MVAPICH
MPI, PGAS and Hybrid MPI+PGAS Library

*Follow us on*

https://twitter.com/mvapich



High-Performance
Big Data

2024 OFA Virtual Workshop

# OPTIMIZED ALL-TO-ALL CONNECTION ESTABLISHMENT FOR HIGH-PERFORMANCE MPI LIBRARIES OVER INFINIBAND

Shulei Xu, **Mustafa Abduljabbar**, Dhabaleswar K. (DK) Panda
Network Based Computing Laboratory
The Ohio State University
http://nowlab.cse.ohio-state.edu/

# Presentation Outline

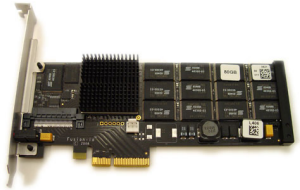
- Introduction & Background
- Motivation & Challenges
- Design & Optimization
- Performance Evaluation
- Conclusion

# What is Featured in Modern HPC Cluster Architectures?


Multi-core Processors

High Performance Interconnects - InfiniBand
<1usec latency, 400Gbps Bandwidth>

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



SSD, NVMe-SSD, NVRAM

- Multi-core/many-core technologies
- Remote Direct Memory Access (RDMA)-enabled networking (InfiniBand and RoCE)
- Solid State Drives (SSDs), Non-Volatile Random-Access Memory (NVRAM), NVMe-SSD
- Accelerators (NVIDIA GPGPUs and AMD GPUs)

***Summit***

***Sierra***

***Sunway TaihuLight***


***K - Computer***

# Overview of the MVAPICH 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 3,375 organizations in 91 countries**
- **More than 1.77 Million downloads from the OSU site directly**
- Empowering many TOP500 clusters (Nov '23 ranking)
  - 11th , 10,649,600-core (Sunway TaihuLight) at NSC, Wuxi, China
  - 29th , 448, 448 cores (Frontera) at TACC
  - 46th, 288,288 cores (Lassen) at LLNL
  - 61st, 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 29th ranked TACC Frontera system
- **Empowering Top500 systems for more than 18 years**

# Recent Trends

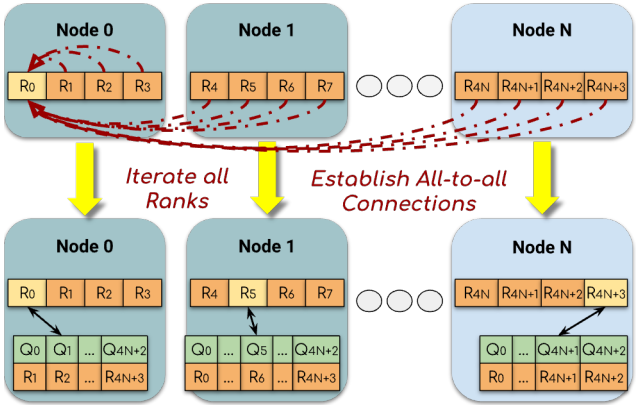
- **Number of cores/node is increasing**
  - 128 cores/node on the latest AMD EPYC systems
- **Total number of nodes for a given HPC system (certain number of cores) is decreasing**
- **Example: 8,192 core HPC systems**
  - 512 nodes with 16 cores/node (Past)
  - 64 nodes with 128 cores/node (Current)
- **Many challenges in designing MPI libraries for current and future systems**
- **Focus on this work: Minimizing Connection Establishment time in Job Startup**

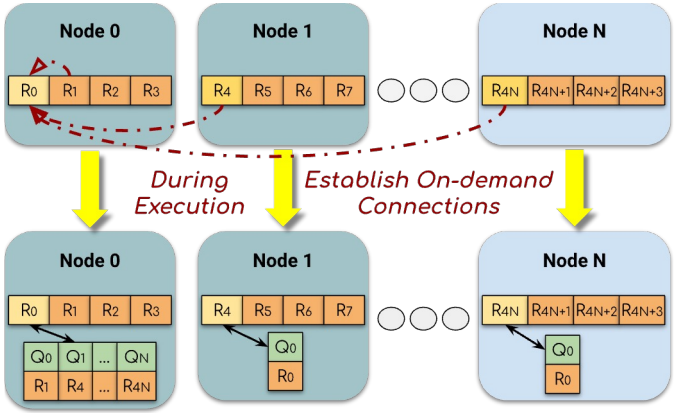
# The Process of Queue Pair (QP) Connection Establishment

- QPs are the building blocks of RDMA communication.
- A crucial step for sending/receiving data is done through QP connection establishment phase:
  - **Exchange QP Information:** Use an out-of-band method (e.g., sockets) to exchange QP identifiers and network parameters (LID, GID) between peers.
  - **Modify QP to RTR:** Update both QPs to the Ready to Receive state using the exchanged information. Essential for establishing a path for data reception.
  - **Modify QP to RTS:** Advance both QPs to the Ready to Send state, completing the connection setup. Enables full bi-directional communication.
  - **Verification:** Optionally perform a test operation (e.g., RDMA read or write) to verify the connection integrity and performance.

# Message Passing Interface (MPI) & SOTA Startup Methods

- One of the most popular programming model for HPC parallel applications
- Two major methods to establish QP connections during at initialization time
  - All-to-all connection establishment
  - On-demand connection establishment



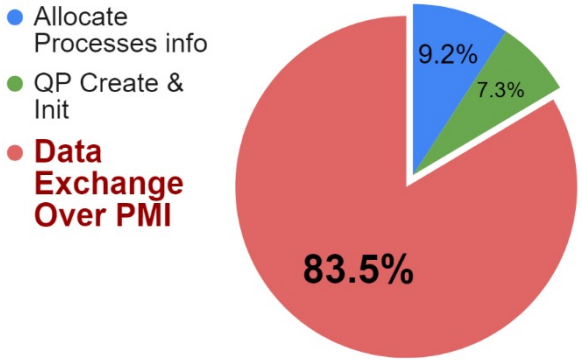
# Comparing All-to-all vs. On-demand Connection Establishment Models

- Two major methods to establish connections
  - All-to-all connection establishment
  - On-demand connection establishment

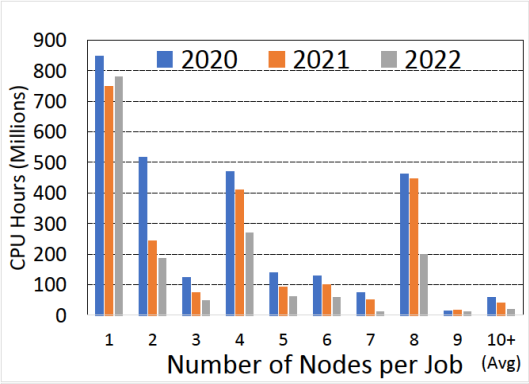
| | All-to-all | On-demand |
|---|---|---|
| **When to build connection** | Make all connections during MPI_Init | Make connections on-demand during communication |
| **Total number of queue pairs** | NP * (NP – 1) | Decided by comm. pattern, always <= NP * (NP – 1) |
| **MPI_Init duration** | Growing along with NP increasing | Always short |
| **Comm latency of the initial iterations** | Consistent with later iterations | Much slower than later iterations |
| **Suitable Comm. Patterns** | Complex patterns (e.g. alltoall) | Simple patterns (e.g. bcast) |

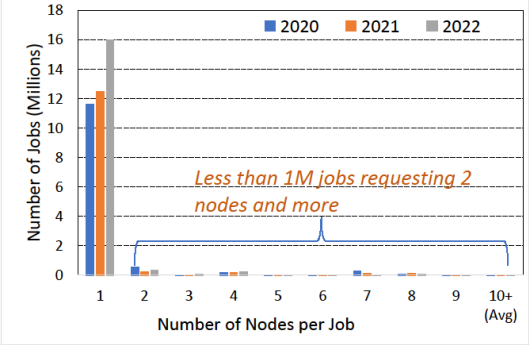
# Motivation

- Majority of the HPC jobs are in 8 nodes or smaller scale
- The startup performance can benefit a lot from optimizations to All-to-all connection establishment
- Large amount of time is consumed in data exchange step of MPI job initialization




Total time percentage of different operations during all-to-all RDMA connection startup, run with up to 16 nodes 56 PPN (Process Per Node) on TACC Frontera




Usage trends of Various HPC systems observed by NSF funded XSEDE project, https://www.xsede.org/

# Challenges

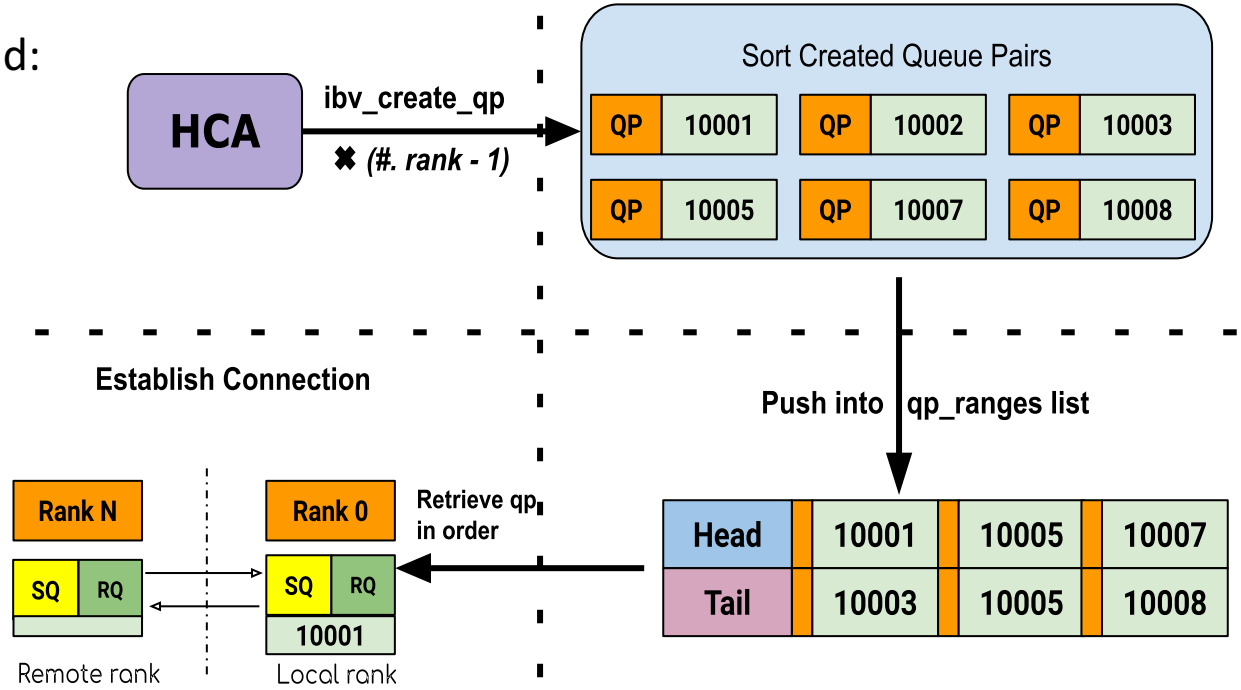
- **How to minimize the data, especially the queue pair numbers exchanged via PMI for optimizing all-to-all connection establishment?**
  - *Rank-by-rank all-to-all connection establishment design*
- **How to utilize the system characteristics to balance different types of overhead in the all-to-all connection establishment process?**
  - Parallelized queue pair creation & initialization
  - Enhanced Affinity-aware all-to-all startup
- **How to select the best connection establishment approach to benefit the overall HPC application performance?**
  - Comprehensive performance analysis to identify best use cases

# Presentation Outline

- Introduction & Background
- Motivation & Challenges
- Design & Optimization
- Performance Evaluation
- Conclusion

# Design – Rank-by-rank All-to-all Connection Establishment

- **Observation 1:** Vast majority of created QPs are returned by adapter with ID number in order (very small chance of few QPs out-of-order)
- Design based on this trend:
  - Step 1: Create QP
  - Step 2: Sort QP
  - Step 3: Enqueue Sorted QPs
  - Step 4: Retrieve QP in order

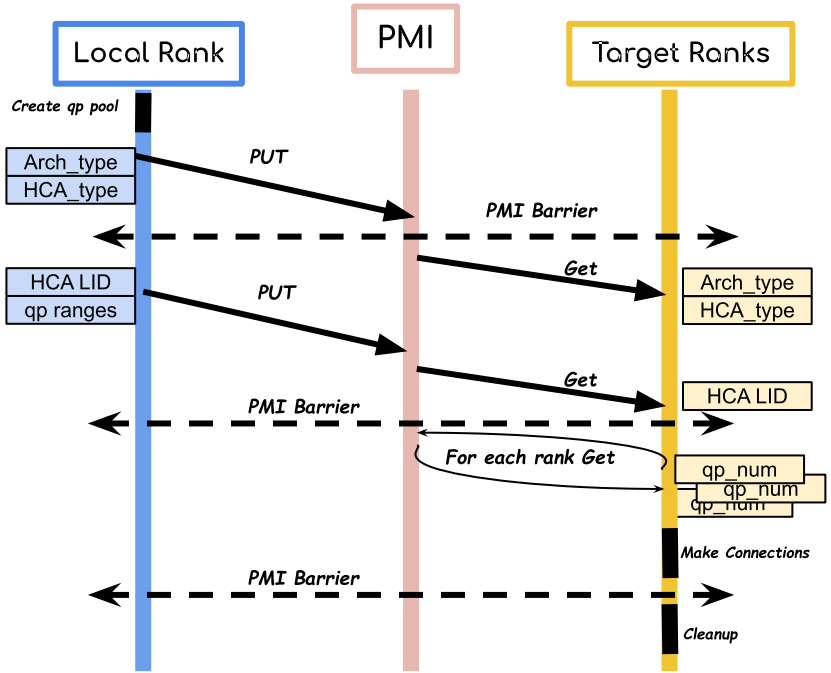


# Example of Information Exchanged across processes via Process Management Interface (PMI)

**Consists of two primary steps**

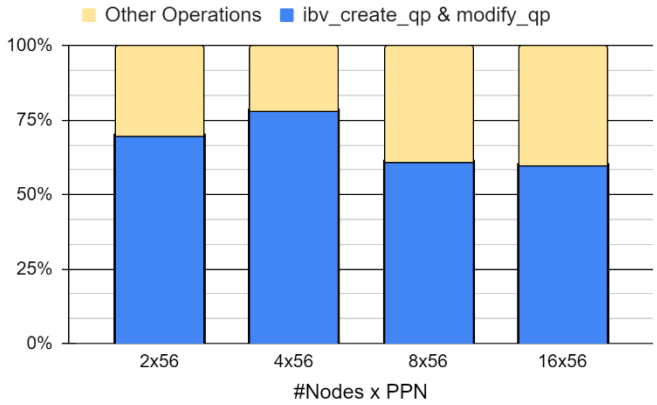
- Exchange hardware information (e.g. architecture, HCA/network adapter)

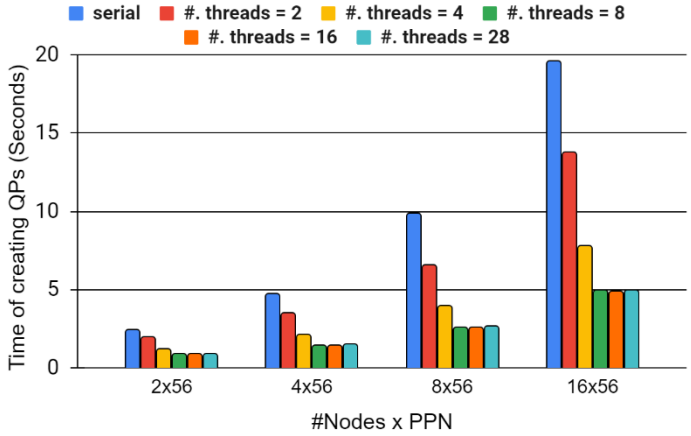
- Exchange HCA local ID and created QPs' IDs



# Design – Parallelized QP Creation & Initialization

- **Observation 2:** QP creation & initialization is the major overhead in MPI_Init, and can be parallelized.
- Leverage OpenMP to parallelize QP creation & initialization with multiple threads.
- In this example, we choose 8 threads following Occam Razor's rule.
- To avoid performance degradation from CPU pinning, MPI affinity is configured after inter-node startup step.



# Presentation Outline

- Introduction & Background
- Motivation & Challenges
- Design & Optimization
- Performance Evaluation
- Conclusion

# Performance Evaluation – Experimental Setups

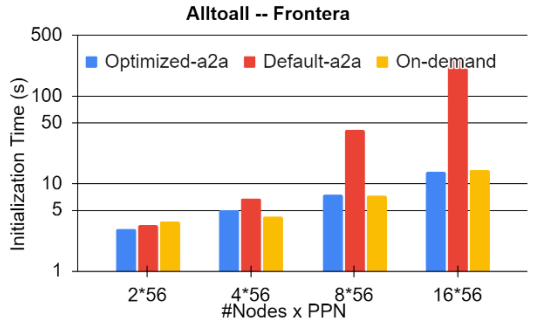
- MVAPICH2-2.3.7 MPI library
- OSU-Microbenchmark (OMB) v5.9
- Applications including NPB-3.4.2 & 3D-stencil
- Experiments conducted on two different modern systems:

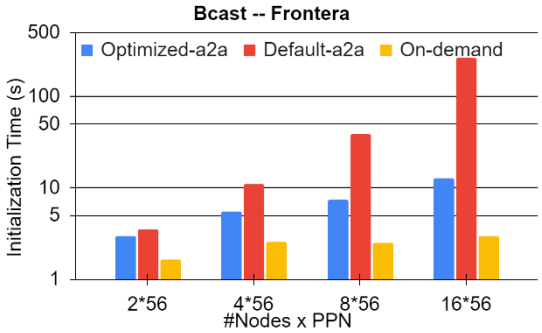
| Specification | TACC Frontera | TACC Lonestar6 |
|---|---|---|
| Processor Family | Intel Cascade Lake | AMD EPYC |
| Processor Model | Xeon Platinum 8280 | EPYC 7763 |
| Sockets | 2 | 2 |
| Cores Per Socket | 28 | 64 |
| RAM (DDR4) | 192 GB | 256 GB |
| Interconnect | InfiniBand HDR (100G) | InfiniBand HDR (200G) |

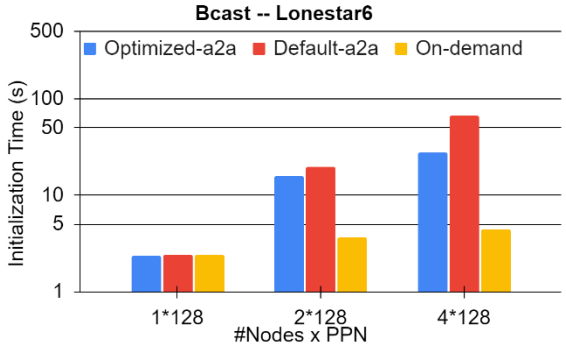
# Micro-benchmark Level Results

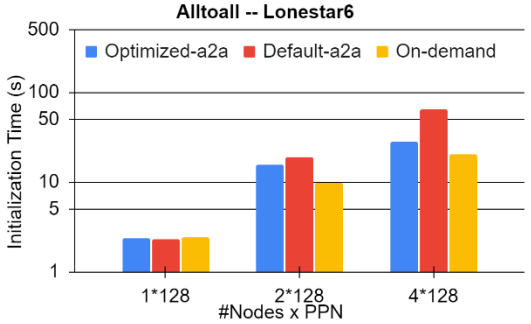
**Total time consumed by different startup approaches**

- The proposed all-to-all startup design is up to 20x faster than existing all-to-all approaches



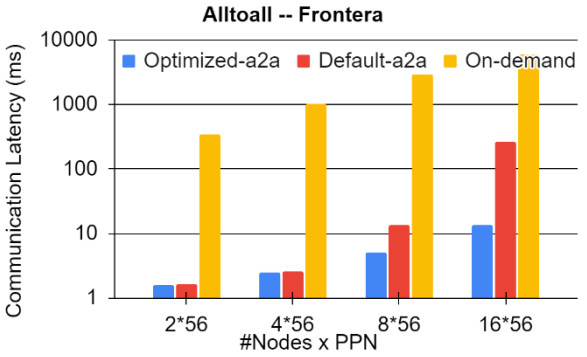



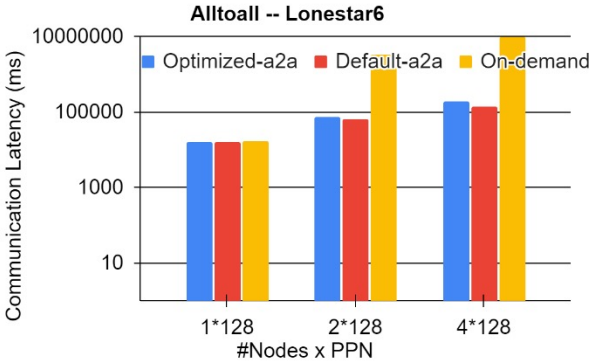



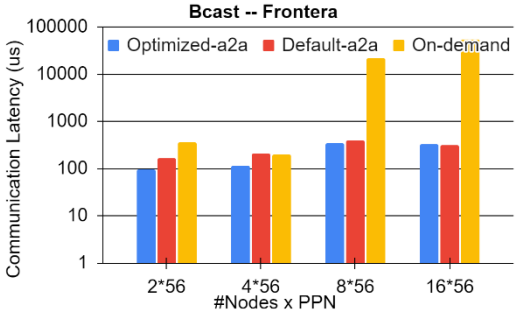
# Micro-benchmark Level Results

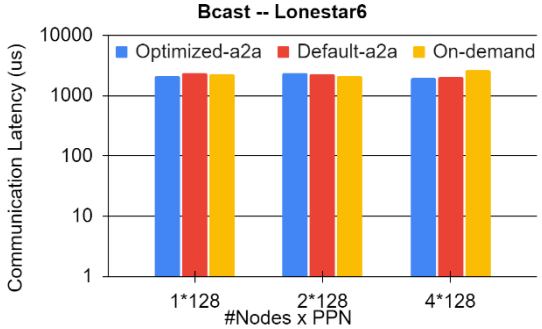
**First iteration communication latency after different approaches of connection establishment**

- The proposed all-to-all startup design is up to 448x faster than existing all-to-all approaches in first iteration of communication



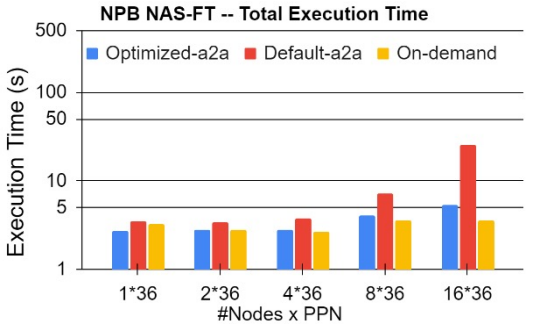



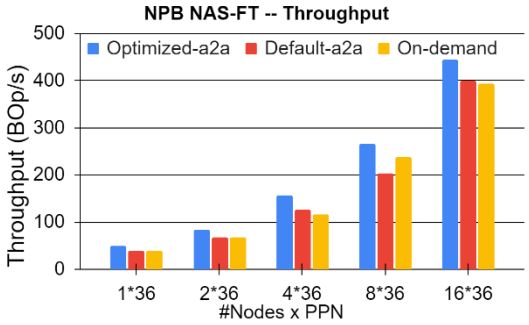



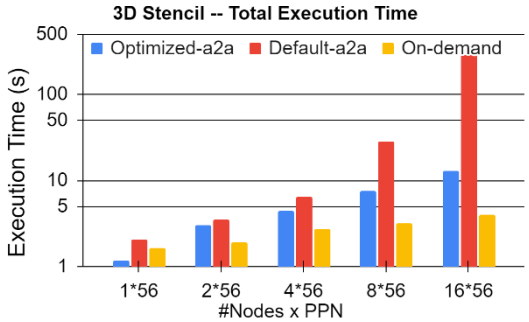



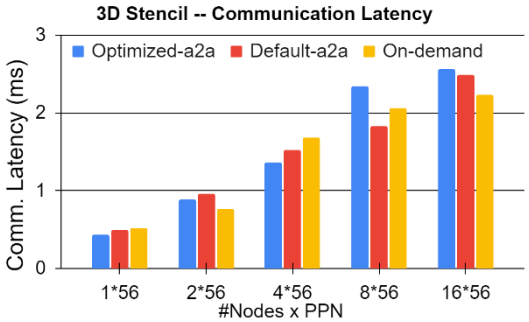
# Application Results

- 3D stencil:
  - Up to 10x faster than existing all-to-all startup design
- NPB NAS-FT (Class B):
  - Up to 31% higher throughput comparing to existing on-demand startup approach



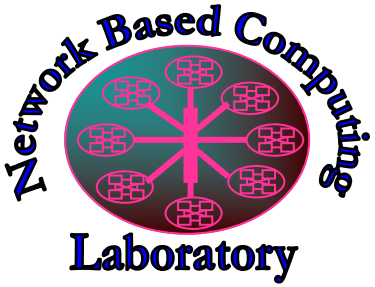



# Conclusion

- The connection establishment methodology is becoming a crucial factor in the overall performance of the job
- We investigate the primary bottleneck of existing SOTA all-to-all connection establishment designs and propose an optimized rank-by-rank connection establishment design
- We implement the design into MVAPICH2 MPI library, and observe up to **20x** faster MPI_Init latency in micro-benchmark level and **31%** higher throughput with NPB NAS-FT application
- The new designs will be made available with upcoming MVAPICH releases
- We plan to extend the proposed design to additional architectures (e.g. ARM) or network adapters (e.g. Broadcom, PSM)

# Q&A

# 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/