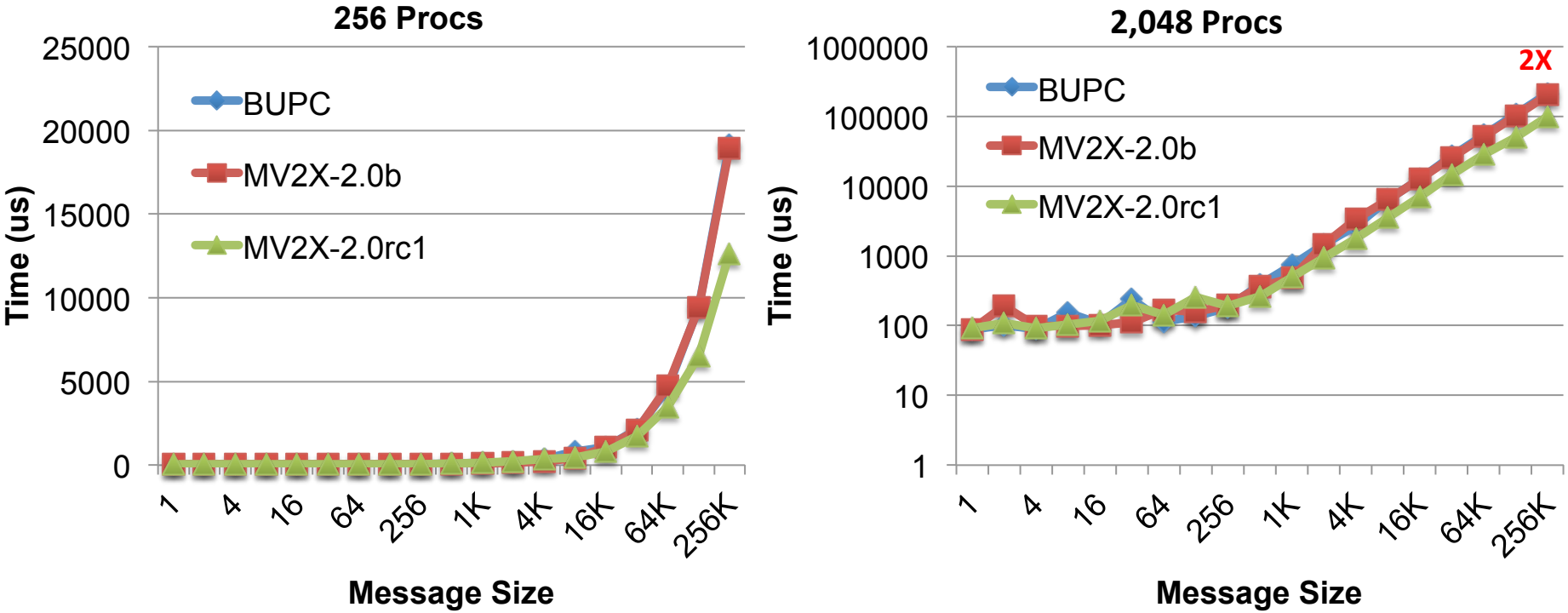
Broadcast Performance





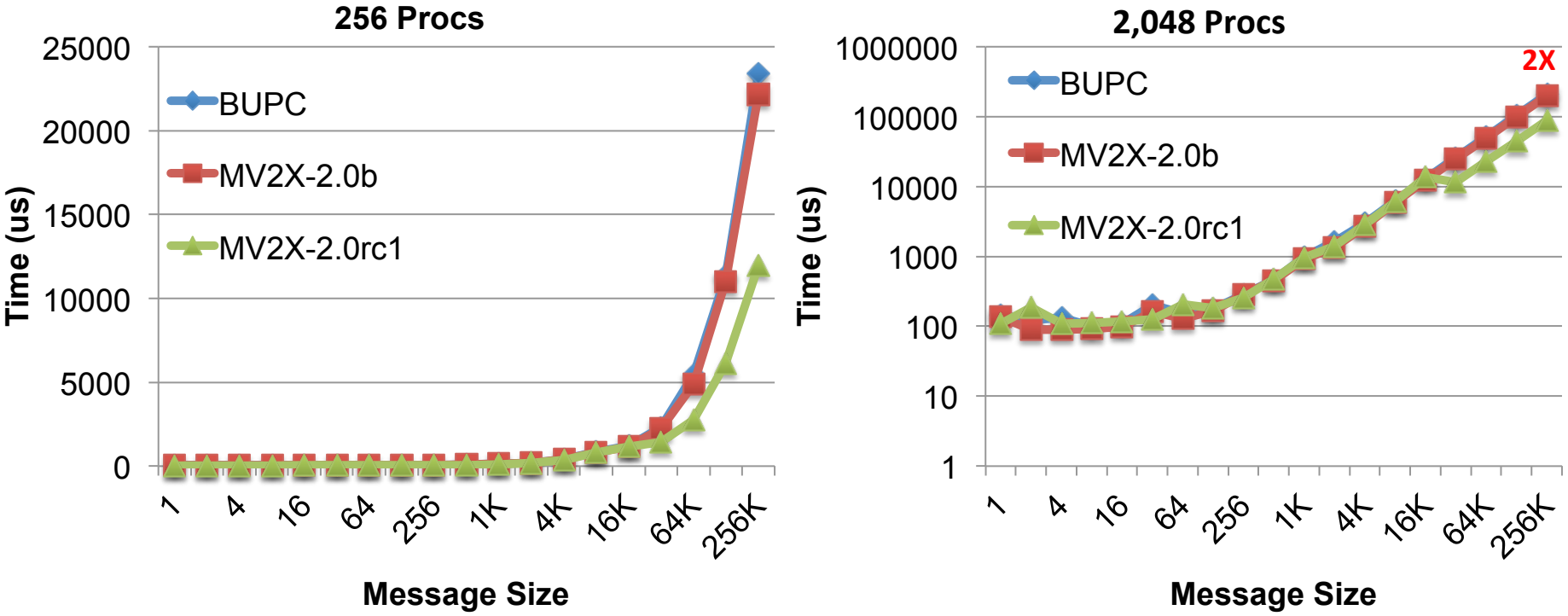
- BUPC and MV2-X 2.0b uses default collectives designs in GASNet
- At 2,048 processes, latency for 256 byte broadcast:
  - BUPC – 83us, MV2X-2.0b – 81us, MV2X-2.0rc1 - 68 us
- At 2,048 processes, latency for 256 Kbyte broadcast:
  - BUPC – 7297us, MV2X-2.0b – 7479us, MV2X-2.0rc1 - 299 us
  - 25X improvement compared to BUPC

Scatter Performance



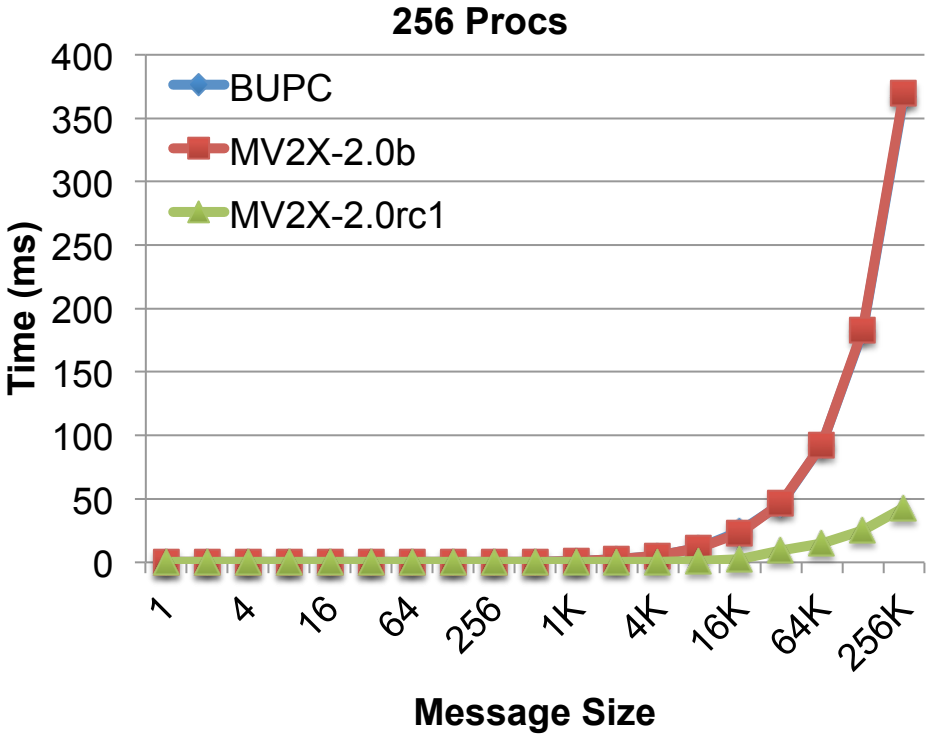
- At 2,048 processes, latency for 512 byte scatter:
  - BUPC – 388us, MV2X-2.0b – 362us, MV2X-2.0rc1 - 260 us
- At 2,048 processes, latency for 128 Kbyte scatter:
  - BUPC – 107ms, MV2X-2.0b – 102ms, MV2X-2.0rc1 - 50 ms
  - 2X improvement compared to BUPC

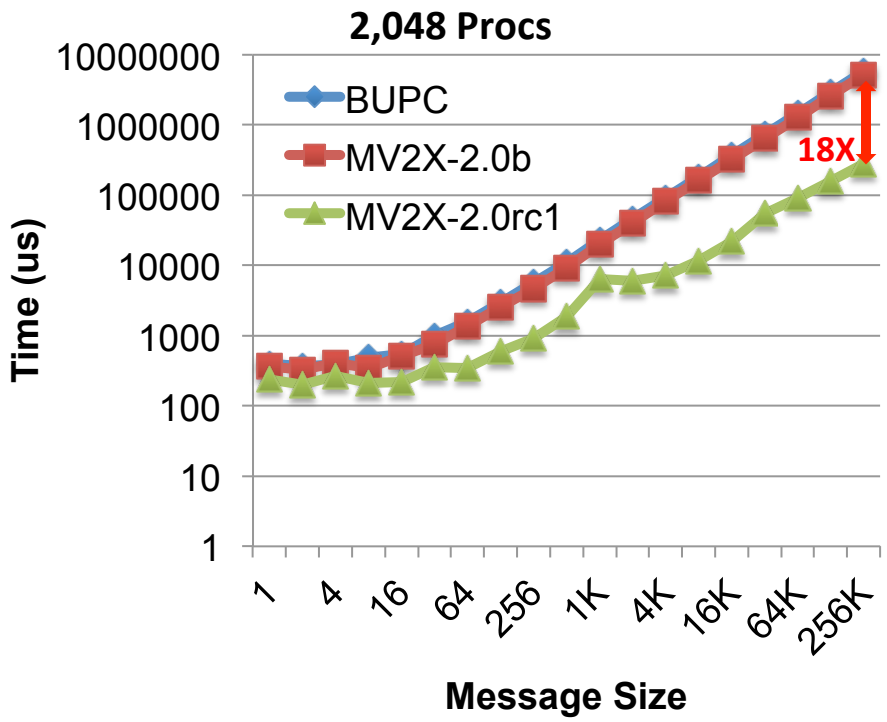
Gather Performance



- At 2,048 processes, latency for 256 byte gather:
  - BUPC – 286us, MV2X-2.0b – 286us, MV2X-2.0rc1 - 256 us
- At 2,048 processes, latency for 128 Kbyte gather:
  - BUPC – 104ms, MV2X-2.0b – 98ms, MV2X-2.0rc1 - 44 ms
  - 2X improvement compared to BUPC

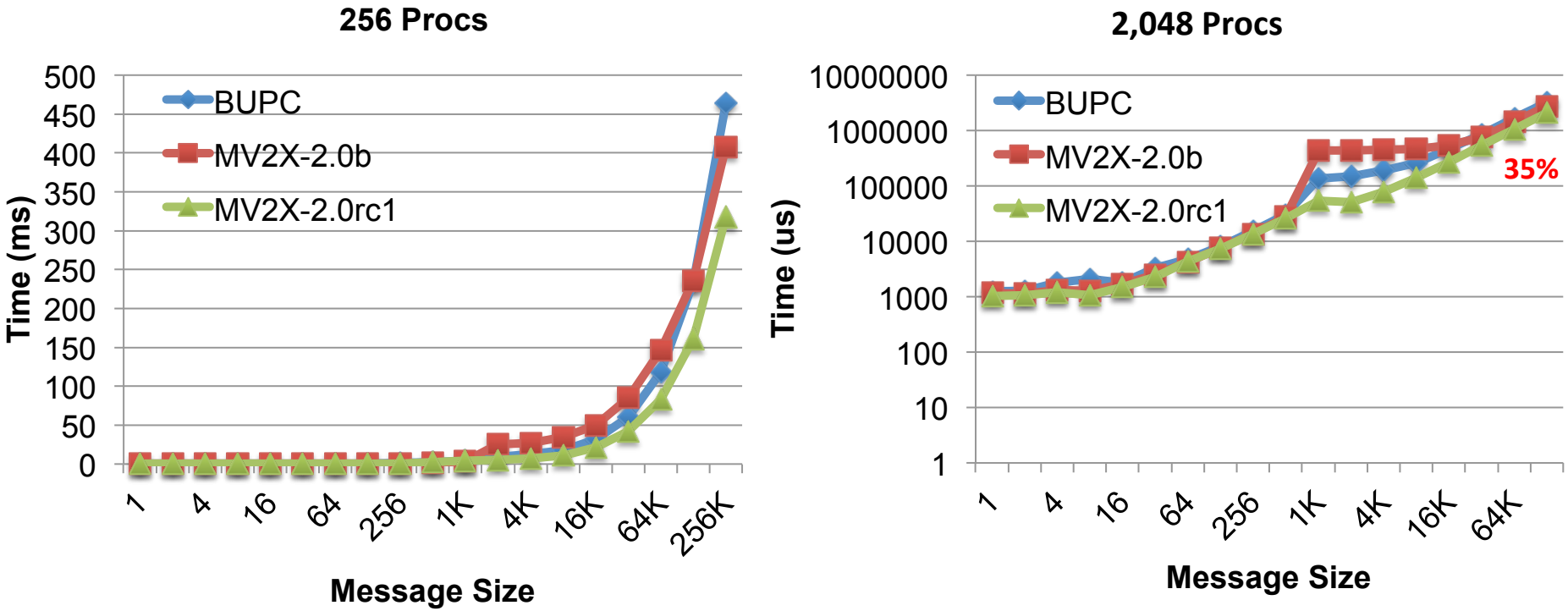
AllGather Performance





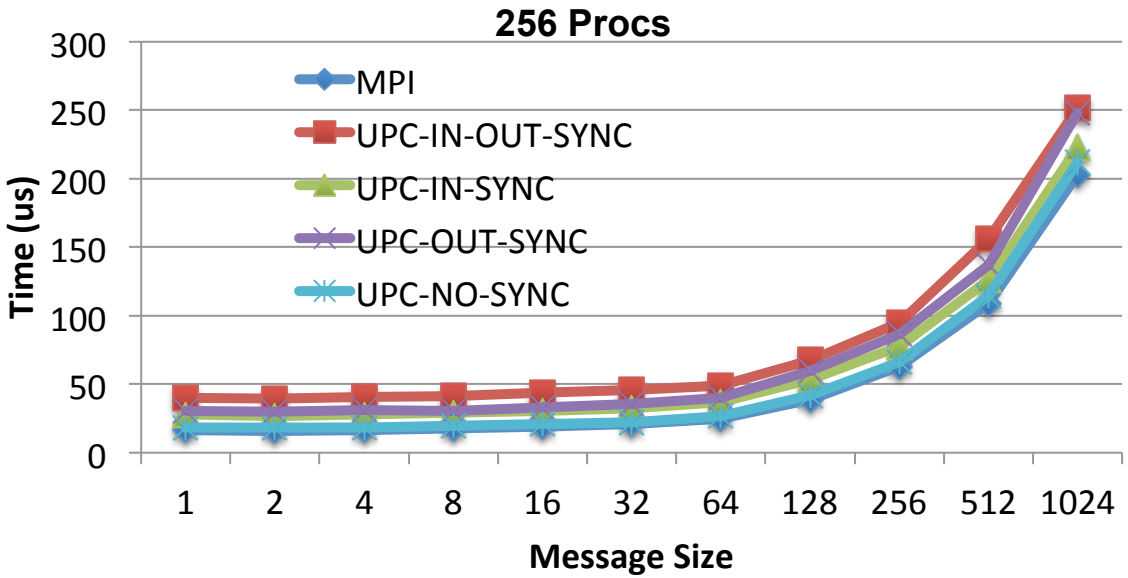
- At 2,048 processes, latency for 256 byte all-gather:
  - BUPC – 5.6ms, MV2X-2.0b – 4.6ms, MV2X-2.0rc1 - .9 ms
- At 2,048 processes, latency for 128 Kbyte all-gather:
  - BUPC – 2936ms, MV2X-2.0b – 2570ms, MV2X-2.0rc1 - 158 ms
  - 18X improvement compared to BUPC

Exchange Performance



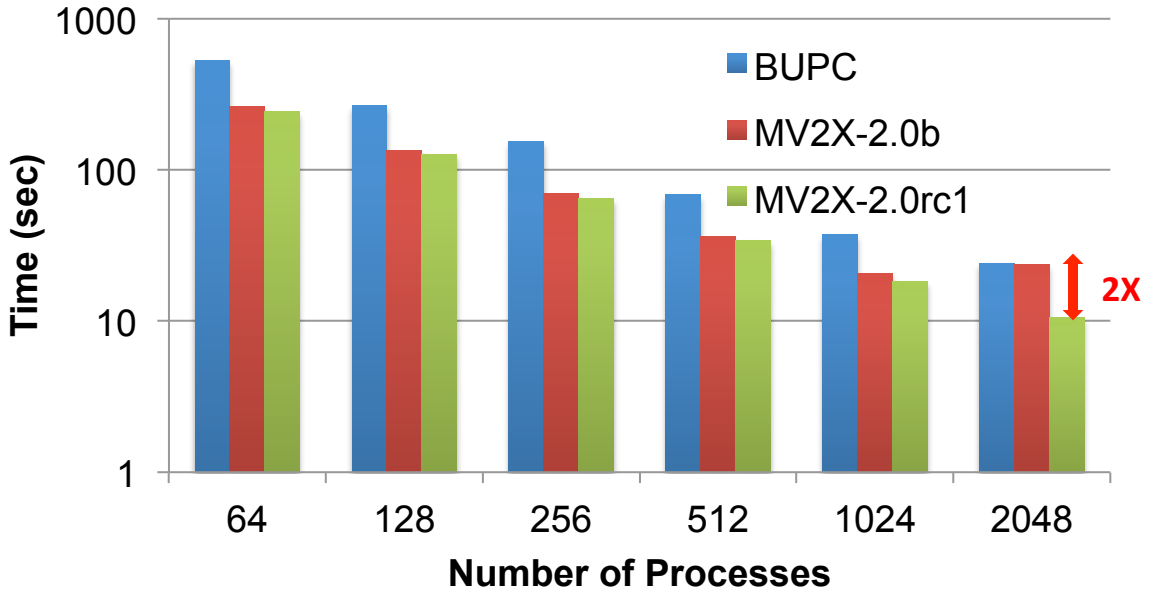
- At 2,048 processes, latency for 256 byte all-exchange:
  - BUPC – 8.1ms, MV2X-2.0b – 7.5ms, MV2X-2.0rc1 – 7.3 ms
- At 2,048 processes, latency for 128 Kbyte all-exchange:
  - BUPC – 3246ms, MV2X-2.0b – 2727ms, MV2X-2.0rc1 - 2100 ms
  - 35% improvement compared to BUPC

Performance Comparison with MPI Collectives



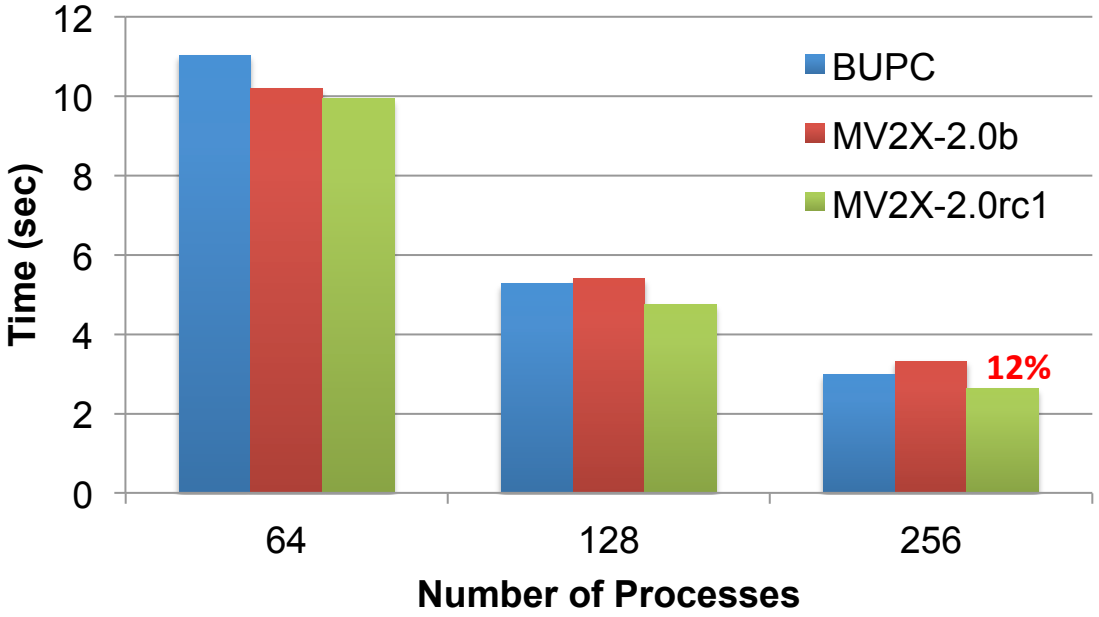
- Comparison with MPI_Allgather and UPC-Allgatherall
- IN/OUT sync offers higher latencies
- Similar latencies observed for MPI-Allgather and UPC-Allgather for NO-SYNC
- Indicates no overhead for the proposed design

Application Evaluation – 2D Heat Transfer



- 2D Heat Transfer Application (8K x 8K)
  - Repeats Jacobi kernel until the value convergence
  - Uses upc_barrier for synchronization, and upc_all_reduce and upc_all_broadcast in Jocobi kernel and for calculating convergence
  - Execution time (2,048 procs): BUPC – 24.13s, MV2X-2.0b – 23.74s, MV2X-2.0rc1 – 10.5 s
  - 2X Improvement compared to BUPC

Application Evaluation – NAS FT Benchmarks



- UPC NAS Benchmarks
  - FT Kernel (Class C)
  - Uses upc_all_exchange for the all-to-all exchange
  - Kernel repeated until the convergence value is reached
  - Execution time (512 procs): Linear – 2.99s, Tree – 3.4s, MV2X-2.0rc1 – 2.6s
  - 12% improvement compared to BUPC

Outline

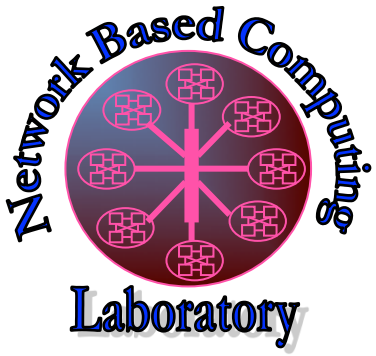
- Introduction & Motivation
- Problem Statement
- Design
- Performance Evaluation
- Conclusion & Future work

Conclusion

- Proposed a high performance, light weight design to map UPC collectives over MPI
- UPC implementations can directly leverage the advanced designs that are available in MPI
- Orders of magnitude improvement in performance in microbenchmark evaluations
- 2X improvement for 2D-Heat Transfer Modeling Application at 2,048 processes
- 12% improvement for UPC NAS benchmark at 512 processes
- Proposed designs and UPC Collective Benchmarks available in MVAPICH2-X 2.0rc1 release
- Future Work: Extend the designs to support other UPC collectives

Thank You!

{jose, hamidouche, zhanjie, akshay, panda}
@cse.ohio-state.edu



Network-Based Computing Laboratory

http://nowlab.cse.ohio-state.edu/

MVAPICH Web Page

http://mvapich.cse.ohio-state.edu/