Advancing MPI Libraries to the Many-core Era: Designs and Evaluations with MVAPICH2

IXPUG ’17 Presentation

S. Chakraborty, M. Bayatpour, H. Subramoni and DK Panda
The Ohio State University
E-mail: {Chakraborty.52,Bayatpour.1,Subramoni.1,panda.2}@osu.edu
Parallel Programming Models Overview

- 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.
- PGAS models and Hybrid MPI+PGAS models are gradually receiving importance
Supporting Programming Models for Multi-Petaflop and Exaflop Systems: Challenges

Application Kernels/Applications

Middleware

Programming Models
MPI, PGAS (UPC, Global Arrays, OpenSHMEM), CUDA, OpenMP, OpenACC, Cilk, Hadoop (MapReduce), Spark (RDD, DAG), etc.

Communication Library or Runtime for Programming Models
- Point-to-point Communication
- Collective Communication
- Energy-Awareness
- Synchronization and Locks
- I/O and File Systems
- Fault Tolerance

Networking Technologies
(InfiniBand, 40/100GigE, Aries, and Omni-Path)

Multi-/Many-core Architectures

Accelerators (GPU and FPGA)

Co-Design Opportunities and Challenges across Various Layers
Performance
Scalability
Resilience

Network Based Computing Laboratory
Designing (MPI+X) for Exascale

- Scalability for million to billion processors
  - Support for highly-efficient inter-node and intra-node communication (both two-sided and one-sided)

- Scalable Collective communication
  - Offloaded
  - Non-blocking
  - Topology-aware

- Balancing intra-node and inter-node communication for next generation multi-/many-core (128-1024 cores/node)
  - Multiple end-points per node

- Support for efficient multi-threading

- Integrated Support for GPGPUs and FPGAs

- Fault-tolerance/resiliency

- QoS support for communication and I/O

- Support for Hybrid MPI+PGAS programming
  - MPI + OpenMP, MPI + UPC, MPI + OpenSHMEM, CAF, MPI + UPC++...

- Virtualization

- Energy-Awareness
Overview of the MVAPICH2 Project

- High Performance open-source MPI Library for InfiniBand, Omni-Path, Ethernet/iWARP, and RDMA over Converged Ethernet (RoCE)
  - MVAPICH (MPI-1), MVAPICH2 (MPI-2.2 and MPI-3.0), Started in 2001, First version available in 2002
  - MVAPICH2-X (MPI + PGAS), Available since 2011
  - Support for GPGPUs (MVAPICH2-GDR) and MIC (MVAPICH2-MIC), Available since 2014
  - Support for Virtualization (MVAPICH2-Virt), Available since 2015
  - Support for Energy-Awareness (MVAPICH2-EA), Available since 2015
  - Support for InfiniBand Network Analysis and Monitoring (OSU INAM) since 2015
  - Used by more than 2,825 organizations in 85 countries
  - More than 427,000 (> 0.4 million) downloads from the OSU site directly
  - Empowering many TOP500 clusters (June ‘17 ranking)
    - 1st, 10,649,600-core (Sunway TaihuLight) at National Supercomputing Center in Wuxi, China
    - 15th, 241,108-core (Pleiades) at NASA
    - 20th, 462,462-core (Stampede) at TACC
    - 44th, 74,520-core (Tsubame 2.5) at Tokyo Institute of Technology
  - Available with software stacks of many vendors and Linux Distros (RedHat and SuSE)
- http://mvapich.cse.ohio-state.edu
  - Empowering Top500 systems for over a decade
    - System-X from Virginia Tech (3rd in Nov 2003, 2,200 processors, 12.25 TFlops) ->
    - Sunway TaihuLight (1st in Jun’17, 10M cores, 100 PFlops)
MVAPICH Project Timeline

Timeline

MVAPICH

EOL

OMB

MVAPICH2

MVAPICH2-Virt

MVAPICH2-MIC

MVAPICH2-GDR

MVAPICH2-X

MVAPICH2-EA

OSU-INAM

Network Based Computing Laboratory

IXPUG ‘17
MVAPICH2 Release Timeline and Downloads

Number of Downloads

Timeline

MV 0.9.4
MV2 0.9.0
MV2 0.9.8
MV2 1.0
MV2 1.0.3
MV 1.1
MV2 1.4
MV 1.5
MV2 1.6
MV2 1.7
MV2 1.8
MV2 1.9
MV2-GDR 2.0b
MV2-MIC 2.0
MV2-GDR 2.1
MV2-GDR 2.2rc1
MV2-X 2.2
MV2-Virt 2.2
MV2 2.3b
MV2-GDR 2.2rc1
MV2-Virt 2.2
MV2 2.3b
### Architecture of MVAPICH2 Software Family

#### High Performance Parallel Programming Models
- **Message Passing Interface (MPI)**
- **PGAS** (UPC, OpenSHMEM, CAF, UPC++)
- **Hybrid --- MPI + X** (MPI + PGAS + OpenMP/Cilk)

#### High Performance and Scalable Communication Runtime

#### Diverse APIs and Mechanisms

- **Point-to-point Primitives**
- **Collectives Algorithms**
- **Job Startup**
- **Energy-Awareness**
- **Remote Memory Access**
- **I/O and File Systems**
- **Fault Tolerance**
- **Virtualization**
- **Active Messages**
- **Introspection & Analysis**

#### Support for Modern Networking Technology
*(InfiniBand, iWARP, RoCE, Omni-Path)*

<table>
<thead>
<tr>
<th>Transport Protocols</th>
<th>Modern Features</th>
<th>Transport Mechanisms</th>
</tr>
</thead>
<tbody>
<tr>
<td>RC</td>
<td>UMR</td>
<td>Shared Memory</td>
</tr>
<tr>
<td>XRC</td>
<td>ODP</td>
<td>CMA</td>
</tr>
<tr>
<td>UD</td>
<td>SR-IOV</td>
<td>IVSHMEM</td>
</tr>
<tr>
<td>DC</td>
<td>Multi Rail</td>
<td></td>
</tr>
</tbody>
</table>

#### Support for Modern Multi-/Many-core Architectures
*(Intel-Xeon, OpenPower, Xeon-Phi (MIC, KNL), NVIDIA GPGPU)*

<table>
<thead>
<tr>
<th>Modern Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCDRAM*</td>
</tr>
<tr>
<td>NVLink*</td>
</tr>
<tr>
<td>CAPI*</td>
</tr>
</tbody>
</table>

*Upcoming*
# MVAPICH2 Software Family

## High-Performance Parallel Programming Libraries

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MVAPICH2</td>
<td>Support for InfiniBand, Omni-Path, Ethernet/iWARP, and RoCE</td>
</tr>
<tr>
<td>MVAPICH2-X</td>
<td>Advanced MPI features, OSU INAM, PGAS (OpenSHMEM, UPC, UPC++, and CAF), and MPI +PGAS programming models with unified communication runtime</td>
</tr>
<tr>
<td>MVAPICH2-GDR</td>
<td>Optimized MPI for clusters with NVIDIA GPUs</td>
</tr>
<tr>
<td>MVAPICH2-Virt</td>
<td>High-performance and scalable MPI for hypervisor and container based HPC cloud</td>
</tr>
<tr>
<td>MVAPICH2-EA</td>
<td>Energy aware and High-performance MPI</td>
</tr>
<tr>
<td>MVAPICH2-MIC</td>
<td>Optimized MPI for clusters with Intel KNC</td>
</tr>
</tbody>
</table>

## Microbenchmarks

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OMB</td>
<td>Microbenchmarks suite to evaluate MPI and PGAS (OpenSHMEM, UPC, and UPC++) libraries for CPUs and GPUs</td>
</tr>
</tbody>
</table>

## Tools

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSU INAM</td>
<td>Network monitoring, profiling, and analysis for clusters with MPI and scheduler integration</td>
</tr>
<tr>
<td>OEMT</td>
<td>Utility to measure the energy consumption of MPI applications</td>
</tr>
</tbody>
</table>
MVAPICH2 2.3b

- Released on 08/10/2017

- Major Features and Enhancements
  - Based on MPICH-3.2
  - Enhance performance of point-to-point operations for CH3-Gen2 (InfiniBand), CH3-PSM, and CH3-PSM2 (Omni-Path) channels
  - Improve performance for MPI-3 RMA operations
  - Introduce support for Cavium ARM (ThunderX) systems
  - Improve support for process to core mapping on many-core systems
    - New environment variable MV2_THREADS_BINDING_POLICY for multi-threaded MPI and MPI+OpenMP applications
    - Support ‘linear’ and ‘compact’ placement of threads
    - Warn user if over-subscription of core is detected
  - Improve launch time for large-scale jobs with mpirun_rsh
  - Add support for non-blocking Allreduce using Mellanox SHARP
  - Efficient support for different Intel Knight’s Landing (KNL) models
  - Improve performance for Intra- and Inter-node communication for OpenPOWER architecture
  - Improve support for large processes per node and huge pages on SMP systems
  - Enhance collective tuning for many architectures/systems
  - Enhance support for MPI_T PVARs and CVARs
Overview of A Few Challenges being Addressed by the MVAPICH2 Project for Many-core Era

- Fast and Scalable Job Start-up
- Dynamic and Adaptive Communication Protocols and Tag Matching
- Contention-aware Designs for Intra-node Collectives
- Scalable Multi-leader Designs for Collectives
- Kernel-Assisted Communication Designs for KNL
- Efficient RMA-based Designs for Graph500 on KNL
Towards High Performance and Scalable Startup at Exascale

- Near-constant MPI and OpenSHMEM initialization time at any process count
- 10x and 30x improvement in startup time of MPI and OpenSHMEM respectively at 16,384 processes
- Memory consumption reduced for remote endpoint information by $O(\text{processes per node})$
- 1GB Memory saved per node with 1M processes and 16 processes per node

Symbols:
- PGAS – State of the art
- MPI – State of the art
- PGAS/MPI – Optimized
- On-demand Connection
- PMIX_Ring
- PMIX_Ibarrier
- PMIX_Iallgather
- Shmem based PMI

**On-demand Connection Management for OpenSHMEM and OpenSHMEM+MPI.** S. Chakraborty, H. Subramoni, J. Perkins, A. A. Awan, and D K Panda, 20th International Workshop on High-level Parallel Programming Models and Supportive Environments (HIPS ’15)

**PMI Extensions for Scalable MPI Startup.** S. Chakraborty, H. Subramoni, A. Moody, J. Perkins, M. Arnold, and D K Panda, Proceedings of the 21st European MPI Users' Group Meeting (EuroMPI/Asia ’14)


Process Management Interface (PMI) over Shared Memory (SHMEMPMI)

- SHMEMPMI allows MPI processes to directly read remote endpoint (EP) information from the process manager through shared memory segments.
- Only a single copy per node - $O(\text{processes per node})$ reduction in memory usage.
- Estimated savings of 1GB per node with 1 million processes and 16 processes per node.
- Up to 1,000 times faster PMI Gets compared to default design.
- Available since MVAPICH2 2.2rc1.

**Time Taken by one PMI_Get**

![Graph showing time taken by one PMI_Get for different numbers of processes per node.](image)

- **Default**
- **SHMEMPMI**

**Memory Usage for Remote EP Information**

![Graph showing memory usage for remote EP information with different numbers of processes per job.](image)

- Fence - Default
- Allgather - Default
- Fence - Shmem
- Allgather - Shmem

- Estimated savings of 1000x
- Actual savings of 16x
Start-up Performance on KNL + Omni-Path

- **MPI_Init** takes 51 seconds on 231,956 processes on 3,624 KNL nodes (Stampede2 – Full scale)
- 8.8 times faster than Intel MPI at 128K processes (Courtesy: TACC)
- At 64K processes, **MPI_Init** and **Hello World** takes 5.8s and 21s respectively (Oakforest-PACS)
- All numbers reported with 64 processes per node

New designs available in MVAPICH2-2.3a and as patch for SLURM-15.08.8 and SLURM-16.05.1
Overview of A Few Challenges being Addressed by the MVAPICH2 Project for Many-core Era

- Fast and Scalable Job Start-up
- Dynamic and Adaptive Communication Protocols and Tag Matching
- Contention-aware Designs for Intra-node Collectives
- Scalable Multi-leader Designs for Collectives
- Kernel-Assisted Communication Designs for KNL
- Efficient RMA-based Designs for Graph500 on KNL
Dynamic and Adaptive MPI Point-to-point Communication Protocols

Desired Eager Threshold

<table>
<thead>
<tr>
<th>Process Pair</th>
<th>Eager Threshold (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 – 4</td>
<td>32</td>
</tr>
<tr>
<td>1 – 5</td>
<td>64</td>
</tr>
<tr>
<td>2 – 6</td>
<td>128</td>
</tr>
<tr>
<td>3 – 7</td>
<td>32</td>
</tr>
</tbody>
</table>

Eager Threshold for Example Communication Pattern with Different Designs

- **Default**
  - Poor overlap; Low memory requirement
  - Low Performance; High Productivity

- **Manually Tuned**
  - Good overlap; High memory requirement
  - High Performance; Low Productivity

- **Dynamic + Adaptive**
  - Good overlap; Optimal memory requirement
  - High Performance; High Productivity

**Execution Time of Amber**

<table>
<thead>
<tr>
<th>Wall Clock Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

- Default
- Threshold=17K
- Threshold=64K
- Dynamic Threshold

<table>
<thead>
<tr>
<th>Number of Processes</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
</tr>
</tbody>
</table>

**Relative Memory Consumption of Amber**

<table>
<thead>
<tr>
<th>Relative Memory Consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

- Default
- Threshold=128K
- Threshold=64K
- Dynamic Threshold

Dynamic and Adaptive Tag Matching

Challenge
Tag matching is a significant overhead for receivers
Existing Solutions are
- Static and do not adapt dynamically to communication pattern
- Do not consider memory overhead

Solution
A new tag matching design
- Dynamically adapt to communication patterns
- Use different strategies for different ranks
- Decisions are based on the number of request object that must be traversed before hitting on the required one

Results
Better performance than other state-of-the-art tag-matching schemes
Minimum memory consumption
Will be available in future MVAPICH2 releases

Normalized Total Tag Matching Time at 512 Processes
Normalized to Default (Lower is Better)

Normalized Memory Overhead per Process at 512 Processes
Compared to Default (Lower is Better)

Overview of A Few Challenges being Addressed by the MVAPICH2 Project for Many-core Era

- Fast and Scalable Job Start-up
- Dynamic and Adaptive Communication Protocols and Tag Matching
- Contention-aware Designs for Intra-node Collectives
- Scalable Multi-leader Designs for Collectives
- Kernel-Assisted Communication Designs for KNL
- Efficient RMA-based Designs for Graph500 on KNL
### Different Kernel-Assisted Single Copy Copy Mechanisms

<table>
<thead>
<tr>
<th></th>
<th>CMA</th>
<th>KNEM</th>
<th>LiMiC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cookie/Region Creation</td>
<td>Not Required</td>
<td>Required</td>
<td>Required</td>
</tr>
<tr>
<td>Permission Check</td>
<td>Supported</td>
<td>Supported</td>
<td>Not Supported</td>
</tr>
<tr>
<td>Availability</td>
<td>Included in Linux 3.2+</td>
<td>Kernel Module</td>
<td>Kernel Module</td>
</tr>
</tbody>
</table>

### MPI Library Support

<table>
<thead>
<tr>
<th></th>
<th>CMA</th>
<th>KNEM</th>
<th>LiMiC</th>
</tr>
</thead>
<tbody>
<tr>
<td>MVAPICH2-2.3a</td>
<td>√</td>
<td>x</td>
<td>√</td>
</tr>
<tr>
<td>OpenMPI 2.1.0</td>
<td>√</td>
<td>√</td>
<td>x</td>
</tr>
<tr>
<td>Intel MPI 2017</td>
<td>√</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

CMA (Cross Memory Attach) is the most widely supported kernel-assisted transfer mechanism
Impact of Communication Pattern on CMA Performance

**Different Processes**
- PPN-2
- PPN-4
- PPN-8
- PPN-16

**No increase with PPN**

**Same Process, Same Buffer**

**Same Process, Diff Buffers**

**All-to-All – Good Scalability**

**One-to-All - Poor Scalability**

**One-to-All – Poor Scalability**

Contention is at Process level

Network Based Computing Laboratory
One-to-all Communication with CMA on Different Architectures

- Super-linear degradation in all three architectures
- More Cores => More Contention => More Degradation
- Contention aware design: limit number of concurrent reads/writes
  - Hit the “sweet spot” between contention and concurrency
Performance Comparison of MPI_Scatter

- Up to **2x-5x** improvement on KNL and Broadwell for medium to large messages
- Up to **20x** improvement on Power due to large process count
- Significantly faster than Intel MPI and Open MPI for messages > 4KB
- Similar improvements observed for MPI_Gather

Performance Comparison of MPI_Bcast

- Up to **2x - 4x** improvement over existing implementation for 1MB messages
- Up to **1.5x – 2x** faster than Intel MPI and Open MPI for 1MB messages

- Improvements obtained for **large messages only**
  - p-1 copies with CMA, p copies with Shared memory
  - Fallback to SHMEM for small messages
Performance Comparison of MPI_Alltoall

- Improvement from avoiding exchange of control messages
- Improvement observed even for 1KB messages

- Up to 3x-5x improvement for small and medium messages (compared to default)
- Large message performance bound by system bandwidth (5-20% improvement)
- Similar improvements for MPI_Allgather
Overview of A Few Challenges being Addressed by the MVAPICH2 Project for Many-core Era

- Fast and Scalable Job Start-up
- Dynamic and Adaptive Communication Protocols and Tag Matching
- Contention-aware Designs for Intra-node Collectives
- **Scalable Multi-leader Designs for Collectives**
- Kernel-Assisted Communication Designs for KNL
- Efficient RMA-based Designs for Graph500 on KNL
Scalable Reduction Collectives with Multi-Leaders

- Existing designs for MPI_Allreduce do not take advantage of the vast parallelism available in modern multi-/many-core processors
- Proposed a new solution for MPI_Allreduce
- DPML Take advantage of the parallelism offered by
  - Multi-/many-core architectures
  - The high throughput and high-end features offered by InfiniBand and Omni-Path

Performance of MPI_Allreduce On Stampede2 (1,024 processes)*

- For MPI_Allreduce latency with 8K bytes, MVAPICH2-OPT can reduce the latency by 4X

*In all our evaluations, Intel MPI 2017.1.132 and MVAPICH2 2.2 have been used

**Processes Per Node
Performance of MPI_Allreduce On Stampede2 (10,240 Processes)

- For MPI_Allreduce latency with 32K bytes, MVAPICH2-OPT can reduce the latency by 2.4X
For MPI_Allreduce latency with 2K bytes, MVAPICH2-OPT can reduce the latency by 2.6X.
For MPI_Allreduce latency with 4K bytes, MVAPICH2-OPT can reduce the latency by 1.5X.
Performance of MiniAMR Application On Stampede2 and Bridges

- For MiniAMR Application latency with 2,048 processes, MVAPICH2-OPT can reduce the latency by 2.6X on Stampede2.
- On Bridges, with 1,792 processes, MVAPICH2-OPT can reduce the latency by 1.5X.
Overview of A Few Challenges being Addressed by the MVAPICH2 Project for Many-core Era

- Fast and Scalable Job Start-up
- Dynamic and Adaptive Communication Protocols and Tag Matching
- Contention-aware Designs for Intra-node Collectives
- Scalable Multi-leader Designs for Collectives
- Kernel-Assisted Communication Designs for KNL
- Efficient RMA-based Designs for Graph500 on KNL
Kernel-assisted Communication Designs for KNL

- Proposed kernel-assisted on-loading communication engine for many-cores with high bandwidth memories
  - Exploits high concurrency and MCDRAM offered by KNL
- Implemented as a Linux Kernel Module with MPI as a high-level runtime
- Applicable to other programming models such as PGAS, Task-based etc.
- Provides portability, performance, and applicability to runtime as well as applications in a transparent manner
- Low latency and high throughput
  - Medium to large messages
  - Optimized for Deep Learning workloads

• Proposed design exploits KNL cores and MCRAM to accelerate large message transfers
• Two process latency is improved by up to 33% and bandwidth by 30%
HPCG weak-scaling execution time. \([nx, ny, nz]=[104, 104, 104]\)

- 15% improvement over Intel MPI 2017 for weak-scaling HPCG benchmark
- Proposed design improved MLP training time on CNTK framework by 12%

CNTK: MLP Training Time using MNIST (Batch-size:64)
Overview of A Few Challenges being Addressed by the MVAPICH2 Project for Many-core Era

- Fast and Scalable Job Start-up
- Dynamic and Adaptive Communication Protocols and Tag Matching
- Contention-aware Designs for Intra-node Collectives
- Scalable Multi-leader Designs for Collectives
- Kernel-Assisted Communication Designs for KNL
- Efficient RMA-based Designs for Graph500 on KNL
Multi-threading and Lock-Free MPI RMA Based Graph500 on KNL

- Propose multi-threading and lock-free designs in MPI runtime as well as Graph500 to take advantage of
  - Large number of CPU hardware threads
  - High-Bandwidth Memory (HBM)
For Put latency with 4M bytes, MV2-4T can reduce the latency by 3X
Graph500 BFS Kernel Performance Evaluation

- With Flat-Alltoall mode, G500-OPT-HBM scheme could reduce the kernel execution time by 27% compared with the G500-OPT-DDR
Graph500 BFS Kernel Scale-Out Evaluation

- With 1,024 processes, G500-OPT scheme can reduce the kernel execution time by 17\% compared with the G500-S/R scheme
Concluding Remarks

- Many-core nodes will be the foundation blocks for emerging Exascale systems
- Communication mechanisms and runtimes need to be re-designed to take advantage of the availability of large number of cores
- Presented a set of novel designs and demonstrated the performance benefits
- The new designs will be available in upcoming MVAPICH2 libraries
Funding Acknowledgments

Funding Support by

Equipment Support by
Personnel Acknowledgments

Current Students
- A. Awan (Ph.D.)
- M. Bayatpour (Ph.D.)
- S. Chakraborty (Ph.D.)
- C.-H. Chu (Ph.D.)
- S. Guganani (Ph.D.)
- N. Islam (Ph.D.)
- M. Li (Ph.D.)
- M. Rahman (Ph.D.)

Current Research Scientists
- X. Lu
- D. Shankar (Ph.D.)
- A. Venkatesh (Ph.D.)
- J. Zhang (Ph.D.)

Current Post-doc
- A. Ruhela

Past Students
- A. Augustine (M.S.)
- P. Balaji (Ph.D.)
- S. Bhagvat (M.S.)
- A. Bhat (M.S.)
- D. Buntinas (Ph.D.)
- L. Chai (Ph.D.)
- B. Chandrasekharan (M.S.)
- N. Dandapanthula (M.S.)
- V. Dhanraj (M.S.)
- T. Gangadharappa (M.S.)
- K. Gopalakrishnan (M.S.)
- W. Huang (Ph.D.)
- J. Jose (Ph.D.)
- S. Kini (M.S.)
- M. Koop (Ph.D.)
- K. Kulkarni (M.S.)
- R. Kumar (M.S.)
- S. Krishnamoorthy (M.S.)
- K. Kandalla (Ph.D.)
- P. Lai (M.S.)
- J. Liu (Ph.D.)
- M. Luo (Ph.D.)
- A. Mamidala (Ph.D.)
- G. Marsh (M.S.)
- V. Meshram (M.S.)
- A. Moody (M.S.)
- S. Naravula (Ph.D.)
- R. Noronha (Ph.D.)
- X. Ouyang (Ph.D.)
- S. Pai (M.S.)
- S. Potluri (Ph.D.)
- R. Rajachandrasekar (Ph.D.)
- G. Santhanaraman (Ph.D.)
- A. Singh (Ph.D.)
- J. Sridhar (M.S.)
- S. Sur (Ph.D.)
- H. Subramoni (Ph.D.)
- K. Vaidyanathan (Ph.D.)
- A. Vishnu (Ph.D.)
- J. Wu (Ph.D.)
- W. Yu (Ph.D.)

Past Research Scientist
- K. Hamidouche
- S. Sur

Past Programmers
- D. Bureddy
- J. Perkins

Past Post-Docs
- D. Banerjee
- X. Besseron
- H.-W. Jin
- J. Lin
- M. Luo
- E. Mancini
- S. Marcarelli
- J. Vienne
- H. Wang

Current Research Specialist
- J. Smith
- M. Arnold

Past Research Specialist
- J. Smith
Thank You!

panda@cse.ohio-state.edu

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/