Optimization and Tuning of MPI and PGAS Applications using MVAPICH

Latest version of the slides available at
http://go.osu.edu/mug14-tutorial

A Tutorial at MUG’14
by

The MVAPICH Team
The Ohio State University
E-mail: panda@cse.ohio-state.edu
http://www.cse.ohio-state.edu/~panda
Trends for Commodity Computing Clusters in the Top 500 List (http://www.top500.org)
Drivers of Modern HPC Cluster Architectures

- Multi-core processors are ubiquitous
- InfiniBand very popular in HPC clusters
- Accelerators/Coprocessors becoming common in high-end systems
- Pushing the envelope for Exascale computing

Multi-core Processors

High Performance Interconnects - InfiniBand
- <1usec latency, >100Gbps Bandwidth

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

Tianhe – 2 (1)  
Titan (2)  
Stampede (6)  
Tianhe – 1A (10)
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, GlobalArrays, OpenSHMEM), CUDA, OpenACC, Cilk, Hadoop, MapReduce, etc.

Communication Library or Runtime for Programming Models


Networking Technologies
(InfiniBand, 10/40GigE, Aries, BlueGene)

Multi/Many-core Architectures

Accelerators (NVIDIA and MIC)

Co-Design Opportunities and Challenges across Various Layers

MUG'14
Designing (MPI+X) at Exascale

- Scalability for million to billion processors
  - Support for highly-efficient inter-node and intra-node communication (both two-sided and one-sided)
  - Extremely minimum memory footprint
- Hybrid programming (MPI + OpenMP, MPI + UPC, MPI + OpenSHMEM, ...)
- Balancing intra-node and inter-node communication for next generation multi-core (128-1024 cores/node)
  - Multiple end-points per node
- Support for efficient multi-threading
- Scalable Collective communication
  - Offload
  - Non-blocking
  - Topology-aware
  - Power-aware
- Support for MPI-3 RMA Model
- Support for GPGPUs and Accelerators
- Fault-tolerance/resiliency
- QoS support for communication and I/O
MVAPICH2/MVAPICH2-X Software

- [http://mvapich.cse.ohio-state.edu](http://mvapich.cse.ohio-state.edu)

- 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
  - Used by more than 2,200 organizations (HPC Centers, Industry and Universities) in 73 countries

MVAPICH Team Projects

Timeline

- MVAPICH: Oct-00 to Nov-04
- OMB: Nov-04 to Jan-10
- MVAPICH2: Jan-10 to Nov-12
- MVAPICH2-GDR: Nov-12 to Aug-14
- MVAPICH2-X: Aug-14 to EOL

Timeline:

- Oct-00
- Jan-02
- Nov-04
- Jan-10
- Nov-12
- Aug-14
- EOL
MVAPICH/MVAPICH2 Release Timeline and Downloads

- Download counts from MVAPICH2 website
MVAPICH2/MVAPICH2-X Software (Cont’d)

• Available with software stacks of many IB, HSE, and server vendors including Linux Distros (RedHat and SuSE)

• Empowering many TOP500 clusters
  – 7th ranked 519,640-core cluster (Stampede) at TACC
  – 13th, 74,358-core (Tsubame 2.5) at Tokyo Institute of Technology
  – 23rd, 96,192-core (Pleiades) at NASA and many others

• Partner in the U.S. NSF-TACC Stampede System
Strong Procedure for Design, Development and Release

- Research is done for exploring new designs
- Designs are first presented to conference/journal publications
- Best performing designs are incorporated into the codebase
- Rigorous Q&A procedure before making a release
  - Exhaustive unit testing
  - Various test procedures on diverse range of platforms and interconnects
  - Performance tuning
  - Applications-based evaluation
  - Evaluation on large-scale systems
- Even alpha and beta versions go through the above testing
MVAPICH2 Architecture (Latest Release 2.0)

Major Computing Platforms: IA-32, EM64T, Ivybridge, Westmere, Sandybridge, Opteron, Magny, ..
MVAPICH2 2.0 and MVAPICH2-X 2.0

- Released on 06/20/2014
- Major Features and Enhancements
  - Based on MPICH-3.1
  - Extended support for MPI-3 RMA in OFA-IB-CH3, OFA-IWARP-CH3, and OFA-RoCE-CH3 interfaces
  - Multiple optimizations and performance enhancements for RMA in OFA-IB-CH3 channel
  - MPI-3 RMA support for CH3-PSM channel
  - CMA support is now enabled by default
  - Enhanced intra-node SMP performance
  - Dynamic CUDA initialization
  - Support for running on heterogeneous clusters with GPU and non-GPU nodes
  - Multi-rail support for GPU communication and UD-Hybrid channel
  - Improved job-startup performance for large-scale mpirun_rsh jobs
  - Reduced memory footprint
  - Updated compiler wrappers to remove application dependency on network and other extra libraries
  - Updated hwloc to version 1.9

MVAPICH2-X 2.0GA supports hybrid MPI + PGAS (UPC and OpenSHMEM) programming models.
  - Based on MVAPICH2 2.0GA including MPI-3 features; Compliant with UPC 2.18.0 and OpenSHMEM v1.0f
  - Enhanced optimization of OpenSHMEM collectives and Optimized shmalloc routine
  - Optimized UPC collectives and support for GUPC translator
**MVAPICH2-X for Hybrid MPI + PGAS Applications**

- Unified communication runtime for MPI, UPC, OpenSHMEM available with MVAPICH2-X 1.9 onwards!
  - [http://mvapich.cse.ohio-state.edu](http://mvapich.cse.ohio-state.edu)

- Feature Highlights
  - Supports MPI(+OpenMP), OpenSHMEM, UPC, MPI(+OpenMP) + OpenSHMEM, MPI(+OpenMP) + UPC
  - MPI-3 compliant, OpenSHMEM v1.0 standard compliant, UPC v1.2 standard compliant (with initial support for UPC 1.3)
  - Scalable Inter-node and intra-node communication – point-to-point and collectives
MVAPICH2-GDR 2.0

- Released on 08/23/2014
- Major Features and Enhancements
  - Based on MVAPICH2 2.0 (OFA-IB-CH3 interface)
  - High performance RDMA-based inter-node point-to-point communication (GPU-GPU, GPU-Host and Host-GPU) using GPUDirect RDMA and pipelining
  - Support for MPI-3 RMA (one Sided) communication and atomics using GPU buffers with GPUDirect RDMA and pipelining
  - Efficient small message inter-node communication using the new NVIDIA GDRCOPY module
  - Efficient small message inter-node communication using loopback design
  - Multi-rail support for inter-node point-to-point GPU communication
  - High performance intra-node point-to-point communication for multi-GPU adapters/node (GPU-GPU, GPU-Host and Host-GPU) using CUDA IPC and pipelining
  - Automatic communication channel selection for different GPU communication modes (DD, HH and HD) in different configurations (intra-IOH and inter-IOH)
  - Automatic selection and binding of the best GPU/HCA pair in multi-GPU/HCA system configuration
  - Optimized and tuned support for collective communication from GPU buffers
  - Enhanced and Efficient support for datatype processing on GPU buffers including support for vector/h-vector, index/h-index, array and subarray
  - Dynamic CUDA initialization. Support GPU device selection after MPI_Init
  - Support for non-blocking streams in asynchronous CUDA transfers for better overlap
  - Efficient synchronization using CUDA Events for pipelined device data transfers
OMB (OSU Micro-Benchmarks) 4.4

- Released on 08/23/2014
- Include benchmarks for MPI, OpenSHMEM, and UPC Programming Models
- **MPI**
  - Support MPI-1, MPI-2 and MPI-3 standards
  - Point-to-Point Benchmarks: Latency, Multi-threaded latency, Multi-pair latency, Multiple bandwidth / Message rate test bandwidth, Bidirectional bandwidth
  - Collective Benchmarks: Collective latency test for MPI_Allgather, MPI_Alltoall, MPI_Allreduce, MPI_BARRIER, MPI_Bcast, MPI_Gather, MPI_Reduce, MPI_Reduce_Scatter, MPI_Scatter, and vector collectives
  - One-sided Benchmarks: Put latency, Put bandwidth, Put bidirectional bandwidth, Get latency, Get bandwidth, Accumulate latency, Atomics latency, and Get_accumulate latency
  - CUDA Extensions to evaluate MPI communication performance from/to buffers on NVIDIA GPU devices
  - OpenACC Extensions to evaluate MPI communication performance from/to buffers on NVIDIA GPU devices
  - Support for MPI-3 RMA operations using GPU buffers

- **OpenSHMEM**
  - Point-to-Point benchmarks: Put latency, Get latency, Put message rate, and Atomics latency
  - Collective benchmarks: Collect latency, FCollect latency, Broadcast Latency, Reduce latency, and Barrier latency

- **Unified Parallel C (UPC)**
  - Point-to-Point benchmarks: Put latency, Get latency
  - Collective benchmarks: Barrier latency, Exchange latency, Gatherall latency, Gather latency, Reduce latency, and Scatter latency
Presentation Overview

• Runtime Optimization and Tuning Flexibility in MVAPICH2
  – Job start-up
  – Process Mapping Strategies
  – Point-to-point Inter-node Protocol
  – Transport Type Selection
  – Point-to-point Intra-node Protocol and Scheme
  – MPI-3 RMA
  – Collectives
  – Multi-rail, 3D Torus Networks and QoS
  – Fault-tolerance

• Runtime Optimization and Tuning Flexibility in MVAPICH2-GDR
• Runtime Optimization and Tuning Flexibility in MVAPICH2-X
• Runtime Optimization and Tuning Flexibility in MVAPICH2-MIC
• Advanced Optimization and Tuning of MPI Applications using MPI-T features in MVAPICH
• Overview of Installation, Configuration and Debugging Support
• Conclusions and Final Q&A
Job-Launcheders supported by MVAPICH2

Core Launchers

- mpirun_rsh
  - Default and preferred launcher for all interfaces
  - Supports scalable hierarchical job-startup mechanism
- mpiexec.hydra
  - Developed at ANL by the MPICH team

Wrappers and interfaces

- mpiexec
  - To support interfaces prescribed by the MPI standard
- mpiexec.mpirun_rsh
  - Hydra-like interface to mpirun_rsh

SLURM
- Compatibility with external resource managers

TORQUE
Scalable Job-Launching with mpirun_rsh (ScELA)

- Hierarchical launch
  - `mpirun_rsh` launches `mpispawn` on compute nodes
  - mpispawns launch application processes on processor cores

- `mpispawns` interconnect to form a k-ary tree to facilitate communication
- Common communication primitives built on mpispawn tree
Tuning Job-Launch with mpirun_rsh

- **MV2_MT_DEGREE**
  - degree of the hierarchical tree used by mpirun_rsh

- **MV2_FASTSSH_THRESHOLD**
  - #nodes beyond which hierarchical-ssh scheme is used

- **MV2_NPROCS_THRESHOLD**
  - #nodes beyond which file-based communication is used for hierarchical-ssh during start up

- **MV2_HOMOGENEOUS_CLUSTER**
  - Setting it optimizes startup for homogeneous clusters

- **MV2_ON_DEMAND_UD_INFO EXCHANGE**
  - To optimize start-up by exchanging UD connection info on-demand
Performance of Job Launcher on Stampede@TACC

- Several optimizations to enhance job startup performance
  - On-demand exchange of startup related information
  - 45% reduction in time for MPI hello world program at 4K cores
  - Run with 16 processes per node

ConnectX-3-FDR (54 Gbps): 2.7 GHz Dual Octa-core (SandyBridge) Intel PCI Gen3 with Mellanox IB FDR switch
Configuring and Using MVAPICH2 with SLURM

- Configure MVAPICH2 to use SLURM process manager
  - `./configure --with-pm=no --with-pmi=slurm`

- Applications compiled with MVAPICH2 configured with SLURM support can be launched by SLURM
  - `srun -n 2 ./a.out`

- Process Mapping with SLURM
  - In a two process job, to bind P0 to CPU 0 / Memory 0 and P1 to CPU 1 / Memory 1
    - `srun --cpu_bind=v,map_cpu:0,4 --mem_bind=v,map_mem:0,1 -n2 --mpi=none ./a.out`

- Refer to Running MVAPICH2 using SLURM section of MVAPICH2 user guide for more information
- [http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.0-userguide.html#x1-330005.2.3](http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.0-userguide.html#x1-330005.2.3)
Configuring and Using MVAPICH2 with PBS/Torque

- MVAPICH2 has been tightly integrated with PBS
- Use the Hydra process manager (mpiexec)
- Configure MVAPICH2 with --with-pbs option
  - ./configure <other options> --with-pbs
- Configure options for building Hydra with PBS support
  - --with-pbs=PATH
    - Path where pbs include directory and lib directory can be found
  - --with-pbs-include=PATH
    - Path where pbs include directory can be found
  - --with-pbs-lib=PATH
    - Path where pbs lib directory can be found
- Refer to Can I use MVAPICH2 on PBS/Torque Clusters section of MVAPICH2 user guide for more information
- http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.0-userguide.html#x1-1230009.2.1
Presentation Overview

- Runtime Optimization and Tuning Flexibility in MVAPICH2
  - Job start-up
  - Process Mapping Strategies
  - Point-to-point Inter-node Protocol
  - Transport Type Selection
  - Point-to-point Intra-node Protocol and Scheme
  - MPI-3 RMA
  - Collectives
  - Multi-rail, 3D Torus Networks and QoS
  - Fault-tolerance

- Runtime Optimization and Tuning Flexibility in MVAPICH2-GDR
- Runtime Optimization and Tuning Flexibility in MVAPICH2-X
- Runtime Optimization and Tuning Flexibility in MVAPICH2-MIC
- Advanced Optimization and Tuning of MPI Applications using MPI-T features in MVAPICH
- Overview of Installation, Configuration and Debugging Support
- Conclusions and Final Q&A
Impact of NUMA on IB Verbs Latency

- Cores in Socket 0 (closest to network card) have lowest latency
- Cores in Socket 1 (one hop from network card) have highest latency

ConnectX-3 FDR (54 Gbps): 2.6 GHz Octa-core (SandyBridge) Intel with IB (FDR) switches
Impact of NUMA on IB Verbs Bandwidth

- NUMA interactions have significant impact on bandwidth

ConnectX-3 FDR (54 Gbps): 2.6 GHz Octa-core (SandyBridge) Intel with IB (FDR) switches
ConnectX-2-QDR (36 Gbps): 2.5 GHz Hex-core (MagnyCours) AMD with IB (QDR) switches
Process Mapping support in MVAPICH2

Preset Binding Policies

- bunch (Default)
- scatter

User-defined binding

MPI rank-to-core binding

Granularity

- MVAPICH2 detects processor architecture at job-launch
Preset Process-binding Policies – Bunch

- “Core” level “Bunch” mapping (Default)
  - `MV2_CPU_BINDING_POLICY=bunch`

- “Socket/Numanode” level “Bunch” mapping
  - `MV2_CPU_BINDING_LEVEL=socket` `MV2_CPU_BINDING_POLICY=bunch`
Preset Process-binding Policies – Scatter

- “Core” level “Scatter” mapping
  - MV2_CPU_BINDING_POLICY=scatter

- “Socket/Numanode” level “Scatter” mapping
  - MV2_CPU_BINDING_LEVEL=socket MV2_CPU_BINDING_POLICY=scatter
User-Defined Process Mapping

• User has complete-control over process-mapping

• To run 4 processes on cores 0, 1, 4, 5:
  – $ mpirun_rsh -np 4 -hostfile hosts MV2_CPU_MAPPING=0:1:4:5 ./a.out

• Use ‘,’ or ‘-’ to bind to a set of cores:
  – $ mpirun_rsh -np 64 -hostfile hosts MV2_CPU_MAPPING=0,2-4:1:5:6 ./a.out

• Is process binding working as expected?
  – MV2_SHOW_CPU_BINDING=1
    • Display CPU binding information
    • Launcher independent
    • Example
      – MV2_SHOW_CPU_BINDING=1 MV2_CPU_BINDING_POLICY=scatter

            --------------CPU AFFINITY--------------
        RANK:0  CPU_SET:  0
        RANK:1  CPU_SET:  8

• Refer to Running with Efficient CPU (Core) Mapping section of MVAPICH2 user guide for more information
  • http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.0-userguide.html#x1-530006.5
Presentation Overview

- Runtime Optimization and Tuning Flexibility in MVAPICH2
  - Job start-up
  - Process Mapping Strategies
  - Point-to-point Inter-node Protocol
    - Eager and Rendezvous Protocols
    - RDMA Fast Path
  - Transport Type Selection
  - Point-to-point Intra-node Protocol and Scheme
  - MPI-3 RMA
  - Collectives
  - Multi-rail, 3D Torus Networks and QoS
  - Fault-tolerance

- Runtime Optimization and Tuning Flexibility in MVAPICH2-GDR
- Runtime Optimization and Tuning Flexibility in MVAPICH2-X
- Runtime Optimization and Tuning Flexibility in MVAPICH2-MIC
- Advanced Optimization and Tuning of MPI Applications using MPI-T features in MVAPICH
- Overview of Installation, Configuration and Debugging Support
- Conclusions and Final Q&A
Inter-node Point-to-Point Communication

- **EAGER** *(buffered, used for small messages)*
  - RDMA Fast Path
  - Send/Recv

- **RENNDEZVOUS** *(un-buffered, used for large messages)*
  - Reduces memory requirement by MPI library
  - Zero-Copy
  - No remote side involvement
  - Protocols
    - **RPUT** (RDMA Write)
    - **RGET** (RDMA Read)
    - **R3** (Send/Recv with Packetized Send)
Inter-node Point-to-Point Tuning: Eager Thresholds

Eager vs Rendezvous

- Switching Eager to Rendezvous transfer
  - Default: Architecture dependent on common platforms, in order to achieve both best performance and memory footprint
- Threshold can be modified by users to get smooth performance across message sizes
  - `mpirun_rsh -np 2 -f hostfile MV2_IBA_EAGER_THRESHOLD=32K a.out`
  - Memory footprint can increase along with eager threshold

Impact of Eager Threshold
Inter-node Point-to-Point Tuning: Number of Buffers and RNDV Protocols

- RDMA Fast Path has advantages for smaller message range (default is on)
  - Disable: mpirun_rsh \(-np\ 2 \--f\ hostfile MV2_USE_RDMA_FASTPATH=0\ a.out\)
- Adjust the number of RDMA Fast Path buffers (benchmark window size = 64):
  - mpirun_rsh \(-np\ 2 \--f\ hostfile MV2_NUM_RDMA_BUFFER=64\ a.out\)
- Switch between Rendezvous protocols depending on applications:
  - mpirun_rsh \(-np\ 2 \--f\ hostfile MV2_RNDV_PROTOCOL=RGET\ a.out\ (Default: RPUT)\)
Presentation Overview

- Runtime Optimization and Tuning Flexibility in MVAPICH2
  - Job start-up
  - Point-to-point Inter-node Protocol
  - Process Mapping Strategies
  - Transport Type Selection
    - Shared Receive Queue
    - eXtended Reliable Connect transport protocol
    - UD transport protocol and Hybrid
  - Point-to-point Intra-node Protocol and Scheme
  - MPI-3 RMA
  - Collectives
  - Multi-rail, 3D Torus Networks and QoS
  - Fault-tolerance

- Runtime Optimization and Tuning Flexibility in MVAPICH2-GDR
- Runtime Optimization and Tuning Flexibility in MVAPICH2-X
- Runtime Optimization and Tuning Flexibility in MVAPICH2-MIC
- Advanced Optimization and Tuning of MPI Applications using MPI-T features in MVAPICH
- Overview of Installation, Configuration and Debugging Support
- Conclusions and Final Q&A
SRQ is a hardware mechanism for a process to share receive resources (memory) across multiple connections

- Introduced in specification v1.2

\[ 0 < Q \ll P \cdot ((M \cdot N) - 1) \]
Using Shared Receive Queues with MVAPICH2

SRQ reduces the memory used by $\frac{1}{6}$ at 64,000 processes.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Significance</th>
<th>Default</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MV2_USE_SRQ</td>
<td>• Enable / Disable use of SRQ in MVAPICH2</td>
<td>Enabled</td>
<td>• Always Enable</td>
</tr>
<tr>
<td>MV2_SRQ_MAX_SIZE</td>
<td>• Limits the maximum size of the SRQ</td>
<td>4096</td>
<td>• Increase to 8192 for large scale runs</td>
</tr>
<tr>
<td>MV2_SRQ_SIZE</td>
<td>• Number of buffers posted to the SRQ</td>
<td>256</td>
<td>• Upper Bound: MV2_SRQ_MAX_SIZE</td>
</tr>
<tr>
<td></td>
<td>• Automatically doubled by MVAPICH2 on receiving SRQ LIMIT EVENT from IB HCA</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Refer to Shared Receive Queue (SRQ) Tuning section of MVAPICH2 user guide for more information.

http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.0-userguide.html#x1-980008.5
eXtended Reliable Connection (XRC)

- Each QP takes at least one page of memory
  - Connections between all processes is very costly for RC
- New IB Transport added: eXtended Reliable Connection
  - Allows connections between nodes instead of processes
• Memory usage for 32K processes with 8-cores per node can be 54 MB/process (for connections)
• NAMD performance improves when there is frequent communication to many peers
• Enabled by setting `MV2_USE_XRC` to 1 (Default: Disabled)
• Requires OFED version > 1.3
  – Unsupported in earlier versions (< 1.3), OFED-3.x and MLNX_OFED-2.0
  – MVAPICH2 build process will automatically disable XRC if unsupported by OFED
• Automatically enables SRQ and ON-DEMAND connection establishment

• Refer to eXtended Reliable Connection (XRC) section of MVAPICH2 user guide for more information
• [http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.0-userguide.html#x1-990008.6](http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.0-userguide.html#x1-990008.6)
Using UD Transport with MVAPICH2

Memory Footprint of MVAPICH2

<table>
<thead>
<tr>
<th>Number of Processes</th>
<th>Conn.</th>
<th>Buffers</th>
<th>Struct</th>
<th>Total</th>
<th>Buffers</th>
<th>Struct</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>RC (MVAPICH2 2.0b)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>512</td>
<td>22.9</td>
<td>24</td>
<td>0.3</td>
<td>47.2</td>
<td>24</td>
<td>0.2</td>
<td>24.2</td>
</tr>
<tr>
<td>1024</td>
<td>29.5</td>
<td>24</td>
<td>0.6</td>
<td>54.1</td>
<td>24</td>
<td>0.4</td>
<td>24.4</td>
</tr>
<tr>
<td>2048</td>
<td>42.4</td>
<td>24</td>
<td>1.2</td>
<td>67.6</td>
<td>24</td>
<td>0.9</td>
<td>24.9</td>
</tr>
<tr>
<td>UD (MVAPICH2 2.0b)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Performance with SMG2000

- Can use UD transport by configuring MVAPICH2 with the `--enable-hybrid`
  - Reduces QP cache trashing and memory footprint at large scale

Parameter | Significance | Default | Notes |
----------|--------------|---------|-------|
MV2_USE_ONLY_UD | Enable only UD transport in hybrid configuration mode | Disabled | RC/XRC not used |
MV2_USE_UD_ZCOPY | Enables zero-copy transfers for large messages on UD | Enabled | Always Enable when UD enabled |
MV2_UD_RETRY_TIMEOUT | Time (in usec) after which an unacknowledged message will be retried | 500000 | Increase appropriately on large / congested systems |
MV2_UD_RETRY_COUNT | Number of retries before job is aborted | 1000 | Increase appropriately on large / congested systems |

- Refer to Running with scalable UD transport section of MVAPICH2 user guide for more information
- [http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.0-userguide.html#x1-610006.10](http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.0-userguide.html#x1-610006.10)
Hybrid (UD/RC/XRC) Mode in MVAPICH2

- Both UD and RC/XRC have benefits
  - Hybrid for the best of both
- Enabled by configuring MVAPICH2 with the --enable-hybrid
- Available since MVAPICH2 1.7 as integrated interface

### Performance with HPCC Random Ring

- **Time (us)**
  - UD
  - Hybrid
  - RC

<table>
<thead>
<tr>
<th>Number of Processes</th>
<th>Time (us)</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>26%</td>
</tr>
<tr>
<td>256</td>
<td>40%</td>
</tr>
<tr>
<td>512</td>
<td>38%</td>
</tr>
<tr>
<td>1024</td>
<td>30%</td>
</tr>
</tbody>
</table>

### Parameter Significance

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Significance</th>
<th>Default</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MV2_USE_UD_HYBRID</td>
<td>Enable / Disable use of UD transport in Hybrid mode</td>
<td>Enabled</td>
<td>• Always Enable</td>
</tr>
<tr>
<td>MV2_HYBRID_ENABLE_THRESHOLD_SIZE</td>
<td>Job size in number of processes beyond which hybrid mode will be enabled</td>
<td>1024</td>
<td>• Uses RC/XRC connection until job size &lt; threshold</td>
</tr>
<tr>
<td>MV2_HYBRID_MAX_RC_CONN</td>
<td>Maximum number of RC or XRC connections created per process</td>
<td>64</td>
<td>• Prevents HCA QP cache thrashing</td>
</tr>
</tbody>
</table>
  - • Limit the amount of connection memory      |

- Refer to Running with Hybrid UD-RC/XRC section of MVAPICH2 user guide for more information
- [http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.0-userguide.html#x1-620006.11](http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.0-userguide.html#x1-620006.11)
Presentation Overview

• Runtime Optimization and Tuning Flexibility in MVAPICH2
  – Job start-up
  – Process Mapping Strategies
  – Point-to-point Inter-node Protocol
  – Transport Type Selection
  – Point-to-point Intra-node Protocol and Scheme
    • Shared-memory and LiMIC2/CMA based Communication
    • Architecture-based Tuning
  – MPI-3 RMA
  – Collectives
  – Multi-rail, 3D Torus Networks and QoS
  – Fault-tolerance

• Runtime Optimization and Tuning Flexibility in MVAPICH2-GDR
• Runtime Optimization and Tuning Flexibility in MVAPICH2-X
• Runtime Optimization and Tuning Flexibility in MVAPICH2-MIC
• Advanced Optimization and Tuning of MPI Applications using MPI-T features in MVAPICH
• Overview of Installation, Configuration and Debugging Support

• Conclusions and Final Q&A
Intra-node Communication Support in MVAPICH2

• Shared-Memory based two-copy intra-node communication
  – Copy from the sender’s user buffer to the shared buffer
  – Copy from the shared buffer to the receiver’s user buffer

• LiMIC2 on modern multi-core platforms
  – Kernel-level module for achieving single copy intra-node communication
  – LiMIC2 is used for rendezvous protocol message size
  – LiMIC2 module is required

• CMA (Cross Memory Attach) support
  – Single copy intra-node communication through Linux syscalls
  – Available from Linux kernel 3.2
MVAPICH2 Two-Sided Intra-Node Tuning:
Shared memory and Kernel-based Zero-copy Support (LiMIC and CMA)

- **LiMIC2:**
  - configure the library with ‘--with-limic2’
  - mpirun_rsh –np 2 –f hostfile a.out
  - To disable: MV2_SMP_USE_LIMIC2=0

- **CMA:**
  - Enabled by default (from MVAPICH2 2.0-GA) if supported by system
  - mpirun_rsh –np 2 –f hostfile a.out
  - To disable: MV2_SMP_USE_CMA=0

- When LiMIC2 and CMA are included at the same time, LiMIC2 is chosen by default
- If neither LiMIC2 or CMA is supported, shared-memory based design is chosen
MVAPICH2 Two-Sided Intra-Node Tuning:
Shared-Memory based Runtime Parameters

- Adjust eager threshold and eager buffer size:
  - `mpirun_rsh` –np 2 –f hostfile MV2_SMP_EAGERSIZE=16K MV2_SMPI_LENGTH_QUEUE=64 a.out
  - Will affect the performance of small messages and memory footprint

- Adjust number of buffers and buffer size for shared-memory based Rendezvous protocol:
  - `mpirun_rsh` –np 2 –f hostfile MV2_SMP_SEND_BUFFER=32 MV2_SMP_SEND_BUFF_SIZE=8192 a.out
  - Will affect the performance of large messages and memory footprint
Impact of Architecture-Specific Tuning

• Architecture-specific tuning is executed for new architectures and new designs introduced into MV2
• MV2_SMP_EAGERSIZE and MV2_SMP_SEND_BUFF_SIZE are updated from Default (1.8) to Tuned (2.0)
Presentation Overview

- **Runtime Optimization and Tuning Flexibility in MVAPICH2**
  - Job start-up
  - Process Mapping Strategies
  - Point-to-point Inter-node Protocol
  - Transport Type Selection
  - Point-to-point Intra-node Protocol and Scheme
  - MPI-3 RMA
    - InterNode Communication
    - IntraNode Communication
    - MPI-3 RMA Model
  - Collectives
  - Multi-rail, 3D Torus Networks and QoS
  - Fault-tolerance

- Runtime Optimization and Tuning Flexibility in MVAPICH2-GDR
- Runtime Optimization and Tuning Flexibility in MVAPICH2-X
- Runtime Optimization and Tuning Flexibility in MVAPICH2-MIC
- Advanced Optimization and Tuning of MPI Applications using MPI-T features in MVAPICH
- Overview of Installation, Configuration and Debugging Support
- Conclusions and Final Q&A
Internode One-sided Communication: Direct RDMA-based Designs

- MPI RMA offers one-sided communication
  - Separates communication and synchronization
  - Reduces synchronization overheads
  - Better computation and communication overlap
- Most MPI libraries implement RMA over send/recv calls
- MVAPICH2 offers direct RDMA-based implementation
  - Put/Get implemented as RDMA Write/Read
  - Better performance
  - Better computation-communication overlap

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Significance</th>
<th>Default</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MV2_USE_RDMA_ONE_SIDED</td>
<td>• Enable / Disable RDMA-based designs</td>
<td>1</td>
<td>• Disable only for debugging purposes</td>
</tr>
</tbody>
</table>
Intranode One-sided Communication

- MVAPICH2 provides truly one-sided implementation of RMA synchronization and communication within a node
  - Shared Memory Backed Windows

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Significance</th>
<th>Default</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MV2_USE_SHARED_MEM</td>
<td>• Enable / disable shared memory</td>
<td>1 (Enabled)</td>
<td>• Enable when using intra-node one-sided communication</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Requires window memory to be allocated using MPI_Win_allocate/MPI_Win_allocate_shared</td>
</tr>
</tbody>
</table>
MPI-3 RMA Model: Performance

- RDMA-based and truly 1-sided implementation of MPI-3 RMA in progress

**Atomics with Flush**

<table>
<thead>
<tr>
<th>Operations</th>
<th>Two sided</th>
<th>RDMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compare_and_swap</td>
<td>3</td>
<td>2.5</td>
</tr>
<tr>
<td>Fetch_and_op</td>
<td>3</td>
<td>2.5</td>
</tr>
</tbody>
</table>

**Get with Flush**

<table>
<thead>
<tr>
<th>Message size (byte)</th>
<th>RDMA</th>
<th>Two sided</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>64</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>256</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>1k</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>4k</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>16k</td>
<td>16</td>
<td>16</td>
</tr>
</tbody>
</table>

- MVAPICH2-2.0 and OSU micro-benchmarks (OMB v4.3.1)
- Better performance for MPI_Compare_and_swap and MPI_Fetch_and_op and MPI_Get performance with RDMA-based design
• Process 0 is busy in computation, Process 1 performs atomic operations at P0.
• These benchmarks show the latency of atomic operations. For RDMA based design, the atomic latency at P1 remains consistent even as the busy time at P0 increases.
Presentation Overview

- **Runtime Optimization and Tuning Flexibility in MVAPICH2**
  - Job start-up
  - Process Mapping Strategies
  - Point-to-point Inter-node Protocol
  - Transport Type Selection
  - Point-to-point Intra-node Protocol and Scheme
  - MPI-3 RMA
  - Collectives
    - Tuning Collective Communication Operations in MVAPICH2
    - Improved Bcast in MVAPICH2-2.0b
    - Non-blocking Collectives in MVAPICH2
  - Multi-rail, 3D Torus Networks and QoS
  - Fault-tolerance

- Runtime Optimization and Tuning Flexibility in MVAPICH2-GDR
- Runtime Optimization and Tuning Flexibility in MVAPICH2-X
- Runtime Optimization and Tuning Flexibility in MVAPICH2-MIC
- Advanced Optimization and Tuning of MPI Applications using MPI-T features in MVAPICH
- Overview of Installation, Configuration and Debugging Support
- Conclusions and Final Q&A
Collective Communication in MVAPICH2

Run-time flags:
All shared-memory based collectives:  MV2_USE_SHMEM_COLL (Default: ON)
Hardware Mcast-based collectives:  MV2_USE_MCAST (Default : OFF)
Hardware Multicast-aware MPI_Bcast on TACC Stampede

- MCAST-based designs improve latency of MPI_Bcast by up to 85%
- Use MV2_USE_MCAST =1 to enable MCAST-based designs
Enabling Hardware Multicast-aware

- Multicast is applicable to
  - MPI_Bcast
  - MPI_Scatter
  - MPI_Allreduce

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default Nature</th>
</tr>
</thead>
<tbody>
<tr>
<td>MV2_USE_MCAST = 1</td>
<td>Enables hardware Multicast features</td>
<td>Disabled</td>
</tr>
<tr>
<td>--enable-mcast</td>
<td>Configure flag to enable</td>
<td>Enabled</td>
</tr>
</tbody>
</table>
MPI_Scatter - Benefits of using Hardware-Mcast

- MCAST-based designs for MPI_Scatter improves small message latency by up to 75%
- Use MV2_USE_MCAST =1 to enable MCAST-based designs
Shared-memory Aware Collectives

- MVAPICH2 Reduce/Allreduce with 4K cores on TACC Ranger (AMD Barcelona, SDR IB)

- MVAPICH2 Barrier with 1K Intel Westmere cores, QDR IB

\[\text{MV2\_USE\_SHMEM\_BARRIER}=0/1\]
Problems with Blocking Collective Operations

- Communication time cannot be used for compute
  - No overlap of computation and communication
  - Inefficient
Application processes schedule collective operation

Check periodically if operation is complete

Overlap of computation and communication => Better Performance

Catch: Who will progress communication
Non-blocking Collective (NBC) Operations

• Enables overlap of computation with communication

• Non-blocking calls do not match blocking collective calls
  – MPI may use different algorithms for blocking and non-blocking collectives
  – Blocking collectives: Optimized for latency
  – Non-blocking collectives: Optimized for overlap

• A process calling a NBC operation
  – Schedules collective operation and immediately returns
  – Executes application computation code
  – Waits for the end of the collective

• The communication progress by
  – Application code through MPI_Test
  – Network adapter (HCA) with hardware support
  – Dedicated processes / thread in MPI library
Non-Blocking and Neighborhood Collectives in MVAPICH2

- MVAPICH2 provides supports for MPI-3 Non-Blocking Collective communication and Neighborhood collective communication primitives
  - MVAPICH2 1.9 and MVPICH2 2.0
- MPI-3 collectives in MVAPICH2 can use either the Gen2 or the nemesis interfaces, over InfiniBand
- MVAPICH2 implements non-blocking collectives either in a multi-core-aware hierarchical manner, or via a basic flat approach
- Application developers can use MPI-3 collectives to achieve computation/communication overlap
- Upcoming releases of MVAPICH2 will include support for non-blocking collectives based on Mellanox CORE-Direct interface
Micro-benchmarks to Measure Overlap

IMB 4.0.0 includes micro-benchmarks to measure overlap offered by an MPI implementation's collectives.

Methodology

- Compute latency of the collective without any background computation = $t_{\text{pure}}$
- Compute latency of the collective with background computation = $t_{\text{overlapped}}$
- \% overlap = $100 - (100 \times ((t_{\text{overlapped}} - t_{\text{pure}})/t_{\text{pure}}))$

The background computation ($t_{\text{CPU}}$) is run for approximately the same time as pure latency.
Strong Overlap Example Benchmarks

- Pure case = Igatherv+Wait
- Overlapped case = Igatherv+compute+Wait
- Igather and Igatherv show good overlap due to combination of use of eager protocol and one-sided designs through InfiniBand
Presentation Overview

• Runtime Optimization and Tuning Flexibility in MVAPICH2
  – Job start-up
  – Process Mapping Strategies
  – Point-to-point Inter-node Protocol
  – Transport Type Selection
  – Point-to-point Intra-node Protocol and Scheme
  – MPI-3 RMA
  – Collectives
    – Multi-rail, 3D Torus Networks and QoS
    – Fault-tolerance

• Runtime Optimization and Tuning Flexibility in MVAPICH2-GDR
• Runtime Optimization and Tuning Flexibility in MVAPICH2-X
• Runtime Optimization and Tuning Flexibility in MVAPICH2-MIC
• Advanced Optimization and Tuning of MPI Applications using MPI-T features in MVAPICH
• Overview of Installation, Configuration and Debugging Support
• Conclusions and Final Q&A
MVAPICH2 Multi-Rail Design

- What is a rail?
  - HCA, Port, Queue Pair

- Automatically detects and uses all active HCAs in a system
  - Automatically handles heterogeneity

- Supports multiple rail usage policies
  - Rail Sharing – Processes share all available rails
  - Rail Binding – Specific processes are bound to specific rails
Performance Tuning on Multi-Rail Clusters

---

Impact of Default Message Striping on Bandwidth

- **Single-Rail**
- **Dual-Rail**

Impact of Default Rail Binding on Message Rate

- **Single-Rail**
- **Dual-Rail**

Impact of Advanced Multi-rail Tuning on Message Rate

- Use First
- Round Robin
- Scatter
- Bunch

---

Two 24-core Magny Cours nodes with two Mellanox ConnectX QDR adapters

Six pairs with OSU Multi-Pair bandwidth and messaging rate benchmark

---

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Significance</th>
<th>Default</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MV2_IBA_HCA</td>
<td>• Manually set the HCA to be used</td>
<td>Unset</td>
<td>• To get names of HCA ibstat</td>
</tr>
<tr>
<td>MV2_DEFAULT_PORT</td>
<td>• Select the port to use on a active multi port HCA</td>
<td>0</td>
<td>• Set to use different port</td>
</tr>
<tr>
<td>MV2_RAIL_SHARING_LARGE_MSG _THRESHOLD</td>
<td>• Threshold beyond which striping will take place</td>
<td>16 Kbyte</td>
<td></td>
</tr>
<tr>
<td>MV2_RAIL_SHARING_POLICY</td>
<td>• Choose multi-rail rail sharing / binding policy</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• For Rail Sharing set to USE_FIRST or ROUND_ROBIN</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Set to FIXED_MAPPING for advanced rail binding options</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MV2_PROCESS_TO_RAIL_MAPPING</td>
<td>• Determines how HCAs will be mapped to the rails</td>
<td>BUNCH</td>
<td>• Options: SCATTER and custom list</td>
</tr>
</tbody>
</table>

---

- Refer to Enhanced design for Multiple-Rail section of MVAPICH2 user guide for more information

- [http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.0-userguide.html#x1-640006.13](http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.0-userguide.html#x1-640006.13)
Support for 3D Torus Networks in MVAPICH2/MVAPICH2-X

- Deadlocks possible with common routing algorithms in 3D Torus InfiniBand networks
  - Need special routing algorithm for OpenSM
- Users need to interact with OpenSM
  - Use appropriate SL to prevent deadlock
- MVAPICH2 supports 3D Torus Topology
  - Queries OpenSM at runtime to obtain appropriate SL
- Usage
  - Enabled at configure time
    - --enable-3dttorus-support
  - MV2_NUM_SA_QUERY_RETRIES
    - Control number of retries if PathRecord query fails
Exploiting QoS Support in MVAPICH2

- IB is capable of providing network level differentiated service – QoS
- Uses Service Levels (SL) and Virtual Lanes (VL) to classify traffic
- Enabled at configure time using CFLAG `ENABLE_QOS_SUPPORT`
- Check with System administrator before enabling
  - Can affect performance of other jobs in system

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Significance</th>
<th>Default</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MV2_USE_QOS</td>
<td>• Enable / Disable use QoS</td>
<td>Disabled</td>
<td>• Check with System administrator</td>
</tr>
<tr>
<td>MV2_NUM_SLS</td>
<td>• Number of Service Levels user requested</td>
<td>8</td>
<td>• Use to see benefits of Intra-Job QoS</td>
</tr>
<tr>
<td>MV2_DEFAULT_SERVICE_LEVEL</td>
<td>• Indicates the default Service Level to be used by job</td>
<td>0</td>
<td>• Set to different values for different jobs to enable Inter-Job QoS</td>
</tr>
</tbody>
</table>
Presentation Overview

• Runtime Optimization and Tuning Flexibility in MVAPICH2
  – Job start-up
  – Process Mapping Strategies
  – Point-to-point Inter-node Protocol
  – Transport Type Selection
  – Point-to-point Intra-node Protocol and Scheme
  – MPI-3 RMA
  – Collectives
  – Multi-rail, 3D Torus Networks and QoS
  – Fault-tolerance
    • Checkpoint-Restart Schemes
    • Process-Migration Schemes
    • Automatic Path Migration

• Runtime Optimization and Tuning Flexibility in MVAPICH2-GDR
• Runtime Optimization and Tuning Flexibility in MVAPICH2-X
• Runtime Optimization and Tuning Flexibility in MVAPICH2-MIC
• Advanced Optimization and Tuning of MPI Applications using MPI-T features in MVAPICH
• Overview of Installation, Configuration and Debugging Support
• Conclusions and Final Q&A
## Fault-Tolerance in MVAPICH2

### System-Level
- Checkpoint-Restart
- Process-migration
- Automatic Path Migration

### Application-aware
- App-aware CR
- Run-through Stabilization

### Feature Details

<table>
<thead>
<tr>
<th>Feature</th>
<th>MVAPICH2 version</th>
<th>Release Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLCR-based system-level MPI application Checkpointing</td>
<td>0.9.8</td>
<td>2006</td>
</tr>
<tr>
<td>FTB-enabled Checkpoint-Restart</td>
<td>1.4</td>
<td>2008</td>
</tr>
<tr>
<td>FUSE-assisted Write-Aggregation Scheme</td>
<td>1.6</td>
<td>2010</td>
</tr>
<tr>
<td>Basic File-Copy based Process Migration</td>
<td>1.6</td>
<td>2010</td>
</tr>
<tr>
<td>Pipelined Process Migration using RDMA</td>
<td>1.7</td>
<td>2011</td>
</tr>
<tr>
<td>Checkpoint-Restart support for the Nemesis-IB channel</td>
<td>1.8</td>
<td>2012</td>
</tr>
<tr>
<td>Scalable Multi-level Checkpointing using SCR</td>
<td>1.9</td>
<td>2013</td>
</tr>
<tr>
<td>Support for external job-launchers (SLURM, Hydra, etc.)</td>
<td>2.0</td>
<td>2014</td>
</tr>
<tr>
<td>More features under development (In-memory CR, DMTCP, etc.)</td>
<td>2.1</td>
<td>2014</td>
</tr>
</tbody>
</table>
System-Level Checkpoint-Restart

Job Start

Ckpt Rqst

Ckpt Rqst

Compute Node

Compute Node

Phase 1: Suspend communication between all processes

Phase 2: Use the checkpoint library (BLCR) to checkpoint the individual processes

Phase 3: Re-establish connections between the processes, and continue execution

MUG’14
Using the Checkpoint-Restart Feature

- Requires Berkeley Lab Checkpoint-Restart (BLCR) library
- Build with CR support: --enable-ckpt (or) --with-blcr=$PATH_TO_BLCR_INSTALLATION
- Launching the job:
  ```
  $mpirun_rsh -np 2 -hostfile ./hfile
  MV2_CKPT_FILE = /pfs/ckpt/app1
  MV2_CKPT_MAX_SAVE_CKPTS = 3
  MV2_CKPT_NO_SYNC = 0 ./a.out
  ```
- Triggering a checkpoint:
  - `$ cr_checkpoint -p <PID of mpirun_rsh>`
  - Run $MV2_INSTALL_DIR/bin/mv2_checkpoint and select the job to checkpoint
  - Call MVAPICH2_Sync_Checkpoint() from within the application
  - Set MV2_CKPT_INTERVAL = 30 for automated checkpointing
- Restarting from a checkpoint:
  - `$cr_restart /pfs/ckpt/context.<pid>`
- Refer to Running with Fault-Tolerance Support section of MVAPICH2 user guide for more information
- http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.0-userguide.html#x1-650006.14
Multicore-aware Checkpoint I/O Aggregation

- Requires FUSE version 2.8+, better performance for kernels newer than version 2.6.26
- Enable `--enable-ckpt-aggregation` or `--with-fuse=<path_to_fuse_installation>`
- Toggle at runtime using `MV2_CKPT_USE_AGGREGATION` variable
- Ensure that FUSE kernel module is loaded

Multicore-aware Checkpoint I/O Aggregation

• Tunable Parameters
  - `MV2_CKPT_AGGREGATION_BUFPOOL_SIZE` (size of buffer pool used to aggregate I/O)
  - `MV2_CKPT_AGGREGATION CHUNK_SIZE` (chunks in which coalesced data is written to disk)
Multi-Level Checkpointing with ScalableCR (SCR)

- LLNL’s Scalable Checkpoint/Restart library
- Can be used for application guided and application transparent checkpointing
- Effective utilization of storage hierarchy
  - **Local**: Store checkpoint data on node’s local storage, e.g. local disk, ramdisk
  - **Partner**: Write to local storage and on a partner node
  - **XOR**: Write file to local storage and small sets of nodes collectively compute and store parity redundancy data (RAID-5)
  - **Stable Storage**: Write to parallel file system

LLNL’s Scalable Checkpoint/Restart library

Checkpoint Cost and Resiliency

- Low
- High

MUG’14
Application-guided Multi-Level Checkpointing

Representative SCR-Enabled Application

- Checkpoint writing phase times of representative SCR-enabled MPI application
- **512** MPI processes (8 procs/node)
- Approx. **51 GB** checkpoints
**Transparent Multi-Level Checkpointing**

- **ENZO Cosmology application** – Radiation Transport workload
- Using MVAPICH2’s CR protocol instead of the application’s in-built CR mechanism
- **512** MPI processes (8 procs/node)
- Approx. **12.8 GB** checkpoints
Quality-of-Service Aware Checkpoint-Restart

### MPI Message Latency

![Graph showing MPI Message Latency](image)

### MPI Message Bandwidth

![Graph showing MPI Message Bandwidth](image)

### Anelastic Wave Propagation

(64 MPI processes)

<table>
<thead>
<tr>
<th>Message Size (Byte)</th>
<th>Latency (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>64K</td>
<td>8%</td>
</tr>
<tr>
<td>256K</td>
<td>17.9%</td>
</tr>
<tr>
<td>1M</td>
<td>0.9</td>
</tr>
<tr>
<td>4M</td>
<td>0.95</td>
</tr>
</tbody>
</table>

- default
- with I/O noise
- With noise isolated

### NAS Parallel Benchmark

Conjugate Gradient Class D

(64 MPI processes)

<table>
<thead>
<tr>
<th>Message Size (Byte)</th>
<th>Normalized Runtime (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>64K</td>
<td>32.8%</td>
</tr>
<tr>
<td>256K</td>
<td>9.31%</td>
</tr>
<tr>
<td>1M</td>
<td>0.9</td>
</tr>
<tr>
<td>4M</td>
<td>1</td>
</tr>
</tbody>
</table>

- default
- with I/O noise
- I/O noise isolated

---

RDMA-based Pipelined Process-Migration

Using the Process-Migration Feature

- Requires BLCR, Fault-Tolerance Backplane (FTB), and FUSE (RDMA-based)
- Build with Migration support: --enable-ckpt-migration
- Setup FTB and launch the job:
  
  ```
  $mpirun_rsh -np 4 -hostfile ./hosts -sparehosts ./spares ./a.out
  ```

- Triggering a migration:
  - Send SIGUSR2 signal to `mpispawn` on source/failing node
  - `$MV2_INSTALL_PATH/bin/mv2_trigger <hostname_of_source_node>`
  - Automatically triggered by FTB-IPMI available at
    [http://nowlab.cse.ohio-state.edu/projects/ftb-ib/#FTB-IPMI](http://nowlab.cse.ohio-state.edu/projects/ftb-ib/#FTB-IPMI)

- Refer to Job Pause-Migration-Restart Support section of MVAPICH2 user guide for more information
- [http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.0-userguide.html#x1-740006.14.3](http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.0-userguide.html#x1-740006.14.3)
Network-Level FT with Automatic Path Migration (APM)

- Allows recovery from network faults in the presence of multiple paths
- Enabled by the LID-Mask Count (LMC) mechanism
- Run with APM support:
  - `$ mpirun_rsh -np 2 host1 host2 MV2_USE_APM=1 ./a.out`
- Test APM in the absence of actual network faults:
  - `$ mpirun_rsh -np 2 host1 host2 MV2_USE_APM=1 MV2_USE_APM_TEST=1 ./a.out`
  - Periodically migrates between primary and alternate paths

- Refer to Network Fault Tolerance with Automatic Path Migration section of MVAPICH2 user guide
  - [http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.0-userguide.html#x1-760006.14.5](http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.0-userguide.html#x1-760006.14.5)
Run-Through Stabilization

- Proposal made to the MPI Forum’s FT working group pre-3.0
- Communication failures not treated as fatal errors
- Return error code on process failure to user-set handler
- Outstanding send/recv/wild-card recv (with MPI_ANY_SOURCE) posted to failed communicator returns error code
- Supported in the Nemesis-IB channel (\texttt{--with-device=ch3:nemesis:ib})
- Run with mpiexec.hydra
  - Set \texttt{MV2_RUN THROUGH STABILIZATION = 1}
  - Add \texttt{--disable-auto-cleanup} flag
- Query list of failed processes from application:
  - \texttt{MPI_Comm_get_attr(MPI_COMM_WORLD, MPICH_ATTR_FAILED PROCESSES, &failed_procs, &flag)};
Presentation Overview

- Runtime Optimization and Tuning Flexibility in MVAPICH2
  - Runtime Optimization and Tuning Flexibility in MVAPICH2-GDR
    - Designs: Overview, Performance and Tuning
    - Usage: Build, Initialization and Cleanup
    - Support for MPI + OpenACC
  - Runtime Optimization and Tuning Flexibility in MVAPICH2-X
  - Runtime Optimization and Tuning Flexibility in MVAPICH2-MIC
  - Advanced Optimization and Tuning of MPI Applications using MPI-T features in MVAPICH
  - Overview of Installation, Configuration and Debugging Support
  - Conclusions and Final Q&A
InfiniBand + GPU Systems (Past)

- Many systems today want to use systems that have both GPUs and high-speed networks such as InfiniBand
- Problem: Lack of a common memory registration mechanism
  - Each device has to pin the host memory it will use
  - Many operating systems do not allow multiple devices to register the same memory pages
- Previous solution:
  - Use different buffer for each device and copy data
Collaboration between Mellanox and NVIDIA to converge on one memory registration technique

Both devices register a common host buffer

- GPU copies data to this buffer, and the network adapter can directly read from this buffer (or vice-versa)

Note that GPU-Direct does not allow you to bypass host memory
Sample Code - Without MPI integration

• Naïve implementation with standard MPI and CUDA

At Sender:
cudaMemcpy(sbuf, sdev, ...);
MPI_Send(sbuf, size, ...);

At Receiver:
MPI_Recv(rbuf, size, ...);
cudaMemcpy(rdev, rbuf, ...);

• High Productivity and Poor Performance
Sample Code – User Optimized Code

- Pipelining at user level with non-blocking MPI and CUDA interfaces
- Code at Sender side (and repeated at Receiver side)

```c
At Sender:
for (j = 0; j < pipeline_len; j++)
cudaMemcpyAsync(sbuf + j * blk, sdev + j * blksz, ...);
for (j = 0; j < pipeline_len; j++) {
    while (result != cudaSuccess) {
        result = cudaStreamQuery(...);
        if(j > 0) MPI_Test(...);
    }
    MPI_Isend(sbuf + j * block_sz, blksz, ...);
} MPI_Waitall();
```

- User-level copying may not match with internal MPI design
- **High Performance and Poor Productivity**
Can this be done within MPI Library?

• Support GPU to GPU communication through standard MPI interfaces
  – e.g. enable MPI_Send, MPI_Recv from/to GPU memory

• Provide high performance without exposing low level details to the programmer
  – Pipelined data transfer which automatically provides optimizations inside MPI library without user tuning

• A new Design incorporated in MVAPICH2 to support this functionality
Sample Code – MVAPICH2-GPU

- MVAPICH2-GPU: standard MPI interfaces used
- Takes advantage of Unified Virtual Addressing (>= CUDA 4.0)
- Overlaps data movement from GPU with RDMA transfers

At Sender:

MPI_Send(s_device, size, ...);

At Receiver:

MPI_Recv(r_device, size, ...);

- High Performance and High Productivity
Pipelined Data Movement in MVAPICH2

- Pipelines data movement from the GPU, overlaps
  - device-to-host CUDA copies
  - inter-process data movement
    - network transfers or shared memory copies
  - host-to-device CUDA copies

- 45% improvement compared with a naïve (Memcpy+Send)
- 24% improvement compared with an advanced user-level implementation (MemcpyAsync+Isend)

CUDA-Aware MPI: MVAPICH2 1.8, 1.9, 2.0 Releases

• Support for MPI communication from NVIDIA GPU device memory
• High performance RDMA-based inter-node point-to-point communication (GPU-GPU, GPU-Host and Host-GPU)
• High performance intra-node point-to-point communication for multi-GPU adapters/node (GPU-GPU, GPU-Host and Host-GPU)
• Taking advantage of CUDA IPC (available since CUDA 4.1) in intra-node communication for multiple GPU adapters/node
• Optimized and tuned collectives for GPU device buffers
• MPI datatype support for point-to-point and collective communication from GPU device buffers
## Pipelined Data Movement in MVAPICH2: Tuning

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Significance</th>
<th>Default</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MV2_USE_CUDA</td>
<td>Enable / Disable GPU designs</td>
<td>0</td>
<td>(Disabled)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Disabled to avoid pointer checking overheads for host communication</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Always enable to support MPI communication from GPU Memory</td>
</tr>
<tr>
<td>MV2_CUDA_BLOCK_SIZE</td>
<td>Controls the pipeline blocksize</td>
<td>256 KByte</td>
<td>• Tune for your system and application</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Varies based on</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- CPU Platform, IB HCA and GPU</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- CUDA driver version</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Communication pattern (latency/bandwidth)</td>
</tr>
</tbody>
</table>

- Refer to Running on Clusters with NVIDIA GPU Accelerators section of MVAPICH2 user guide
- [http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.0-userguide.html#x1-810006.19](http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.0-userguide.html#x1-810006.19)
GPU-Direct RDMA (GDR) with CUDA

- Network adapter can directly read/write data from/to GPU device memory
- Avoids copies through the host
- Fastest possible communication between GPU and IB HCA
- Allows for better asynchronous communication
- OFED with GDR support is under development by Mellanox and NVIDIA
GPU-Direct RDMA (GDR) with CUDA

- OFED with support for GPUDirect RDMA is under work by NVIDIA and Mellanox
- OSU has an initial design of MVAPICH2 using GPUDirect RDMA
  - Hybrid design using GPU-Direct RDMA
    - GPUDirect RDMA and Host-based pipelining
    - Alleviates P2P bandwidth bottlenecks on SandyBridge and IvyBridge
  - Support for communication using multi-rail
  - Support for Mellanox Connect-IB and ConnectX VPI adapters
  - Support for RoCE with Mellanox ConnectX VPI adapters

SNB E5-2670 / IVB E5-2680V2

- Support for communication using multi-rail
- Support for Mellanox Connect-IB and ConnectX VPI adapters
- Support for RoCE with Mellanox ConnectX VPI adapters
Performance of MVAPICH2 with GPU-Direct-RDMA: Latency

GPU-GPU Internode MPI Latency

Small Message Latency

Latency (us)

Message Size (bytes)

MV2-GDR2.0
MV2-GDR2.0b
MV2 w/o GDR

MVAPICH2-GDR-2.0
Intel Ivy Bridge (E5-2680 v2) node with 20 cores
NVIDIA Tesla K40c GPU, Mellanox Connect-IB Dual-FDR HCA
CUDA 6.5, Mellanox OFED 2.1 with GPU-Direct-RDMA

Latency (us):
- MV2-GDR2.0: 2.18 usec
- MV2-GDR2.0b: 71%
- MV2 w/o GDR: 90%
Performance of MVAPICH2 with GPU-Direct-RDMA: Bandwidth

GPU-GPU Internode MPI Uni-Directional Bandwidth

Small Message Bandwidth

- MV2-GDR2.0
- MV2-GDR2.0b
- MV2 w/o GDR

Bandwidth (MB/s)

Message Size (bytes)

MVAPICH2-GDR-2.0
Intel Ivy Bridge (E5-2680 v2) node with 20 cores
NVIDIA Tesla K40c GPU, Mellanox Connect-IB Dual-FDR HCA
CUDA 6.5, Mellanox OFED 2.1 with GPU-Direct-RDMA
Performance of MVAPICH2 with GPU-Direct-RDMA: Bi-Bandwidth

GPU-GPU Internode MPI Bi-directional Bandwidth

Small Message Bi-Bandwidth

- MV2-GDR2.0
- MV2-GDR2.0b
- MV2 w/o GDR

Message Size (bytes)

MVAPICH2-GDR-2.0
Intel Ivy Bridge (E5-2680 v2) node with 20 cores
NVIDIA Tesla K40c GPU, Mellanox Connect-IB Dual-FDR HCA
CUDA 6.5, Mellanox OFED 2.1 with GPU-Direct-RDMA
Performance of MVAPICH2 with GPU-Direct-RDMA: MPI-3 RMA

GPU-GPU Internode MPI Put latency (RMA put operation Device to Device)

MPI-3 RMA provides flexible synchronization and completion primitives

![Graph showing small message latency vs. message size](image)

**Small Message Latency**
- MV2-GDR2.0
- MV2-GDR2.0b

- Intel Ivy Bridge (E5-2680 v2) node with 20 cores
- NVIDIA Tesla K40c GPU, Mellanox Connect-IB Dual-FDR HCA
- CUDA 6.5, Mellanox OFED 2.1 with GPU-Direct-RDMA
Using MVAPICH2-GPUDirect Version

• MVAPICH2-2.0 with GDR support can be downloaded from [https://mvapich.cse.ohio-state.edu/download/mvapich2gdr/](https://mvapich.cse.ohio-state.edu/download/mvapich2gdr/)

• System software requirements
  • Mellanox OFED 2.1 or later
  • NVIDIA Driver 331.20 or later
  • NVIDIA CUDA Toolkit 6.0 or later
  • Plugin for GPUDirect RDMA
    – Strongly Recommended: use the new GDRCOPY module from NVIDIA
      • [https://github.com/drossetti/gdrcopy](https://github.com/drossetti/gdrcopy)
  
• Has optimized designs for point-to-point communication using GDR

• Contact MVAPICH help list with any questions related to the package
  mvapich-help@cse.ohio-state.edu
## GPUDirect RDMA (GDR) Designs in MVAPICH2: Tuning

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Significance</th>
<th>Default</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MV2_USE_GPUDIRECT</td>
<td>Enable / Disable GDR-based designs</td>
<td>1 (Disabled)</td>
<td>• Always enable</td>
</tr>
<tr>
<td>MV2_GPUDIRECT_LIMIT</td>
<td>Controls messages size until which GPUDirect RDMA is used</td>
<td>8 KByte</td>
<td>• Tune for your system&lt;br&gt;• GPU type, host architecture and CUDA version: impact pipelining overheads and P2P bandwidth bottlenecks</td>
</tr>
<tr>
<td>MV2_USE_GPUDIRECT_GDRCOPY_LIMIT</td>
<td>Controls messages size until which the GDRCOPY is used</td>
<td>8 KByte</td>
<td>• Tune for your system&lt;br&gt;• GPU type, host architecture and CUDA version and application pattern: impact the copy between CPU and GPU</td>
</tr>
<tr>
<td>MV2_USE_GPUDIRECT_GDRCOPY_NAIVE_LIMIT</td>
<td>Controls messages size until which the GDRCOPY is used for collective operations</td>
<td>8 KByte</td>
<td>• Tune for your system&lt;br&gt;• GPU type, host architecture and CUDA version and application pattern: impact the copy between CPU and GPU</td>
</tr>
</tbody>
</table>
## GPUDirect RDMA (GDR) Designs in MVAPICH2: Tuning

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Significance</th>
<th>Default</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MV2_USE_GPUDIRECT_LOOPBACK_LIMIT</td>
<td>• Controls messages size until which the loopback feature is used</td>
<td>8 KByte</td>
<td>• Tune for your system • GPU type, host architecture and CUDA version and application pattern: impact the copy between CPU and GPU</td>
</tr>
<tr>
<td>MV2_USE_GPUDIRECT_LOOPBACK_NAIVE_LIMIT</td>
<td>• Controls messages size until which the loopback is used for collective operations</td>
<td>8 KByte</td>
<td>• Tune for your system • GPU type, host architecture and CUDA version and application pattern: impact the copy between CPU and GPU</td>
</tr>
</tbody>
</table>
Multi-GPU Configurations

- Multi-GPU node architectures are becoming common
- Until CUDA 3.2
  - Communication between processes staged through the host
  - Shared Memory (pipelined)
  - Network Loopback [asynchronous]
- CUDA 4.0 and later
  - Inter-Process Communication (IPC)
  - Host bypass
  - Handled by a DMA Engine
  - Low latency and Asynchronous
  - Requires creation, exchange and mapping of memory handles - overhead
Designs in MVAPICH2 and Performance

- MVAPICH2 takes advantage of CUDA IPC for MPI communication between GPUs
- Hides the complexity and overheads of handle creation, exchange and mapping
- Available in standard releases from MVAPICH2 1.8
## Runtime Parameters

- Works between GPUs within the same socket or IOH

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Significance</th>
<th>Default</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MV2_CUDA_IPC</td>
<td>• Enable / Disable CUDA IPC-based designs</td>
<td>1 (Enabled)</td>
<td>• Always leave set to 1</td>
</tr>
</tbody>
</table>
| MV2_CUDA_SMP_IPC         | • Enable / Disable CUDA IPC fastpath design for short messages                | 0 (Disabled)  | • Benefits Device-to-Device transfers  
                          |                                                               |               | • Hurts Device-to-Host/Host-to-Device transfers  
                          |                                                               |               | • Always set to 1 if application involves only Device-to-Device transfers |

![Graph](attachment://graph.png)

- Graph: Intranode osu_latency small

- Chart: SHARED-MEM, IPC, SMP-IPC

- Chart X-axis: Message Size (Bytes)

- Chart Y-axis: Latency (usec)
Non-contiguous Data Exchange

Halo data exchange

- Multi-dimensional data
  - Row based organization
  - Contiguous on one dimension
  - Non-contiguous on other dimensions

- Halo data exchange
  - Duplicate the boundary
  - Exchange the boundary in each iteration
MPI Datatype support in MVAPICH2

• Datatypes support in MPI
  – Operate on customized datatypes to improve productivity
  – Enable MPI library to optimize non-contiguous data

   At Sender:
   
   ```c
   MPI_Type_vector (n_blocks, n_elements, stride, old_type, &new_type);
   MPI_Type_commit(&new_type);
   ...
   MPI_Send(s_buf, size, new_type, dest, tag, MPI_COMM_WORLD);
   ```

• Inside MVAPICH2
  - Use datatype specific CUDA Kernels to pack data in chunks
  - Efficiently move data between nodes using RDMA
  - In progress - currently optimizes `vector` and `hindexed` datatypes
  - Transparent to the user

MPI Datatype support in MVAPICH2 (Cont)

• Comprehensive support
  • Targeted kernels for regular datatypes - vector, subarray, indexed_block
  • Generic kernels for all other irregular datatypes

• Separate non-blocking stream for kernels launched by MPI library
  • Avoids stream conflicts with application kernels

• Flexible set of parameters for users to tune kernels

  • Vector
    • MV2_CUDA_KERNEL_VECTOR_TIDBLK_SIZE
    • MV2_CUDA_KERNEL_VECTOR_YSIZE

  • Subarray
    • MV2_CUDA_KERNEL_SUBARR_TIDBLK_SIZE
    • MV2_CUDA_KERNEL_SUBARR_XDIM
    • MV2_CUDA_KERNEL_SUBARR_YDIM
    • MV2_CUDA_KERNEL_SUBARR_ZDIM

  • Indexed_block
    • MV2_CUDA_KERNEL_IDXBLK_XDIM

---

Application-Level Evaluation (LBMGPU-3D)

3D LBM-CUDA

- LBM-CUDA (Courtesy: Carlos Rosales, TACC)
  - Lattice Boltzmann Method for multiphase flows with large density ratios
  - 3D LBM-CUDA: one process/GPU per node, 512x512x512 data grid, up to 64 nodes
- Oakley cluster at OSC: two hex-core Intel Westmere processors, two NVIDIA Tesla M2070, one Mellanox IB QDR MT26428 adapter and 48 GB of main memory
Application-Level Evaluation (HOOMD-blue)

• Platform: Wilkes (Intel Ivy Bridge + NVIDIA Tesla K20c + Mellanox Connect-IB)
• MV2-GDR 2.0 (released on 08/23/14) : try it out !
  • GDRCOPY enabled: MV2_USE_CUDA=1 MV2_IBA_HCA=mlx5_0
    MV2_IBA_EAGER_THRESHOLD=32768 MV2_VBUF_TOTAL_SIZE=32768
    MV2_USE_GPUDIRECT_LOOPBACK_LIMIT=32768 MV2_USE_GPUDIRECT_GDRCOPY=1
    MV2_USE_GPUDIRECT_GDRCOPY_LIMIT=16384
Enabling Support for MPI Communication from GPU Memory

• Configuring the support
  – --enable-cuda --with-cuda=<path-to-cuda-installation>
  – With PGI compilers
    • PGI does not handle an asm string leading to linker issues with CUDA kernels in MVAPICH2 library
    • --enable-cuda=basic --with-cuda=<path-to-cuda-installation>

• Enabling the support at runtime
  – Disabled by default to avoid pointer checking overheads for Host communication
  – Set MV2_USE_CUDA=1 to enable support of communication from GPU memory
GPU Device Selection, Initialization and Cleanup

• MVAPICH2 1.9 and earlier versions require GPU device to be selected before MPI_Init
  – To allocate and initialize required GPU resources in MPI_Init
  – Node rank information is exposed by the launchers (mpirun_rsh or hydra) as MV2_COMM_WORLD_LOCAL_RANK
    
    int local_rank = atoi(getenv("MV2_COMM_WORLD_LOCAL_RANK"));
    cudaSetDevice (local_rank% num_devices)

• MVAPICH2 2.0 and onwards remove this restriction
  – Does GPU resource initialization dynamically
  – Applications can select the device before or after MPI_Init

• CUDA allocates resources like shared memory files which require explicit cleanup
  – cudaDeviceReset or cuCtxDestroy
  – Applications have to do this after MPI_Finalize to allow MVAPICH2 runtime to deallocate resources
OpenACC

- OpenACC is gaining popularity
- Several sessions during GTC
- A set of compiler directives (#pragma)
- Offload specific loops or parallelizable sections in code onto accelerators
  ```
  #pragma acc region
  {
    for(i = 0; i < size; i++) {
      A[i] = B[i] + C[i];
    }
  }
  ```
- Routines to allocate/free memory on accelerators
  ```
  buffer = acc_malloc(MYBUFSIZE);
  acc_free(buffer);
  ```
- Supported for C, C++ and Fortran
- Huge list of modifiers – copy, copyout, private, independent, etc..

112
Using MVAPICH2 with OpenACC 1.0

- `acc_malloc` to allocate device memory
  - No changes to MPI calls
  - MVAPICH2 detects the device pointer and optimizes data movement
  - Delivers the same performance as with CUDA

```c
A = acc_malloc(sizeof(int) * N);

......

#pragma acc parallel loop deviceptr(A) . . .
//compute for loop

MPI_Send (A, N, MPI_INT, 0, 1, MPI_COMM_WORLD);

......
acc_free(A);
```
Using MVAPICH2 with OpenACC 2.0

- **acc_deviceptr** to get device pointer (in OpenACC 2.0)
  - Enables MPI communication from memory allocated by compiler when it is available in OpenACC 2.0 implementations
  - MVAPICH2 will detect the device pointer and optimize communication
  - Delivers the same performance as with CUDA

```c
A = malloc(sizeof(int) * N);

......

#pragma acc data copyin(A) . . .
{

#pragma acc parallel loop . . .
//compute for loop
MPI_Send(acc_deviceptr(A), N, MPI_INT, 0, 1, MPI_COMM_WORLD);

}

......
free(A);
```
CUDA and OpenACC Extensions in OMB

- OSU Micro-benchmarks are widely used to compare performance of different MPI stacks and networks
- Enhancements to measure performance of MPI communication from GPU memory
  - Point-to-point: Latency, Bandwidth and Bi-directional Bandwidth
  - RMA (one-sided) and atomics operations
  - Collectives: Latency test for all collectives
- Support for CUDA and OpenACC
- Flexible selection of data movement between CPU(H) and GPU(D): D->D, D->H and H->D
- Available from [http://mvapich.cse.ohio-state.edu/benchmarks](http://mvapich.cse.ohio-state.edu/benchmarks)
- Available in an integrated manner with MVAPICH2 stack
Configuring, Building and Running OMB

• Configuring with Runtime Optimization and Tuning Flexibility in MVAPICH2-GDR
  – Enabling CUDA support: “--enable-cuda --with-cuda-include=<path-to-CUDA-headers>
    --with-cuda-libraries=<path-to-CUDA-libraries>”
  – Enabling OpenACC support: “--enable-openacc”
  – Both enabled in integrated version when library is build with CUDA support

• Running the benchmarks
  – “pt2pt-benchmark [options] [RANK0 RANK1]”
  – “collective-benchmark [options]”

• Option to select between CUDA and OpenACC: “-d [cuda|openacc]”

• Parameters to select location of buffers
  – Pt2pt benchmarks support selection at each rank: “D D, D H, H D, H H”
  – The –d option enables use of device buffers in collective benchmarks
Device Selection

- MVAPICH2 1.8, 1.9 and other MPI libraries require device selection before MPI_Init
- This restriction has been removed with MVAPICH2 2.0 and later
- OSU micro-benchmarks still select device before MPI_Init for backward compatibility
- Uses node-level rank information exposed by launchers
- We provide a script which exports this node rank information to be used by the benchmark
  - Can be modified to work with different MPI libraries without modifying the benchmarks themselves
  - Sample script provided with OMB: “get_local_rank”
    
    ```
    export LOCAL_RANK=$MV2_COMM_WORLD_LOCAL_RANK
    exec $*
    ```
Examples

Consider two GPU nodes: \textit{n1 and n2 each with two GPUs}

Measure internode GPU-to-GPU latency using CUDA

\begin{verbatim}
mpirun_rsh –np 2 n1 n2 MV2_USE_CUDA=1 ./get_local_rank ./osu_latency D D
mpirun_rsh –np 2 n1 n2 MV2_USE_CUDA=1 ./get_local_rank ./osu_latency –d cuda D D
\end{verbatim}

Measure internode GPU-to-GPU latency using OpenACC

\begin{verbatim}
mpirun_rsh –np 2 n1 n2 MV2_USE_CUDA=1 ./get_local_rank ./osu_latency –d openacc D D
\end{verbatim}

Measure internode GPU-to-Host latency using CUDA

\begin{verbatim}
mpirun_rsh –np 2 n1 n2 MV2_USE_CUDA=1 ./get_local_rank ./osu_latency D H
\end{verbatim}
Examples

Measure intranode GPU-to-GPU latency

```bash
mpirun_rsh –np 2 n1 n1 MV2_USE_CUDA=1 ./get_local_rank ./osu_latency D D
```

Measure MPI_Alltoall latency between GPUs with CUDA

```bash
mpirun_rsh –np 4 n1 n1 n2 n2 MV2_USE_CUDA=1 ./get_local_rank ./osu_alltoall –d CUDA
```
Presentation Overview

- Runtime Optimization and Tuning Flexibility in MVAPICH2
- Runtime Optimization and Tuning Flexibility in MVAPICH2-GDR

- **Runtime Optimization and Tuning Flexibility in MVAPICH2-X**
  - MVAPICH2-X, Feature highlights
  - Usage instructions
  - Performance highlights
  - FAQs

- Runtime Optimization and Tuning Flexibility in MVAPICH2-MIC
- Advanced Optimization and Tuning of MPI Applications using MPI-T features in MVAPICH
- Overview of Installation, Configuration and Debugging Support
- Conclusions and Final Q&A
MVAPICH2-X

• Unified communication runtime for MPI, UPC, OpenSHMEM available with MVAPICH2-X 1.9 onwards!
  – [http://mvapich.cse.ohio-state.edu](http://mvapich.cse.ohio-state.edu)

• Feature Highlights
  – Supports MPI(+OpenMP), OpenSHMEM, UPC, MPI(+OpenMP) + OpenSHMEM, MPI(+OpenMP) + UPC
  – MPI-3 compliant, OpenSHMEM v1.0 standard compliant, UPC v1.2 (initial support for UPC v1.3) standard compliant
  – Scalable Inter-node communication with high performance and reduced memory footprint
  – Optimized Intra-node communication using shared memory schemes
  – Optimized OpenSHMEM collectives
  – Supports different CPU binding policies
  – Flexible process manager support
• MVAPICH2-X RPMs available for:
  – Enterprise Linux 5
    • Compatible with OFED 1.5.4.1 and MLNX_OFED 1.5.3
  – Enterprise Linux 5 (Stock InfiniBand Packages)
    • Compatible with OFED 3.5, MLNX_OFED 2.0 and RHEL5 InfiniBand packages
  – Enterprise Linux 6
    • Compatible with OFED 1.5.4.1 and MLNX_OFED 1.5.3
  – Enterprise Linux 6 (Stock InfiniBand Packages)
    • Compatible with OFED 3.5, MLNX_OFED 2.0 and RHEL6 InfiniBand packages
  – Please contact us at mvapich-help@cse.ohio-state.edu for other platforms

• MVAPICH2-X RPMs are relocatable
Downloading and Installing MVAPICH2-X

- **Downloading MVAPICH2-X RPMs**
  
  - `wget http://mvapich.cse.ohio-state.edu/download/mvapich2x/mvapich2-x-2.0.rhel6.tar.gz`

- **Tarball contents:**
  
  - GNU and Intel RPMs for MVAPICH2-X, OpenSHMEM, and UPC

- **Install using rpm command**
  
  - `rpm --Uvh [--prefix=install-path] *.rpm --force --nodeps`
  
  - Default installation location is /opt/mvapich2-x
Compiling programs with MVAPICH2-X

- Compile MPI programs using mpicc
  
  $ mpicc -o helloworld_mpi helloworld_mpi.c

- Compile UPC programs using upcc
  
  $ upcc -o helloworld_upc helloworld_upc.c

- Compile OpenSHMEM programs using oshcc
  
  $ oshcc -o helloworld_oshm helloworld_oshm.c

- Compile Hybrid MPI+UPC programs using upcc
  
  $ upcc -o hybrid_mpi_upc hybrid_mpi_upc.c

- Compile Hybrid MPI+OpenSHMEM programs using oshcc
  
  $ oshcc -o hybrid_mpi_oshm hybrid_mpi_oshm.c
Running Programs with MVAPICH2-X

- MVAPICH2-X programs can be run using
  - mpirun_rsh and mpiexec.hydra (MPI, UPC, OpenSHMEM and hybrid)
  - upcrun (UPC)
  - oshrun (OpenSHMEM)

- Running using mpirun_rsh/mpiexec.hydra
  - $ mpirun rsh -np 4 -hostfile hosts ./test
  - $ mpiexec -f hosts -n 2 ./test

- Running using upcrun
  - $ export MPIRUN CMD="<path-to-MVAPICH2-X-install>/bin/mpirun rsh -np %N -hostfile hosts %P %A"
  - $ upcrun -n 2 ./test

- Running using oshrun
  - $ oshrun -f hosts -np 2 ./test
OSU Microbenchmarks – UPC and OpenSHMEM

- OpenSHMEM Benchmarks
  - Point-to-point Benchmarks
    - osu_oshm_put, osu_oshm_get, osu_oshm_put_mr, osu_oshm_atomics
  - Collective Benchmarks
    - osu_oshm_collect, osu_oshm_broadcast, osu_oshm_reduce, osu_oshm_barrier

- UPC Benchmarks
  - Point-to-point Benchmarks
    - osu_upc_memput, osu_upc_memget
  - Collective Benchmarks
    - osu_upc_all_broadcast, osu_upc_all_scatter, osu_upc_all_gather, osu_upc_all_gather_all, osu_upc_all_exchange, osu_upc_all_barrier, osu_upc_all_reduce
OpenSHMEM Collective Communication Performance

Reduce (1,024 processes)

<table>
<thead>
<tr>
<th>Message Size</th>
<th>MV2X-SHMEM</th>
<th>Scalable-SHMEM</th>
<th>OMPI-SHMEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1000</td>
<td>100</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>10000</td>
<td>1000</td>
<td>100</td>
</tr>
<tr>
<td>100</td>
<td>100000</td>
<td>10000</td>
<td>1000</td>
</tr>
<tr>
<td>1000</td>
<td>1000000</td>
<td>100000</td>
<td>10000</td>
</tr>
</tbody>
</table>

Time (us)

Message Size

Broadcast (1,024 processes)

<table>
<thead>
<tr>
<th>Message Size</th>
<th>MV2X-SHMEM</th>
<th>Scalable-SHMEM</th>
<th>OMPI-SHMEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>100</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>16</td>
<td>1000</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>64</td>
<td>10000</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>256</td>
<td>100000</td>
<td>10000</td>
<td>10000</td>
</tr>
<tr>
<td>1K</td>
<td>1000000</td>
<td>100000</td>
<td>100000</td>
</tr>
</tbody>
</table>

Time (us)

Message Size

Collect (1,024 processes)

<table>
<thead>
<tr>
<th>Message Size</th>
<th>MV2X-SHMEM</th>
<th>Scalable-SHMEM</th>
<th>OMPI-SHMEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>100</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>16</td>
<td>1000</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>64</td>
<td>10000</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>256</td>
<td>100000</td>
<td>10000</td>
<td>10000</td>
</tr>
<tr>
<td>1K</td>
<td>1000000</td>
<td>100000</td>
<td>100000</td>
</tr>
</tbody>
</table>

Time (us)

Message Size

Barrier

<table>
<thead>
<tr>
<th>No. of Processes</th>
<th>Time (us)</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>100</td>
</tr>
<tr>
<td>256</td>
<td>150</td>
</tr>
<tr>
<td>512</td>
<td>200</td>
</tr>
<tr>
<td>1024</td>
<td>250</td>
</tr>
<tr>
<td>2048</td>
<td>300</td>
</tr>
</tbody>
</table>

Time (us)

No. of Processes

Application Level Performance with Graph500 and Sort

**Graph500 Execution Time**

- Performance of Hybrid (MPI+OpenSHMEM) Graph500 Design
  - 8,192 processes
    - 2.4X improvement over MPI-CSR
    - 7.6X improvement over MPI-Simple
  - 16,384 processes
    - 1.5X improvement over MPI-CSR
    - 13X improvement over MPI-Simple

**Sort Execution Time**

- Performance of Hybrid (MPI+OpenSHMEM) Sort Application
  - 4,096 processes, 4 TB Input Size
    - MPI – 2408 sec; 0.16 TB/min
    - Hybrid – 1172 sec; 0.36 TB/min
    - 51% improvement over MPI-design

---


J. Jose, S. Potluri, K. Tomko and D. K. Panda, Designing Scalable Graph500 Benchmark with Hybrid MPI+OpenSHMEM Programming Models, International Supercomputing Conference (ISC’13), June 2013

Hybrid MPI+UPC NAS-FT

- Modified NAS FT UPC all-to-all pattern using MPI_Alltoall
- Truly hybrid program
- For FT (Class C, 128 processes)
  - 34% improvement over UPC-GASNet
  - 30% improvement over UPC-OSU

Hybrid MPI + UPC Support
Available since MVAPICH2-X 1.9

J. Jose, M. Luo, S. Sur and D. K. Panda, Unifying UPC and MPI Runtimes: Experience with MVAPICH, Fourth Conference on Partitioned Global Address Space Programming Model (PGAS ’10), October 2010
MVAPICH2-X: FAQs

- Inadequate shared heap size
  - Set appropriate heap size

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Significance</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPC_SHARED_HEAP_SIZE</td>
<td>Set UPC shared heap size</td>
<td>64M</td>
</tr>
<tr>
<td>OOSHM_USE_SHARED_HEAP_SIZE</td>
<td>Set OpenSHMEM symmetric heap size</td>
<td>512M</td>
</tr>
</tbody>
</table>

- Can’t install mvapich2-x rpm in /opt
  - Use “—prefix” option when installing via rpm command

- Userguide
  - [http://mvapich.cse.ohio-state.edu/support/user_guide_mvapich2-x-2.0.html](http://mvapich.cse.ohio-state.edu/support/user_guide_mvapich2-x-2.0.html)
MVAPICH2-X: Looking forward

- Support for Accelerators and Co-processors
  - For UPC, OpenSHMEM and hybrid MPI+X (UPC/OpenSHMEM)

- Hybrid transport protocols for scalability

- Multi-end point runtime to improve network utilization

- Improving intra-node communication using CMA/LiMIC

- Optimizing collective communication in UPC/OpenSHMEM
Presentation Overview

- Runtime Optimization and Tuning Flexibility in MVAPICH2
- Runtime Optimization and Tuning Flexibility in MVAPICH2-GDR
- Runtime Optimization and Tuning Flexibility in MVAPICH2-X
- Runtime Optimization and Tuning Flexibility in MVAPICH2-MIC
- Advanced Optimization and Tuning of MPI Applications using MPI-T features in MVAPICH
- Overview of Installation, Configuration and Debugging Support
- Conclusions and Final Q&A
MPI Applications on MIC Clusters

• MPI (+X) continues to be the predominant programming model in HPC
• Flexibility in launching MPI jobs on clusters with Xeon Phi

Multi-core Centric

Host-only

Offload (/reverse Offload)

Symmetric

Coprocessor-only

Many-core Centric

Xeon

MPI Program

MPI Program

Offloaded Computation

Xeon Phi

MPI Program

MPI Program

MPI Program
11. Inter-Node MIC-MIC with IB on remote socket 
and more . . .

- Critical for runtimes to optimize data movement, hiding the complexity
Host Proxy-based Designs in MVAPICH2-MIC

- Direct IB channels is limited by P2P read bandwidth
- MVAPICH2-MIC uses a hybrid DirectIB + host proxy-based approach to work around this

S. Potluri, D. Bureddy, K. Hamidouche, A. Venkatesh, K. Kandalla, H. Subramoni and D. K. Panda, 
Getting Started with MVAPICH2-MIC

- Stampede: module swap mvapich2 mvapich2-mic/20130911
- Blueridge (Virginia Tech): module swap mvapich2 mvapich2-mic/1.9
- Beacon (UTK): module unload intel-mpi; module load mvapich2-mic/1.9

- USE the MPMD style to have symmetric execution
  - configfile:
    - -n 1: ./osu_latency-host
    - -n 1: ./osu_latency-mic
  - hfile:
    - host: 1
    - mic0: 1
  - Paramfile
    - MV2_IBA_HCA=mlx4_0 (add the runtime parameters)

- The configfile must match the hfile
- Important environment variables to be set
  - MV2_MIC_INSTALL_PATH=path to the MIC install

- How to launch
  - MV2_USER_CONFIG=./paramfile mpirun_rsh –hostfile hfile –config configfile –use-mic-proxy
Intra-node Runtime Parameters

- Applicable to Intra-node: Host-MIC, Intra-MIC and Inter-MIC

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Significance</th>
<th>Default</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MV2_USE_SCIF</td>
<td>• Enable / Disable SCIF designs</td>
<td>1 (Enabled)</td>
<td>• Always leave set to 1</td>
</tr>
<tr>
<td>MV2_MIC_MAPPING</td>
<td>• To bind process/thread to core</td>
<td>none</td>
<td>• To be used specially for hybrid MPI+Openmp code</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Processes are separated with colon “:” and threads with commas “,”</td>
</tr>
</tbody>
</table>

Intra-MIC Latency

- Graph showing latency (usec) vs. message size (Bytes) with and without SCIF.

Intra-MIC Bandwidth

- Graph showing bandwidth (MB/s) vs. message size (Bytes) with and without SCIF.
Inter-node Runtime Parameters

- Applicable to Inter-node: MIC-MIC, MIC-Host and Host-MIC

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Significance</th>
<th>Default</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MV2_MIC_PROXY_INTER_NODE_MODE</td>
<td>• 1hop or 2hops proxy design</td>
<td>1hop</td>
<td>• Enable proxy by adding –use-mic-proxy to the mpirun_rsh</td>
</tr>
<tr>
<td>MV2_MIC_MAPPING &amp;&amp; MV2_CPU_MAPPING</td>
<td>• To bind process/thread to core</td>
<td>none</td>
<td>• To be used specially for hybrid MPI+OpenMP code</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Processes are separated with colon “:” and threads with commas “,”</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Leave one free core on the host (near the IB socket)</td>
</tr>
</tbody>
</table>

Inter-node MIC-to-MIC Latency

- MV2-MIC w/o proxy

Inter-node MIC-to-MIC Bandwidth

- MV2-MIC w/Proxy
Presentation Overview

- Runtime Optimization and Tuning Flexibility in MVAPICH2
- Runtime Optimization and Tuning Flexibility in MVAPICH2-GDR
- Runtime Optimization and Tuning Flexibility in MVAPICH2-X
- Runtime Optimization and Tuning Flexibility in MVAPICH2-MIC

- Advanced Optimization and Tuning of MPI Applications using MPI-T features in MVAPICH
- Overview of Installation, Configuration and Debugging Support
- Conclusions and Final Q&A
MPI Tools Information Interface (MPI-T)

- Introduced in MPI 3.0 standard to expose internals of MPI to tools and applications
- Generalized interface – no defined variables in the standard
- Control Variables (CVARS) and Performance Variables (PVARS)
- More about the interface: mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf
MPI-T in MVAPICH2: Performance Variables (PVARS)

- Insights into performance of the MPI library
- Highly-implementation specific
- Memory consumption, timing information, resource-usage, data transmission info.
- Per-call basis or an entire MPI job

**Memory Usage:**
- current level
- maximum watermark

**InfiniBand N/W:**
- #control packets
- #out-of-order packets

**Pt-to-pt messages:**
- unexpected queue length
- unexp. match attempts
- recvq. length

**Registration cache:**
- hits
- misses

**Shared-memory:**
- limic/ CMA
- buffer pool size & usage

**Collective ops:**
- comm. creation
- #algorithm invocations
  [Bcast – 8; Gather – 10]
MPI-T workflows supported in MVAPICH2

• Several workflows based on role: End Users / Performance Tuners / MPI Implementers

• Two main workflows for Users and Tuners to improve application performance
  - Transparently using MPIT-Aware external tools
  - Co-designing applications and MPI-libraries using MPI-T

To use the MPI-T interface with MVAPICH2, add the following configure flag:

```bash
--enable-mpit-pvars=mv2
```
Workflow #1: Using External Tools - Capabilities

Simple CLI tool to query MPI-T capabilities of an MPI Implementation
Workflow #1: Using External Tools - Profiling

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Type</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>mem_allocated</td>
<td>ULL</td>
<td>5191069</td>
<td>5191297</td>
<td>5191097</td>
</tr>
<tr>
<td>mem_allocated</td>
<td>ULL</td>
<td>5191069</td>
<td>5191297</td>
<td>5191097</td>
</tr>
<tr>
<td>num_malloc_calls</td>
<td>UL</td>
<td>497</td>
<td>6547</td>
<td>1253</td>
</tr>
<tr>
<td>num_malloc_calls</td>
<td>UL</td>
<td>55</td>
<td>58</td>
<td>55</td>
</tr>
<tr>
<td>num_allocCalls</td>
<td>UL</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>num_malign_calls</td>
<td>UL</td>
<td>245</td>
<td>245</td>
<td>245</td>
</tr>
<tr>
<td>num_strdup_calls</td>
<td>UL</td>
<td>33</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>num_realloc_calls</td>
<td>UL</td>
<td>144</td>
<td>6194</td>
<td>900</td>
</tr>
<tr>
<td>num_free_calls</td>
<td>UL</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>num_malign_free_calls</td>
<td>UL</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mv2_num_2level_comm_requests</td>
<td>ULL</td>
<td>17680</td>
<td>17680</td>
<td>17680</td>
</tr>
<tr>
<td>mv2_num_2level_comm_success</td>
<td>ULL</td>
<td>704</td>
<td>198654</td>
<td>51858</td>
</tr>
<tr>
<td>mv2_num_shmem_coll_calls</td>
<td>UL</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mpit_progress_poll</td>
<td>ULL</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>rdma_ud_retransmissions</td>
<td>ULL</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mpit_bcast_mv2_binomial</td>
<td>ULL</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mpit_bcast_mv2_scatter_doubling_allgather</td>
<td>ULL</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mpit_bcast_mv2_scatter_ring_allgather</td>
<td>ULL</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

PMPI+MPIT-based CLI tool to profile job-level PVAR statistics
Workflow #1: Using External Tools - Tracing

A modified version of VampirTrace tool that is MPI-T-Aware
(http://tu-dresden.de/die_tu_dresden/zentrale_einrichtungen/zh/forschung/projekte/vampirtrace)
Workflow #2: Co-Design Applications

Example #1: Optimizing the eager limit dynamically

```c
MPI_T_init_thread()
MPI_T_cvar_get_info(MV2_EAGER_THRESHOLD)
if (msg_size < MV2_EAGER_THRESHOLD + 1KB) {
    MPI_T_cvar_write(MV2_EAGER_THRESHOLD, +1024)
}
MPI_Send(..)
MPI_T_finalize()
```

Example #2: Identifying unexpected message hotspots

```c
MPI_T_init_thread()
MPI_T_pvar_get_info(mv2_unexp_queue_len)
MPI_T_pvar_session_create()
MPI_T_pvar_start()
...
MPI_Isend(..)
MPI_T_pvar_read(mv2_unexp_queue_len)
print(mv2_unexp_queue_len)
...
MPI_T_pvar_session_free()
MPI_T_finalize()
```
Case Study: Reducing memory footprint with UH3D

- Load-imbalance in application
- Memory usage increases drastically with scale as more VBUFs are used
- MPI-T assisted analysis attributed memory usage to statically-established RC connections
- Optimal selection of UD threshold leads to **7.5X reduction in memory usage**
- Total execution time with tuned version was consistently equal to, or lesser than, default
Presentation Overview

- Runtime Optimization and Tuning Flexibility in MVAPICH2
- Runtime Optimization and Tuning Flexibility in MVAPICH2-GDR
- Runtime Optimization and Tuning Flexibility in MVAPICH2-MIC
- Runtime Optimization and Tuning Flexibility in MVAPICH2-X
- Advanced Optimization and Tuning of MPI Applications using MPI-T features in MVAPICH

- **Overview of Installation, Configuration and Debugging Support**
  - User Resources
  - Configuration File Support in MVAPICH2
  - Configuring MVAPICH2 to run on QLogic (PSM) Adapters
  - Run using TotalView Debugger Support
  - Run using a Profiling Library
  - Frequently reported issues and Common mistakes
  - Useful Diagnostics
  - Performance Troubleshooting
  - Getting help and Bug report details

- Conclusions and Final Q&A
User Resources

- **MVAPICH2 Quick Start Guide**
- **MVAPICH2 User Guide**
  - Long and very detailed
  - FAQ
- **MVAPICH2 Web-Site**
  - **Overview** and **Features**
  - **Reference performance**
  - **Publications**
- **Mailing List** Support
  - mvapich-discuss
- **Mailing List Archives**

All above resources accessible from: [http://mvapich.cse.ohio-state.edu/](http://mvapich.cse.ohio-state.edu/)
Configuration File Support in MVAPICH2

• Supports use of configuration values to ease repeatedly setting many different environment variables

• Configurations stored in a configuration file
  – Contains statements of the form “VARIABLE = VALUE”
  – Full line comments are supported and begin with the “#” character.

• System configuration file can be placed at /etc/mvapich2.conf

• User configuration files are located at “~/.mvapich2.conf” by default
  – Path to user configuration file can be specified at runtime by MV2_USER_CONFIG

• The processing of these files can be disabled by the use of the MV2_IGNORE_SYSTEM_CONFIG and MV2_IGNORE_USER_CONFIG.

• Refer to Configuration File Processing section of MVAPICH2 user guide for more information
• http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.0-userguide.html#x1-480006.3
Configuring MVAPICH2 to run on QLogic (PSM) Adapters

- QLogic PSM interface must be built to use MVAPICH2 on InfiniPath adapters
  - `./configure --with-device=ch3:psm`

- Configuration Options for QLogic PSM channel
  - Configuring with Shared Libraries
    - Default: Enabled (Disable: --disable-shared)
  - Path to QLogic PSM header files
    - Default: The build systems search path for header files
      - Specify: `--with-psm-include=path`
  - Path to QLogic PSM library
    - Default: The build systems search path for libraries
      - Specify: `--with-psm=path`
  - Support for 64K or greater number of cores
    - Default: 64K or lower number of cores
    - Enable: `--with-ch3-rank-bits=32`

- Refer to Configuring a build for QLogic PSM-CH3 section of MVAPICH2 user guide for more information
- [http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.0-userguide.html#x1-160004.8](http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.0-userguide.html#x1-160004.8)
Run using TotalView Debugger Support

• Configure MVAPICH2 with debug support
  
  ```
  ./configure <other options> --enable-g=dbg --enable-fast=none
  ```

• Compile your MPI application with debug symbols
  
  ```
  mpicc -g -o prog prog.c
  ```

• Define the correct path to TotalView as the TOTALVIEW variable
  
  ```
  export TOTALVIEW=<path_to_TotalView>
  ```

• Run your program with “-tv” option
  
  ```
  mpirun_rsh -tv -np 2 n0 n1 prog
  ```

• Troubleshooting:
  
  X authentication errors: check if you have enabled X Forwarding
  
  ```
  echo "ForwardX11 yes" >> $HOME/.ssh/config
  ```

  ssh authentication error
  
  ```
  ssh to the computer node with its long form hostname
  ```
  ```
  ssh i0.domain.osu.edu
  ```

• Refer to Run using TotalView Debugger Support section of MVAPICH2 user guide for more information
  
  ```
  http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.0-userguide.html#x1-400005.2.10
  ```
Run using a Profiling Library

• Applications built with MVAPICH2 can be profiled with several profiling libraries
• Use of two profiling libraries are described below, mpiP and Scalasca
• mpiP
  – Link your program with the required mpiP libraries
    • $ mpicc -g prg.o -o prg -L${mpiP_root}/lib -lmpiP -lm -lbfd -liberty –lunwind
    • MpiP has many features. Please refer to the mpiP web page for more details.
• Scalasca
  – Configure Scalasca by supplying the ‘--mpi=mpich2’ option
    • ./configure --mpi=mpich2
    • Once the installation is done, you will be able to use Scalasca with MVAPICH2
    • Refer to Run using a Profiling Library section of MVAPICH2 user guide for more information
    • [http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.0-userguide.html#x1-410005.2.11](http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.0-userguide.html#x1-410005.2.11)
Frequently reported issues and Common mistakes

- Job Startup issues
- MPI_Init and Other MPI errors
- Creation of CQ or QP failure
- Failed to register memory with Infiniband HCA
- Multicast group creation failed
- InfiniBand setup issues
- MVAPICH2 over RoCE issues
- MPI + OpenMP, Multi-threaded MPI shows bad performance
- MVAPICH2 + CUDA + PGI doesn't build
Job Startup issues

- **Symptoms**
  - [mpirun_rsh][child_handler] Error in init phase, aborting! (0/2 mpispawn connections)

- **Cause**
  - Host file is not correct
  - SSH issues

- **Troubleshooting**
  - Verify host file
  - Password less ssh
  - DNS or /etc/hosts
MPI_Init and Other MPI errors

• **Symptoms**
  • “Fatal error in MPI_Init:
    Other MPI error”

• **Cause**
  • Could be because of multiple reasons

• **Troubleshooting**
  • Reconfigure with –enable-g=dbg –enable fast=none to better understand the problem

    [cli_0]: aborting job:
    Fatal error in MPI_Init:
    Other MPI error, error stack:
    MPIR_Init_thread(408):......:
    MPID_Init(308):............: channel initialization failed
    MPIDI_CH3_Init(283):........:
    MPIDI_CH3I_RDMA_init(171):...:
    rdma_setup_startup_ring(389): cannot open hca device
Creation of CQ or QP failure

• **Symptoms**
  - `libibverbs: Warning: RLIMIT_MEMLOCK is 32768 bytes.`
    This will severely limit memory registrations.
  Other MPI error, error stack:
  MPIR_Init_thread(449)........:
  MPID_Init(365)..............: channel initialization failed
  MPIDI_CH3_Init(313)........:
  MPIDI_CH3I_RDMA_init(170)...:
  rdma_setup_startup_ring(416): cannot create cq

• **Cause**
  - Memory buffers used in verbs operations and ib context uses pinned memory
  - Inability to pin the required memory

• **Troubleshooting**
  - Make sure enough memory set for “max locked memory” (limit –l)
  - Recommended “unlimited” on all compute nodes

• Refer to **Creation of CQ or QP failure** section of MVAPICH2 user guide for more information

  • [http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.0-userguide.html#x1-1330009.4.3](http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.0-userguide.html#x1-1330009.4.3)
Failed to register memory with InfiniBand HCA

- **Symptoms**
  - “Cannot register vbuf region”
  - “Abort: vbuf pool allocation failed”
  - QP errors, node failures

- **Cause**
  - Limited registered (pinned) memory

- **Troubleshooting**
  - OFED parameters: \( \text{log\_num\_mtt}, \text{log\_mtts\_per\_seg} \)
  - \( \text{max\_reg\_mem} = (2^{\text{log\_num\_mtt}}) \times (2^{\text{log\_mtts\_per\_seg}}) \times \text{PAGE\_SIZE} \)
  - Some OFED default values are too low (< 2GB)
  - clusters with large physical memory (> 64)
  - **Recommendation**: increase \( \text{log\_num\_mtt} \) value
    - \( \text{max\_reg\_mem} = (2^{24}) \times (2^1) \times (4 \text{ kB}) = 128 \text{ GB} \)

- Refer to MVAPICH2 failed to register memory with InfiniBand HCA section of MVAPICH2 user guide
- [http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.0-userguide.html#x1-1100009.1.1](http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.0-userguide.html#x1-1100009.1.1)
Multicast group creation failed

• **Symptoms**
  • [host1:mpi_rank_0][create_2level_comm]
    Warning: Multicast group setup failed. Not using any multicast features

• **Cause**
  • Umad device permission
  • OpenSM issues

• **Troubleshooting**
  • Check umad device user permissions
    $ ls -l /dev/infiniband/umad0
    crw-rw-rw- 1 root root 231, 0 Aug 9 02:04 /dev/infiniband/umad0
  • Slow opensm response
    ○ MV2_MCAST_COMM_INIT_TIMEOUT
  • Maximum multicast groups reached (very unlikely). Check opensm logs

• Refer to Running Collectives with Hardware based Multicast support section of MVAPICH2 user guide
• [http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.0-userguide.html#x1-590006.8](http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.0-userguide.html#x1-590006.8)
InfiniBand Setup Issues

• **Symptoms**
  - [0->6150] send desc error, wc_opcode=0
  - [0->6150] `wc.status=12`, wc.opcode=0, vbuf->phead->type=25 = XXXX
  - [4979] Abort: [] Got completion with error 12, vendor code=0x81, **dest rank=6150**
  - wc.status : 12 (**IBV_WC_RETRY_EXC_ERR**), 13 (**IBV_WC_RNR_RETRY_EXC_ERR**)

• **Cause**
  - Bad QP attributes
  - Loose cable, bad HCA or a bad switch blade
  - Remote side is in a bad state
  - Heavy congestion in the network

• **Troubleshooting**
  - **MV2_DEFAULT_RETRY_COUNT**
  - Map src, dest ranks to host file and check those specific nodes
Issues with Running MVAPICH2 over RoCE

- **Symptoms**
  - Intermittent hangs

- **Cause**
  - Most likely setup issues

- **Troubleshooting**
  - Requires loss-less Ethernet fabric
  - Configure Ethernet switch to treat RoCE traffic as loss-less
  - Create a separate VLAN interface
  - All VLAN interfaces appear as additional GID index
  - Select non-default GID index with MV2_DEFAULT_GID_INDEX
  - Use VLAN IP addresses in `/etc/mv2.conf` in RDMA CM mode

- Refer to Run with mpirun_rsh using OFA-RoCE Interface section of MVAPICH2 user guide
- [http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.0-userguide.html#x1-1390009.5](http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.0-userguide.html#x1-1390009.5)
MPI + OpenMP, Multi-threaded MPI shows bad performance

- **Symptoms**
  
  Poor performance, hangs

- **Cause**
  
  CPU affinity enabled by default
  
  All OpenMP, pthreads in the application process bind to same core

- **Troubleshooting**
  
  - Turn off affinity
    
    MV2_ENABLE_AFFINITY = 0
  
  - Choose binding level
    
    MV2_CPU_BINDING_LEVEL=socket

- Refer to [MPI+OpenMP shows bad performance section of MVAPICH2 user guide](http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.0-userguide.html#x1-1130009.1.4)
MVAPICH2 + CUDA + PGI doesn't build

• **Symptoms**
  • nvlink fatal : Unsupported file type ‘../lib/.libs/libmpich.so’

• **Cause**
  • Recent PGI compilers are using a new linkage convention

• **Troubleshooting**
  • add “-ta=tesla:nordc” to your CFLAGS
  • This should be fixed in the next CUDA release
Useful Diagnostics

- What parameters are being used by my job?
- Where is the segmentation fault?
- What is the peak memory used by my app?
- Is process binding working as expected?
What parameters are being used by my job?

- **MV2_SHOW_ENV_INFO**
  - Show values of the run time parameters
  - 1 (short list), 2 (full list)

- **Example**

  ```sh
  $ mpirun_rsh -np 2 --hostfile hfile MV2_SHOW_ENV_INFO=1 ./exec
  ```

MVAPICH2-2.0 Parameters

---

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROCESSOR ARCH NAME</td>
<td>MV2_ARCH_INTEL_XEON_E5_2680_16</td>
</tr>
<tr>
<td>HCA NAME</td>
<td>MV2_HCA_MLX_CX_FDR</td>
</tr>
<tr>
<td>HETEROGENEOUS</td>
<td>NO</td>
</tr>
<tr>
<td>MV2_VBUF_TOTAL_SIZE</td>
<td>17408</td>
</tr>
<tr>
<td>MV2_IBA_EAGER_THRESHOLD</td>
<td>17408</td>
</tr>
<tr>
<td>MV2_RDMA_FAST_PATH_BUF_SIZE</td>
<td>5120</td>
</tr>
<tr>
<td>MV2_EAGERSIZE_1SC</td>
<td>8192</td>
</tr>
<tr>
<td>MV2_PUT_FALLBACK_THRESHOLD</td>
<td>8192</td>
</tr>
<tr>
<td>MV2_GET_FALLBACK_THRESHOLD</td>
<td>0</td>
</tr>
<tr>
<td>MV2_SMP_EAGERSIZE</td>
<td>8193</td>
</tr>
<tr>
<td>MV2_SMPI_LENGTH_QUEUE</td>
<td>524288</td>
</tr>
<tr>
<td>MV2_SMP_NUM_SEND_BUFFER</td>
<td>16</td>
</tr>
<tr>
<td>MV2_SMP_BATCH_SIZE</td>
<td>8</td>
</tr>
</tbody>
</table>
---

MUG'14
Where is the segmentation fault?

- **MV2_DEBUG_SHOW_BACKTRACE**
  - Shows backtrace with debug builds (--enable-g=dbg, --enable-fast=none)

- **Example**

- segmentation fault report with out much information
  
  [host1:mpi_rank_0][error_sighandler] Caught error: Segmentation fault (signal 11)

- mpirun_rsh -np 2 –hostfile hfile **MV2_DEBUG_SHOW_BACKTRACE=1** ./exec
  
  [error_sighandler] Caught error: Segmentation fault (signal 11)

  [print_backtrace] 0: libmpich.so.10(print_backtrace+0x22) [0x2af447e29d9a]
  [print_backtrace] 1: libmpich.so.10(error_sighandler+0x7c) [0x2af447e29ef2]
  [print_backtrace] 2: libmpich.so.10(allocate_vbufs+0x71) [0x2af447de6d9f]
  [print_backtrace] 3: libmpich.so.10(rdma_iba_allocate_memory+0x101) [0x2af447dd5ca2]
  [print_backtrace] 4: libmpich.so.10(MPIDI_CH3I_RDMA_init+0x1569) [0x2af447dce9f1]
  [print_backtrace] 5: libmpich.so.10(MPIDI_CH3_Init+0x406) [0x2af447da32f4]
  [print_backtrace] 6: libmpich.so.10(MPID_Init+0x31f) [0x2af447d8a91b]
  [print_backtrace] 7: libmpich.so.10(MPIR_Init_thread+0x3e0) [0x2af447f90aca]
  [print_backtrace] 8: libmpich.so.10(MPI_Init+0x1de) [0x2af447f8f645]
  [print_backtrace] 9: ./mpi_hello() [0x400746]
What is the peak memory used by my app?

- **MV2_DEBUG_MEM_USAGE_VERBOSE**
  - Show memory usage statistics
  - 1 (rank 0 usage), 2 (all ranks)

- **Example**

  ```bash
  $ mpirun_rsh -np 2 -hostfile hfile MV2_DEBUG_MEM_USAGE_VERBOSE=1 ./exec
  [mv2_print_mem_usage] VmPeak: 79508 kB VmHWM: 16340 kB
  [mv2_print_vbuf_usage] RC VBUFs:512 UD VBUFs:0 TOT MEM:8828 kB
  ```
Is process binding working as expected?

- **MV2_SHOW_CPU_BINDING**
  - Display CPU binding information
  - Launcher independent

**Examples**

- **MV2_SHOW_CPU_BINDING=1 MV2_CPU_BINDING_POLICY=scatter**
  
  ---------------CPU AFFINITY--------------
  RANK:0 CPU_SET: 0  
  RANK:1 CPU_SET: 8  

- **MV2_SHOW_CPU_BINDING=1 MV2_CPU_BINDING_POLICY=core**

  ---------------CPU AFFINITY--------------
  RANK:0 CPU_SET: 0  
  RANK:1 CPU_SET: 1  

- **MV2_SHOW_CPU_BINDING=1 MV2_CPU_BINDING_POLICY=scatter MV2_CPU_BINDING_LEVEL=socket**

  ---------------CPU AFFINITY--------------
  RANK:0 CPU_SET: 0 1 2 3 4 5 6 7  
  RANK:1 CPU_SET: 8 9 10 11 12 13 14 15  

- **MV2_SHOW_CPU_BINDING=1 MV2_CPU_BINDING_POLICY=bunch MV2_CPU_BINDING_LEVEL=socket**

  ---------------CPU AFFINITY--------------
  RANK:0 CPU_SET: 0 1 2 3 4 5 6 7  
  RANK:1 CPU_SET: 0 1 2 3 4 5 6 7  

---

169
Performance Troubleshooting

- Check “active_speed” in “ibv_devinfo –v” output
- Check OFED memory registration limits (log_num_mtt, log_mtt_per_seg)
- Increase registration cache size
  - MV2_NDREG_ENTRIES, MV2_NDREG_ENTRIES_MAX
- Are huge pages configured?
- SMP copy block size: MV2_SMP_SEND_BUF_SIZE
- Small message performance
  - RDMA fast path thresholds
    - MV2_NUM_RDMA_BUFFER, MV2_RDMA_FAST_PATH_BUF_SIZE
  - Eager thresholds
    - MV2_IBA_EAGER_THRESHOLD, MV2_VBUF_TOTAL_SIZE
- Large message performance
  - RNDV protocols: MV2_RNDV_PROTOCOL
- Collectives
  - Try different algorithms, change algorithm specific parameters
Troubleshooting Functionality Issues

- Check the MVAPICH2 FAQ
- Check the Mailing List Archives
- Basic System Diagnostics
  - `ibv_devinfo` - at least one port should be PORT_ACTIVE
  - `ulimit -l` - should be “unlimited” on all compute nodes
  - host resolution: DNS or `/etc/hosts`
  - password-less ssh login
  - run IB perf tests for all the message sizes(-a option)
    - `Ib_send_lat`, `ib_send_bw`
  - run system program (like hostname) and MPI hello world program
Getting Help (contd.)

- More diagnostics
  - Already fixed issue: always try with latest release
  - Regression: verifying with previous release
  - Application issue: verify with other MPI libraries
  - Launcher issue: verifying with multiple launchers (mpirun_rsh, mpiexec.hydra)
  - Debug mode
  - Compiler optimization issues: try with different compiler
Submitting Bug Report

- Subscribe to mvapich-discuss and send problem report
- Include as much information as possible
- Run-time issues
  - Config flags ("mpiname -a" output)
  - Exact command used to run the application
  - Run-rime parameters in the environment
  - Standalone reproducer program
  - Information about the IB network
    - OFED version
    - ibv_devinfo
  - Remote system access
Submitting Bug Report (contd.)

- Build and Installation issues
  - MVAPICH2 version
  - Compiler version
  - Platform details (OS, kernel version..etc)
  - Configure flags
  - Attach Config.log file
  - Attach configure, make and make install step output
    - ./configure {–flags} 2>&1 | tee config.out
    - Make 2>&1 | tee make.out
    - Make install 2>&1 | tee install.out
Presentation Overview

- Runtime Optimization and Tuning Flexibility in MVAPICH2
- Runtime Optimization and Tuning Flexibility in MVAPICH2-GDR
- Runtime Optimization and Tuning Flexibility in MVAPICH2-X
- Runtime Optimization and Tuning Flexibility in MVAPICH2-MIC
- Advanced Optimization and Tuning of MPI Applications using MPI-T features in MVAPICH
- Overview of Installation, Configuration and Debugging Support
- **Conclusions and Final Q&A**
MVAPICH2 – Plans for Exascale

• Performance and Memory scalability toward 500K-1M cores
  – Dynamically Connected Transport (DCT) service with Connect-IB
• Hybrid programming (MPI + OpenSHMEM, MPI + UPC, MPI + CAF ...)
• Enhanced Optimization for Runtime Optimization and Tuning Flexibility in MVAPICH2-GDR and Accelerators
• Taking advantage of Collective Offload framework
  – Including support for non-blocking collectives (MPI 3.0)
  – Core-Direct support
• Extended RMA support (as in MPI 3.0)
• Extended topology-aware collectives
• Power-aware collectives
• Extended Support for MPI Tools Interface (as in MPI 3.0)
• Extended Checkpoint-Restart and migration support with SCR
Concluding Remarks

- Provided an overview of MVAPICH2, MVAPICH2-GDR, MVAPICH2-X and MVAPICH2-MIC libraries
- Presented in-depth details on configuration and runtime parameters, optimizations and their impacts
- Provided an overview of debugging support
- Demonstrated how MPI and PGAS users can use these optimization techniques to extract performance and scalability while using MVAPICH2, MVAPICH2-GDR, MVAPICH2-X and MVAPICH2-MIC
Funding Acknowledgments

**Funding Support by**

- Office of Science
- NSF
- Mellanox Technologies
- Cisco Systems
- QLogic
- Cray
- Intel
- Linux Network
- NVIDIA
- NetApp
- Sun Microsystems

**Equipment Support by**

- Mellanox Technologies
- Intel
- AMD
- Sun Microsystems
- AFPR

advanced clustering technologies, inc.

Microway

NVIDIA

QLogic
Personnel Acknowledgments

**Current Students**
- S. Chakraborthy (Ph.D.)
- N. Islam (Ph.D.)
- M. Li (Ph.D.)
- R. Rajachandrasekhar (Ph.D.)
- M. Rahman (Ph.D.)
- D. Shankar (Ph.D.)
- R. Shir (Ph.D.)
- A. Venkatesh (Ph.D.)
- J. Zhang (Ph.D.)

**Past Students**
- P. Balaji (Ph.D.)
- D. Buntinas (Ph.D.)
- S. Bhagvat (M.S.)
- 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.)
- W. Jiang (M.S.)
- J. Jose (Ph.D.)
- S. Kini (M.S.)
- M. Koop (Ph.D.)
- R. Kumar (M.S.)
- S. Krishnamoorthy (M.S.)
- K. Kandalla (Ph.D.)
- P. Lai (M.S.)
- J. Liu (Ph.D.)

**Past Post-Docs**
- H. Wang
- X. Besseron
- H.-W. Jin
- M. Luo

**Past Research Scientist**
- S. Sur

**Past Programmers**
- D. Bureddy

**Past Researchers**
- S. Sur

**Current Senior Research Associates**
- X. Lu
- H. Subramoni

**Current Post-Docs**
- K. Hamidouche
- J. Lin

**Current Programmers**
- M. Arnold
- J. Perkins

**Current Senior Research Associates**
- M. Luo (Ph.D.)
- A. Mamidala (Ph.D.)
- G. Marsh (M.S.)
- V. Meshram (M.S.)
- S. Naravula (Ph.D.)
- R. Noronha (Ph.D.)
- X. Ouyang (Ph.D.)
- S. Pai (M.S.)
- S. Potluri (Ph.D.)

**Past Post-Docs**
- W. Huang (Ph.D.)
- W. Jiang (M.S.)
- J. Jose (Ph.D.)
- S. Kini (M.S.)
- M. Koop (Ph.D.)
- R. Kumar (M.S.)
- S. Krishnamoorthy (M.S.)
- K. Kandalla (Ph.D.)
- P. Lai (M.S.)
- J. Liu (Ph.D.)

**Past Researchers**
- S. Sur

**Past Programmers**
- D. Bureddy
Web Pointers

http://www.cse.ohio-state.edu/~panda
http://nowlab.cse.ohio-state.edu

MVAPICH Web Page
http://mvapich.cse.ohio-state.edu

panda@cse.ohio-state.edu