How to Boost the Performance of Your MPI and PGAS Applications with MVAPICH2 Libraries

A Tutorial at SEA Symposium ’18

by

Dhabaleswar K. (DK) Panda
The Ohio State University
E-mail: panda@cse.ohio-state.edu
http://www.cse.ohio-state.edu/~panda

Hari Subramoni
The Ohio State University
E-mail: subramon@cse.ohio-state.edu
http://www.cse.ohio-state.edu/~subramon

Latest version of the slides can be obtained from
http://www.cse.ohio-state.edu/~panda/sea18-mpi.pdf
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

<table>
<thead>
<tr>
<th>Application Kernels/Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Middleware</td>
</tr>
<tr>
<td><strong>Programming Models</strong></td>
</tr>
<tr>
<td>MPI, PGAS (UPC, Global Arrays, OpenSHMEM), CUDA, OpenMP, OpenACC, Cilk, Hadoop (MapReduce), Spark (RDD, DAG), etc.</td>
</tr>
</tbody>
</table>

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

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

**Multi-/Many-core Architectures**

**Accelerators (GPU and MIC)**

Co-Design Opportunities and Challenges across Various Layers
- Performance
- Scalability
- Resilience
Designing (MPI+X) for Exascale

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

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

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

• Support for efficient multi-threading

• Integrated Support for GPGPUs and Accelerators

• Fault-tolerance/resiliency

• QoS support for communication and I/O

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

• Virtualization

• Energy-Awareness
Overview of the MVAPICH2 Project

• High Performance open-source MPI Library for InfiniBand, Omni-Path, Ethernet/iWARP, and RDMA over Converged Ethernet (RoCE)
  – MVAPICH (MPI-1), MVAPICH2 (MPI-2.2 and MPI-3.1), Started in 2001, First version available in 2002
  – MVAPICH2-X (MPI + PGAS), Available since 2011
  – Support for GPGPUs (MVAPICH2-GDR) and MIC (MVAPICH2-MIC), Available since 2014
  – Support for Virtualization (MVAPICH2-Virt), Available since 2015
  – Support for Energy-Awareness (MVAPICH2-EA), Available since 2015
  – Support for InfiniBand Network Analysis and Monitoring (OSU INAM) since 2015

• Used by more than 2,875 organizations in 86 countries

• More than 461,000 (> 0.46 million) downloads from the OSU site directly

• Empowering many TOP500 clusters (Nov ‘17 ranking)
  • 1st, 10,649,600-core (Sunway TaihuLight) at National Supercomputing Center in Wuxi, China
  • 9th, 556,104 cores (Oakforest-PACS) in Japan
  • 12th, 368,928-core (Stampede2) at TACC
  • 17th, 241,108-core (Pleiades) at NASA
  • 48th, 76,032-core (Tsubame 2.5) at Tokyo Institute of Technology

• Available with software stacks of many vendors and Linux Distros (RedHat and SuSE)
  • http://mvapich.cse.ohio-state.edu

• Empowering Top500 systems for over a decade
Architecture of MVAPICH2 Software Family

High Performance Parallel Programming Models

- **Message Passing Interface (MPI)**
- **PGAS** (UPC, OpenSHMEM, CAF, UPC++)
- **Hybrid --- MPI + X** (MPI + PGAS + OpenMP/Cilk)

High Performance and Scalable Communication Runtime

**Diverse APIs and Mechanisms**

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

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

- **Transport Protocols**
  - RC
  - XRC
  - UD
  - DC

- **Modern Features**
  - UMR
  - ODP
  - SR-IOV
  - Multi Rail

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

- **Transport Mechanisms**
  - Shared Memory
  - CMA
  - IVSHMEM
  - XPMEM*

- **Modern Features**
  - MCDRAM*
  - NVLink*
  - CAPI*

* Upcoming
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 Software Family

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MPI with IB, iWARP and RoCE</strong></td>
<td>MVAPICH2</td>
</tr>
<tr>
<td>Advanced MPI, OSU INAM, PGAS and MPI+PGAS with IB and RoCE</td>
<td>MVAPICH2-X</td>
</tr>
<tr>
<td><strong>MPI with IB &amp; GPU</strong></td>
<td>MVAPICH2-GDR</td>
</tr>
<tr>
<td>Energy-aware MPI with IB, iWARP and RoCE</td>
<td>MVAPICH2-EA</td>
</tr>
<tr>
<td><strong>MPI Energy Monitoring Tool</strong></td>
<td>OEMT</td>
</tr>
<tr>
<td>InfiniBand Network Analysis and Monitoring</td>
<td>OSU INAM</td>
</tr>
<tr>
<td>Microbenchmarks for Measuring MPI and PGAS Performance</td>
<td>OMB</td>
</tr>
</tbody>
</table>
MVAPICH2 2.3rc1

- Released on 02/19/2018
- Major Features and Enhancements
  - Enhanced performance for Allreduce, Reduce_scatter_block, Allgather, Allgatherv through new algorithms
  - Enhance support for MPI_T PVARs and CVARs
  - Improved job startup time for OFA-IB-CH3, PSM-CH3, and PSM2-CH3
  - Support to automatically detect IP address of IB/RoCE interfaces when RDMA_CM is enabled without relying on mv2.conf file
  - Enhance HCA detection to handle cases where node has both IB and RoCE HCAs
  - Automatically detect and use maximum supported MTU by the HCA
  - Added logic to detect heterogeneous CPU/HFI configurations in PSM-CH3 and PSM2-CH3 channels
  - Enhanced intra-node and inter-node tuning for PSM-CH3 and PSM2-CH3 channels
  - Enhanced HFI selection logic for systems with multiple Omni-Path HFIs
  - Enhanced tuning and architecture detection for OpenPOWER, Intel Skylake and Cavium ARM (ThunderX) systems
  - Added 'SPREAD', 'BUNCH', and 'SCATTER' binding options for hybrid CPU binding policy
  - Rename MV2_THREADS_BINDING_POLICY to MV2_HYBRID_BINDING_POLICY
  - Added support for MV2_SHOW_CPU_BINDING to display number of OMP threads
  - Update to hwloc version 1.11.9
Presentation Overview

- Job start-up
- Point-to-point Inter-node Protocol
- Transport Type Selection
- Multi-rail
- Process Mapping and Point-to-point Intra-node Protocols
- Collectives
- MPI_T Support
Towards High Performance and Scalable Startup at Exascale

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

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


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

New designs available in MVAPICH2-2.3a and as patch for SLURM-15.08.8 and SLURM-16.05.1
On-demand Connection Management for OpenSHMEM+MPI

<table>
<thead>
<tr>
<th>Breakdown of OpenSHMEM Startup</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Number of Processes</strong></td>
</tr>
<tr>
<td>32</td>
</tr>
<tr>
<td><strong>Time Taken (Seconds)</strong></td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>Connection Setup</td>
</tr>
<tr>
<td>PMI Exchange</td>
</tr>
<tr>
<td>Memory Registration</td>
</tr>
<tr>
<td>Shared Memory Setup</td>
</tr>
<tr>
<td>Other</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Performance of OpenSHMEM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Number of Processes</strong></td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td><strong>Time Taken (Seconds)</strong></td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>Hello World - Static</td>
</tr>
<tr>
<td>Initialization - Static</td>
</tr>
<tr>
<td>Hello World - On-demand</td>
</tr>
<tr>
<td>Initialization - On-demand</td>
</tr>
</tbody>
</table>

- Static connection establishment wastes memory and takes a lot of time
- On-demand connection management improves OpenSHMEM initialization time by **29.6 times**
- Time taken for Hello World reduced by **8.31 times** at 8,192 processes
- Available since MVAPICH2-X 2.1rc1
How to Get the Best Startup Performance with MVAPICH2?

• **MV2_HOMOGENEOUS_CLUSTER=1**  //Set for homogenous clusters
• **MV2_ON_DEMAND_UD_INFO_EXCHANGE=1**  //Enable UD based address exchange

Using SLURM as launcher

• Use PMI2
  – ./configure --with-pm=slurm --with-pmi=pmi2
  – srun --mpi=pmi2 ./a.out

• Use PMI Extensions
  – Patch for SLURM available at [http://mvapich.cse.ohio-state.edu/download/](http://mvapich.cse.ohio-state.edu/download/)
  – Patches available for SLURM 15, 16, and 17
  – PMI Extensions are automatically detected by MVAPICH2

Using mpirun_rsh as launcher

• **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
Presentation Overview

- Job start-up
- **Point-to-point Inter-node Protocol**
- Transport Type Selection
- Multi-rail
- Process Mapping and Point-to-point Intra-node Protocols
- Collectives
- MPI_T Support
Inter-node Point-to-Point Tuning: Eager Thresholds

- 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 -hostfile hostfile MV2_IBA_EAGER_THRESHOLD=32K a.out
    - Memory footprint can increase along with eager threshold
Analyzing Overlap Potential of Eager Protocol

- Application processes schedule communication operation
- Network adapter progresses communication in the background
- Application process free to perform useful compute in the foreground
- Overlap of computation and communication => Better Overall Application Performance
- Increased buffer requirement
- Poor communication performance if used for all types of communication operations

Impact of changing Eager Threshold on performance of multi-pair message-rate benchmark with 32 processes on Stampede
Analyzing Overlap Potential of Rendezvous Protocol

- Application processes schedule communication operation
- Application process free to perform useful compute in the foreground
- Little communication progress in the background
- All communication takes place at final synchronization

- Reduced buffer requirement
- Good communication performance if used for large message sizes and operations where communication library is progressed frequently

- Poor overlap of computation and communication => Poor Overall Application Performance
Dynamic and Adaptive MPI Point-to-point Communication Protocols

### Desired Eager Threshold

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

### Eager Threshold for Example Communication Pattern with Different Designs

- **Default**
  - 0 – 4: 32 KB
  - 1 – 5: 64 KB
  - 2 – 6: 128 KB
  - 3 – 7: 32 KB

- **Manually Tuned**
  - 0 – 4: 16 KB
  - 1 – 5: 16 KB
  - 2 – 6: 16 KB
  - 3 – 7: 128 KB

- **Dynamic + Adaptive**
  - 0 – 4: 32 KB
  - 1 – 5: 64 KB
  - 2 – 6: 128 KB
  - 3 – 7: 32 KB

### Design Metrics: Overlap & Memory Requirement
- **Default**
  - Poor overlap; Low memory requirement
- **Manually Tuned**
  - Good overlap; High memory requirement
- **Dynamic + Adaptive**
  - Good overlap; Optimal memory requirement

### Metrics: Performance & Productivity
- **Default**
  - Low Performance; High Productivity
- **Manually Tuned**
  - High Performance; Low Productivity
- **Dynamic + Adaptive**
  - High Performance; High Productivity

---

Dynamic and Adaptive Tag Matching

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

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

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

---

Intra-node Point-to-point Performance on ARMv8

Platform: ARMv8 (aarch64) MIPS processor with 96 cores dual-socket CPU. Each socket contains 48 cores.
Intra-node Point-to-Point Performance on OpenPower

**Intra-Socket Small Message Latency**

- MVAPICH2-2.3rc1
- SpectrumMPI-10.1.0.2
- OpenMPI-3.0.0

**Intra-Socket Large Message Latency**

- MVAPICH2-2.3rc1
- SpectrumMPI-10.1.0.2
- OpenMPI-3.0.0

**Intra-Socket Bandwidth**

- MVAPICH2-2.3rc1
- SpectrumMPI-10.1.0.2
- OpenMPI-3.0.0

**Intra-Socket Bi-directional Bandwidth**

- MVAPICH2-2.3rc1
- SpectrumMPI-10.1.0.2
- OpenMPI-3.0.0

**Platform:** Two nodes of OpenPOWER (Power8-ppc64le) CPU using Mellanox EDR (MT4115) HCA
Inter-node Point-to-Point Performance on OpenPower

**Small Message Latency**

- MVAPICH2-2.3rc1
- SpectrumMPI-10.1.0.2
- OpenMPI-3.0.0

**Large Message Latency**

- MVAPICH2-2.3rc1
- SpectrumMPI-10.1.0.2
- OpenMPI-3.0.0

**Bandwidth**

- MVAPICH2-2.3rc1
- SpectrumMPI-10.1.0.2
- OpenMPI-3.0.0

**Bi-directional Bandwidth**

- MVAPICH2-2.3rc1
- SpectrumMPI-10.1.0.2
- OpenMPI-3.0.0

**Platform:** Two nodes of OpenPOWER (Power8-ppc64le) CPU using Mellanox EDR (MT4115) HCA
Presentation Overview

- Job start-up
- Point-to-point Inter-node Protocol
- **Transport Type Selection**
  - Multi-rail
  - Process Mapping and Point-to-point Intra-node Protocols
- Collectives
- MPI_T Support
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

<table>
<thead>
<tr>
<th>Number of Processes</th>
<th>Time (us)</th>
<th>UD</th>
<th>Hybrid</th>
<th>RC</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>26%</td>
<td>40%</td>
<td>38%</td>
<td>30%</td>
</tr>
<tr>
<td>256</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>512</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1024</td>
<td></td>
<td></td>
<td></td>
<td></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 • Limits the amount of connection memory</td>
<td>64</td>
<td>Prevents HCAQP cache thrashing</td>
</tr>
</tbody>
</table>

- 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/mvapich2-2.3a-userguide.html#x1-690006.11](http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.3a-userguide.html#x1-690006.11)
Minimizing Memory Footprint by Direct Connect (DC) Transport

- Constant connection cost (One QP for any peer)
- Full Feature Set (RDMA, Atomics etc)
- Separate objects for send (DC Initiator) and receive (DC Target)
  - DC Target identified by “DCT Number”
  - Messages routed with (DCT Number, LID)
  - Requires same “DC Key” to enable communication
- Available since MVAPICH2-X 2.2a

Presentation Overview

- Job start-up
- Point-to-point Inter-node Protocol
- Transport Type Selection
- Multi-rail
- Process Mapping and Point-to-point Intra-node Protocols
- Collectives
- MPI_T Support
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

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>Rail Binding in Round Robin mode</td>
<td>• Advanced tuning can result in better performance</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/mvapich2-2.3a-userguide.html#x1-700006.12](http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.3a-userguide.html#x1-700006.12)
Presentation Overview

- Job start-up
- Point-to-point Inter-node Protocol
- Transport Type Selection
- Multi-rail
  - Process Mapping and Point-to-point Intra-node Protocols
- Collectives
- MPI_T Support
Process Mapping support in MVAPICH2

- 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**
  - \( \text{MV2\_CPU\_BINDING\_POLICY} = \text{scatter} \)

- **“Socket/Numanode” level “Scatter” mapping**
  - \( \text{MV2\_CPU\_BINDING\_LEVEL} = \text{socket} \) \( \text{MV2\_CPU\_BINDING\_POLICY} = \text{scatter} \)
Process and thread binding policies in hybrid MPI+Threads

• A new process binding policy – “hybrid”
  – MV2_CPU_BINDING_POLICY = hybrid

• A new environment variable for co-locating Threads with MPI Processes
  – MV2_THREADS_PER_PROCESS = \( k \)
  – Automatically set to OMP_NUM_THREADS if OpenMP is being used
  – Provides a hint to the MPI runtime to spare resources for application threads.

• New variable for threads bindings with respect to parent process and architecture
  – MV2_THREADS_BINDING_POLICY = \{linear | compact\}
    • Linear – binds MPI ranks and OpenMP threads sequentially (one after the other)
      – Recommended to be used on non-hyper threaded systems with MPI+OpenMP
    • Compact – binds MPI rank to physical-core and locates respective OpenMP threads on hardware threads
      – Recommended to be used on multi-/many-cores e.g., KNL, POWER8, and hyper-threaded Xeon, etc.
**Binding Example in Hybrid (MPI+Threads)**

- MPI Processes = 4, OpenMP Threads per Process = 4
- MV2_CPU_BINDING_POLICY = hybrid
- MV2_THREADS_PER_PROCESS = 4
- MV2_THREADS_BINDING_POLICY = compact

- Detects hardware-threads support in architecture
- Assigns MPI ranks to physical cores and respective OpenMP Threads to HW threads
Binding Example in Hybrid (MPI+Threads) ---- Cont’d

- MPI Processes = 4, OpenMP Threads per Process = 4
- MV2_CPU_BINDING_POLICY = hybrid
- MV2_THREADS_PER_PROCESS = 4
- MV2_THREADS_BINDING_POLICY = linear

- MPI Rank-0 with its 4-OpenMP threads gets bound on Core-0 through Core-3, and so on
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/mvapich2-2.3rc1-userguide.html#x1-600006.5
Presentation Overview

- Job start-up
- Point-to-point Inter-node Protocol
- Transport Type Selection
- Multi-rail
- Process Mapping and Point-to-point Intra-node Protocols
- **Collectives**
- MPI_T Support
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)
- CMA-based collectives: MV2_USE_CMA_COLL (Default: ON)
• MCAST-based designs improve latency of MPI_Bcast by up to 85%
• Use MV2_USE_MCAST=1 to enable MCAST-based designs
MPI_Scatter - Benefits of using Hardware-Mcast

- Enabling MCAST-based designs for MPI_Scatter improves small message up to 75%.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</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>
Advanced Allreduce Collective Designs Using SHArP

**osu_allreduce (OSU Micro Benchmark) using MVAPICH2 2.3b**

**4 PPN*, 16 Nodes**

- **MVAPICH2**
- **MVAPICH2-SHArP**

**2.3x** increase in performance for 4 PPN and 16 Nodes.

**28 PPN, 16 Nodes**

- **MVAPICH2**
- **MVAPICH2-SHArP**

**1.5x** increase in performance for 28 PPN and 16 Nodes.

---

**Advanced Allreduce Collective Designs Using SHArP**

**osu_allreduce (OSU Micro Benchmark) using MVAPICH2 2.3b**

**4 PPN*, 16 Nodes**

- **MVAPICH2**
- **MVAPICH2-SHArP**

**2.3x** increase in performance for 4 PPN and 16 Nodes.

**28 PPN, 16 Nodes**

- **MVAPICH2**
- **MVAPICH2-SHArP**

**1.4x** increase in performance for 28 PPN and 16 Nodes.

---

**Notes**:

- **PPN**: Processes Per Node
- The graphs show the latency (us) against message size (Bytes) for different numbers of nodes and PPNs.
- SHArP provides a significant performance boost across various configurations.
Benefits of SHARP at Application Level

Avg DDOT Allreduce time of HPCG

Mesh Refinement Time of MiniAMR

SHARP support available since MVAPICH2 2.3a

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>MV2_ENABLE_SHARP=1</td>
<td>Enables SHARP-based collectives</td>
<td>Disabled</td>
</tr>
<tr>
<td>--enable-sharp</td>
<td>Configure flag to enable SHARP</td>
<td>Disabled</td>
</tr>
</tbody>
</table>

- Refer to Running Collectives with Hardware based SHArP support section of MVAPICH2 user guide for more information
- [http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.3b-userguide.html#x1-990006.26](http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.3b-userguide.html#x1-990006.26)
Optimized CMA-based Collectives for Large Messages

Performance of MPI_Gather on KNL nodes (64PPN)

- Significant improvement over existing implementation for Scatter/Gather with 1MB messages
  - Up to 4x on KNL, 2x on Broadwell, 14x on OpenPower

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>MV2_USE_CMA_COLL=1</td>
<td>Enables CMA-based collectives</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

Available in MVAPICH2-X 2.3b

- Refer to CMA Collective Specific Runtime Parameters section of MVAPICH2-X user guide for more information
- [http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-x-2.3b-userguide.html#x1-630008.5](http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-x-2.3b-userguide.html#x1-630008.5)

**Shared Address Space (XPMEM)-based Collectives Design**

**OSU_Allreduce (Broadwell 256 procs)**

- **MVAPICH2-2.3b**
- **IMPI-2017v1.132**
- **MVAPICH2-Opt**

- **73.2**
- **1.8X** improvement
- **36.1**

**Message Size**

16K 32K 64K 128K 256K 512K 1M 2M 4M

**Latency (us)**

100000 10000 1000 100 10 1

---

**OSU_Reduce (Broadwell 256 procs)**

- **MVAPICH2-2.3b**
- **IMPI-2017v1.132**
- **MVAPICH2-Opt**

- **37.9**
- **4X** improvement
- **16.8**

**Message Size**

16K 32K 64K 128K 256K 512K 1M 2M 4M

**Latency (us)**

100000 10000 1000 100 10 1

---

- “**Shared Address Space**”-based true **zero-copy** Reduction collective designs in MVAPICH2
- Offloaded computation/communication to peers ranks in reduction collective operation
- Up to **4X** improvement for 4MB Reduce and up to **1.8X** improvement for 4M AllReduce

*J. Hashmi, S. Chakraborty, M. Bayatpour, H. Subramoni, and D. Panda, Designing Efficient Shared Address Space Reduction Collectives for Multi-/Many-cores, International Parallel & Distributed Processing Symposium (IPDPS '18), May 2018.*

Will be available in future
Application-Level Benefits of XPMEM-Based Collectives

**CNTK AlexNet Training**
(Broadwell, B.S=default, iteration=50, ppn=28)

- Up to **20%** benefits over IMPI for CNTK DNN training using AllReduce
- Up to **27%** benefits over IMPI and up to **15%** improvement over MVAPICH2 for MiniAMR application kernel

**MiniAMR (Broadwell, ppn=16)**
Problems with Blocking Collective Operations

- Communication time cannot be used for compute
  - No overlap of computation and communication
  - Inefficient
Concept of Non-blocking Collectives

- 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
- There is a non-blocking equivalent for each blocking operation
  - Has an “I” in the name
    - MPI_Bcast -> MPI_Ibcast; MPI_Reduce -> MPI_Ireduce
How do I write applications with NBC?

void main()
{
    MPI_Init()
    .....  
    MPI_Ialltoall(...)  
    Computation that does not depend on result of Alltoall
    MPI_Test(for Ialltoall) /* Check if complete (non-blocking) */
    Computation that does not depend on result of Alltoall
    MPI_Wait(for Ialltoall) /* Wait till complete (Blocking) */
    ...
    MPI_Finalize()
}
P3DFFT Performance with Non-Blocking Alltoall using RDMA Primitives

- Weak scaling experiments; problem size increases with job size
- RDMA-Aware delivers 19% improvement over Default @ 8,192 procs
- Default-Thread exhibits worst performance
  - Possibly because threads steal CPU cycles from P3DFFT
  - Do not consider for large scale experiments

Will be available in future
Offloading with Scalable Hierarchical Aggregation Protocol (SHArP)

- Management and execution of MPI operations in the network by using SHArP
  - Manipulation of data while it is being transferred in the switch network
- SHArP provides an abstraction to realize the reduction operation
  - Defines Aggregation Nodes (AN), Aggregation Tree, and Aggregation Groups
  - AN logic is implemented as an InfiniBand Target Channel Adapter (TCA) integrated into the switch ASIC *
  - Uses RC for communication between ANs and between AN and hosts in the Aggregation Tree *

* Bloch et al. Scalable Hierarchical Aggregation Protocol (SHArP): A Hardware Architecture for Efficient Data Reduction
Evaluation of SHArP based Non Blocking Allreduce

**MPI_Iallreduce Benchmark**

- Complete offload of Allreduce collective operation to Switch helps to have much higher overlap of communication and computation

*PPN: Processes Per Node

Available since MVAPICH2 2.3a
Collective Offload in ConnectX-2, ConnectX-3, Connect-IB and ConnectX-4, ConnectX-5

- Mellanox’s ConnectX-2, ConnectX-3, ConnectIB, ConnectX-4, and ConnectX-5 adapters feature “task-list” offload interface
  - Extension to existing InfiniBand APIs
- Collective communication with ‘blocking’ feature is usually a scaling bottleneck
  - Matches with the need for non-blocking collective in MPI
- Accordingly MPI software stacks need to be re-designed to leverage offload in a comprehensive manner
- Can applications be modified to take advantage of non-blocking collectives and what will be the benefits?
Collective Offload Support in ConnectX InfiniBand Adapter (Recv followed by Multi-Send)

- Sender creates a task-list consisting of only send and wait WQEs
  - One send WQE is created for each registered receiver and is appended to the rear of a singly linked task-list
  - A wait WQE is added to make the ConnectX-2 HCA wait for ACK packet from the receiver
Co-designing HPL with Core-Direct and Performance Benefits

HPL Performance Comparison with 512 Processes

HPL-Offload consistently offers higher throughput than HPL-1ring and HPL-Host. Improves peak throughput by up to 4.5% for large problem sizes.

HPL-Offload surpasses the peak throughput of HPL-1ring with significantly smaller problem sizes and run-times!

K. Kandalla, H. Subramoni, J. Vienne, S. Pai Raikar, K. Tomko, S. Sur, and D K Panda,
Designing Non-blocking Broadcast with Collective Offload on InfiniBand Clusters: A Case Study with HPL, (HOTI 2011)

Available in MVAPICH2-X
Presentation Overview

- Job start-up
- Point-to-point Inter-node Protocol
- Transport Type Selection
- Multi-rail
- Process Mapping and Point-to-point Intra-node Protocols
- Collectives

- MPI_T Support
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
• Variables can differ between
  - MPI implementations
  - Compilations of same MPI library (production vs debug)
  - Executions of the same application/MPI library
  - There could be no variables provided
• Control Variables (CVARS) and Performance Variables (PVARS)
• More about the interface: mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf
MPI_T usage semantics

**Performance Variables**
- Initialize MPI-T
- Get #variables
- Query Metadata
- Allocate Session
- Allocate Handle
- Read/Write/Reset
  - Start/Stop var
- Free Handle
- Free Session

**Control Variables**
- Allocate Handle
- Read/Write var
- Free Handle

```c
int MPI_T_init_thread(int required, int *provided);
int MPI_T_cvar_get_num(int *num_cvar);
int MPI_T_cvar_get_info(int cvar_index, char *name, int *name_len, int *verbosity,
                        MPI_Datatype *datatype, MPI_T_enum *enumtype,
                        char *desc, int *desc_len, int *bind, int *scope);
int MPI_T_pvar_session_create(MPI_T_pvar_session *session);
int MPI_T_pvar_handle_alloc(MPI_T_pvar_session session, int pvar_index,
                            void *obj_handle, MPI_T_pvar_handle *handle, int *count);
int MPI_T_pvar_start(MPI_T_pvar_session session, MPI_T_pvar_handle handle);
int MPI_T_pvar_read(MPI_T_pvar_session session, MPI_T_pvar_handle handle, void* buf);
int MPI_T_pvar_reset(MPI_T_pvar_session session, MPI_T_pvar_handle handle);
int MPI_T_pvar_handle_free(MPI_T_pvar_session session, MPI_T_pvar_handle *handle);
int MPI_T_pvar_session_free(MPI_T_pvar_session *session);
int MPI_T_finalize(void);
```
Co-designing Applications to use MPI-T

Example Pseudo-code: 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(..)
```
Evaluating Applications with MPI-T

- Users can gain insights into application communication characteristics!
Performance Engineering Applications using MVAPICH2 and TAU

- Enhance existing support for MPI_T in MVAPICH2 to expose a richer set of performance and control variables
- Get and display MPI Performance Variables (PVARs) made available by the runtime in TAU
- Control the runtime’s behavior via MPI Control Variables (CVARs)
- Introduced support for new MPI_T based CVARs to MVAPICH2
  - MPIR_CVAR_MAX_INLINE_MSG_SZ, MPIR_CVAR_VBUF_POOL_SIZE, MPIR_CVAR_VBUF_SECONDARY_POOL_SIZE
- TAU enhanced with support for setting MPI_T CVARs in a non-interactive mode for uninstrumented applications
- S. Ramesh, A. Maheo, S. Shende, A. Malony, H. Subramoni, and D. K. Panda, MPI Performance Engineering with the MPI Tool Interface: the Integration of MVAPICH and TAU, EuroMPI/USA ‘17, Best Paper Finalist

Available in MVAPICH2

VBUF usage without CVAR based tuning as displayed by ParaProf

VBUF usage with CVAR based tuning as displayed by ParaProf
Enhancing MPI_T Support

- Introduced support for new MPI_T based CVARs to MVAPICH2
  - MPIR_CVAR_MAX_INLINE_MSG_SZ
    - Controls the message size up to which “inline” transmission of data is supported by MVAPICH2
  - MPIR_CVAR_VBUF_POOL_SIZE
    - Controls the number of internal communication buffers (VBUFs) MVAPICH2 allocates initially. Also,
      MPIR_CVAR_VBUF_POOL_REduced_VALUE[1] ([2...n])
  - MPIR_CVAR_VBUF_SECONDARY_POOL_SIZE
    - Controls the number of VBUFs MVAPICH2 allocates when there are no more free VBUFs available
  - MPIR_CVAR_IBA_EAGER_THRESHOLD
    - Controls the message size where MVAPICH2 switches from eager to rendezvous protocol for large messages
- TAU enhanced with support for setting MPI_T CVARs in a non-interactive mode for uninstrumented applications
# PVARs Exposed by MVAPICH2

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPT_PVAR0</td>
<td>mem_allocated</td>
<td>Current level of allocated memory within the MRT library</td>
</tr>
<tr>
<td>MPT_PVAR1</td>
<td>mv2_num_2level.comm.success</td>
<td>Number of successful 2-level comm creations</td>
</tr>
<tr>
<td>MPT_PVAR11</td>
<td>mv2_num_shmem_coll_calls</td>
<td>Number of times MV2 shared-memory collective calls were invoked</td>
</tr>
<tr>
<td>MPT_PVAR12</td>
<td>mv2_progress_poll</td>
<td>CH3 RDMA progress engine polling count</td>
</tr>
<tr>
<td>MPT_PVAR13</td>
<td>mv2_progress_poll completed</td>
<td>CH3 RDMA progress engine polling count</td>
</tr>
<tr>
<td>MPT_PVAR14</td>
<td>mv2_progress_poll</td>
<td>CH3 RDMA progress engine polling count</td>
</tr>
<tr>
<td>MPT_PVAR15</td>
<td>mv2_progress_poll completed</td>
<td>CH3 RDMA progress engine polling count</td>
</tr>
<tr>
<td>MPT_PVAR16</td>
<td>mv2_progress_poll</td>
<td>CH3 RDMA progress engine polling count</td>
</tr>
<tr>
<td>MPT_PVAR17</td>
<td>rdma_ud_retransmissions</td>
<td>CH3 RDMA UD retransmission count</td>
</tr>
<tr>
<td>MPT_PVAR18</td>
<td>mv2_bcast_bcast</td>
<td>Number of times MV2 bcast bcast algorithm was invoked</td>
</tr>
<tr>
<td>MPT_PVAR19</td>
<td>mv2_bcast_scatter_double_all</td>
<td>Number of times MV2 scatter-double all bcast algorithm was invoked</td>
</tr>
<tr>
<td>MPT_PVAR20</td>
<td>mv2_bcast_scatter_scatter</td>
<td>Number of times MV2 scatter-scatter bcast algorithm was invoked</td>
</tr>
<tr>
<td>MPT_PVAR21</td>
<td>mv2_bcast_scatter_scatter</td>
<td>Number of times MV2 scatter-scatter bcast algorithm was invoked</td>
</tr>
<tr>
<td>MPT_PVAR22</td>
<td>mv2_bcast_shmem</td>
<td>Number of times MV2 shmem bcast algorithm was invoked</td>
</tr>
<tr>
<td>MPT_PVAR23</td>
<td>mv2_bcast_kbcast</td>
<td>Number of times MV2 bcast bcast algorithm was invoked</td>
</tr>
<tr>
<td>MPT_PVAR24</td>
<td>mv2_bcast_kbcast</td>
<td>Number of times MV2 bcast bcast algorithm was invoked</td>
</tr>
<tr>
<td>MPT_PVAR25</td>
<td>mv2_bcast_mcast</td>
<td>Number of times MV2 mcast bcast algorithm was invoked</td>
</tr>
<tr>
<td>MPT_PVAR26</td>
<td>mv2_bcast_mcast</td>
<td>Number of times MV2 mcast bcast algorithm was invoked</td>
</tr>
<tr>
<td>MPT_PVAR27</td>
<td>mv2_bcast_mcast</td>
<td>Number of times MV2 mcast bcast algorithm was invoked</td>
</tr>
<tr>
<td>MPT_PVAR28</td>
<td>mv2_bcast_mcast</td>
<td>Number of times MV2 mcast bcast algorithm was invoked</td>
</tr>
<tr>
<td>MPT_PVAR29</td>
<td>mv2_bcast_mcast</td>
<td>Number of times MV2 mcast bcast algorithm was invoked</td>
</tr>
<tr>
<td>MPT_PVAR30</td>
<td>mv2_bcast_scatter</td>
<td>Number of times MV2 scatter bcast algorithm was invoked</td>
</tr>
<tr>
<td>MPT_PVAR31</td>
<td>mv2_alloct</td>
<td>Number of times MV2 alloct allocation was invoked</td>
</tr>
<tr>
<td>MPT_PVAR32</td>
<td>mv2_alloct</td>
<td>Number of times MV2 alloct allocation was invoked</td>
</tr>
<tr>
<td>MPT_PVAR33</td>
<td>mv2_alloct</td>
<td>Number of times MV2 alloct allocation was invoked</td>
</tr>
<tr>
<td>MPT_PVAR34</td>
<td>mv2_alloct</td>
<td>Number of times MV2 alloct allocation was invoked</td>
</tr>
<tr>
<td>MPT_PVAR35</td>
<td>mv2_alloct</td>
<td>Number of times MV2 alloct allocation was invoked</td>
</tr>
<tr>
<td>MPT_PVAR36</td>
<td>mv2_alloct</td>
<td>Number of times MV2 alloct allocation was invoked</td>
</tr>
<tr>
<td>MPT_PVAR37</td>
<td>mv2_alloct</td>
<td>Number of times MV2 alloct allocation was invoked</td>
</tr>
<tr>
<td>MPT_PVAR38</td>
<td>mv2_alloct</td>
<td>Number of times MV2 alloct allocation was invoked</td>
</tr>
<tr>
<td>MPT_PVAR39</td>
<td>mv2_alloct</td>
<td>Number of times MV2 alloct allocation was invoked</td>
</tr>
<tr>
<td>MPT_PVAR40</td>
<td>mv2_reg_cache_hit</td>
<td>Number of times MV2 reg cache hits</td>
</tr>
<tr>
<td>MPT_PVAR41</td>
<td>mv2_reg_cache_miss</td>
<td>Number of times MV2 reg cache misses</td>
</tr>
<tr>
<td>MPT_PVAR42</td>
<td>mv2_bcast</td>
<td>Number of MV2 bcast calls</td>
</tr>
<tr>
<td>MPT_PVAR43</td>
<td>mv2_bcast</td>
<td>Number of MV2 bcast calls</td>
</tr>
<tr>
<td>MPT_PVAR44</td>
<td>mv2_bcast</td>
<td>Number of MV2 bcast calls</td>
</tr>
<tr>
<td>MPT_PVAR45</td>
<td>mv2_bcast</td>
<td>Number of MV2 bcast calls</td>
</tr>
<tr>
<td>MPT_PVAR46</td>
<td>mv2_bcast</td>
<td>Number of MV2 bcast calls</td>
</tr>
<tr>
<td>MPT_PVAR47</td>
<td>mv2_bcast</td>
<td>Number of MV2 bcast calls</td>
</tr>
<tr>
<td>MPT_PVAR48</td>
<td>mv2_bcast</td>
<td>Number of MV2 bcast calls</td>
</tr>
<tr>
<td>MPT_PVAR49</td>
<td>mv2_bcast</td>
<td>Number of MV2 bcast calls</td>
</tr>
<tr>
<td>MPT_PVAR50</td>
<td>mv2_bcast</td>
<td>Number of MV2 bcast calls</td>
</tr>
<tr>
<td>MPT_PVAR51</td>
<td>mv2_bcast</td>
<td>Number of MV2 bcast calls</td>
</tr>
</tbody>
</table>

*Courtesy: The TAU Team*
CVARs Exposed by MVAPICH2

Network Based Computing Laboratory

SEA Symposium (April'18)

Network Based Computing Laboratory

CVARs Exposed by MVAPICH2

- Courtesy: The TAU Team
Using MVAPICH2 and TAU

- To set CVARs or read PVARs using TAU for an uninstrumented binary:
  % export TAU_TRACK_MPI_T_PVARS=1
  % export TAU_MPI_T_CVAR_METRICS=
      MPIR_CVAR_VBUF_POOL_REDUCE_VALUE[1],
      MPIR_CVAR_IBA_EAGER_THRESHOLD
  % export TAU_MPI_T_CVAR_VALUES=32,64000
  % export PATH=/path/to/tau/x86_64/bin:$PATH
  % mpirun -np 1024 tau_exec -T mvapich2,mpit ./a.out
  % paraprof

Courtesy: The TAU Team
VBUF usage without CVARs

<table>
<thead>
<tr>
<th>Name</th>
<th>MaxValue</th>
<th>MinValue</th>
<th>MeanValue</th>
<th>Std. Dev.</th>
<th>NumSamples</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>mv2_total_vbuf_memory (Total amount of memory in bytes used for VBUFs)</td>
<td>3,313,056</td>
<td>3,313,056</td>
<td>3,313,056</td>
<td>0</td>
<td>1</td>
<td>3,313,056</td>
</tr>
<tr>
<td>mv2_ud_vbuf_allocated (Number of UD VBUFs allocated)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mv2_ud_vbuf_available (Number of UD VBUFs available)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mv2_ud_vbuf_freed (Number of UD VBUFs freed)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mv2_ud_vbuf_inuse (Number of UD VBUFs inuse)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mv2_ud_vbuf_max_use (Maximum number of UD VBUFs used)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mv2_vbuf_allocated (Number of VBUFs allocated)</td>
<td>320</td>
<td>320</td>
<td>320</td>
<td>0</td>
<td>1</td>
<td>320</td>
</tr>
<tr>
<td>mv2_vbuf_available (Number of VBUFs available)</td>
<td>255</td>
<td>255</td>
<td>255</td>
<td>0</td>
<td>1</td>
<td>255</td>
</tr>
<tr>
<td>mv2_vbuf_freed (Number of VBUFs freed)</td>
<td>25,545</td>
<td>25,545</td>
<td>25,545</td>
<td>0</td>
<td>1</td>
<td>25,545</td>
</tr>
<tr>
<td>mv2_vbuf_inuse (Number of VBUFs inuse)</td>
<td>65</td>
<td>65</td>
<td>65</td>
<td>0</td>
<td>1</td>
<td>65</td>
</tr>
<tr>
<td>mv2_vbuf_max_use (Maximum number of VBUFs used)</td>
<td>65</td>
<td>65</td>
<td>65</td>
<td>0</td>
<td>1</td>
<td>65</td>
</tr>
<tr>
<td>num_malloc_calls (Number of MPIT_malloc calls)</td>
<td>89</td>
<td>89</td>
<td>89</td>
<td>0</td>
<td>1</td>
<td>89</td>
</tr>
<tr>
<td>num_free_calls (Number of MPIT_free calls)</td>
<td>47,801</td>
<td>47,801</td>
<td>47,801</td>
<td>0</td>
<td>1</td>
<td>47,801</td>
</tr>
<tr>
<td>num_malloc Calls (Number of MPIT_malloc calls)</td>
<td>49,258</td>
<td>49,258</td>
<td>49,258</td>
<td>0</td>
<td>1</td>
<td>49,258</td>
</tr>
<tr>
<td>num_memalign_calls (Number of MPIT_memalign calls)</td>
<td>34</td>
<td>34</td>
<td>34</td>
<td>0</td>
<td>1</td>
<td>34</td>
</tr>
<tr>
<td>num_memalign_free_calls (Number of MPIT_memalign_free calls)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Courtesy: The TAU Team
VBUF usage with CVARs

Total memory used by VBUFs is reduced from 3,313,056 to 1,815,056

Courtesy: The TAU Team
VBUF Memory Usage Without CVAR

Courtesy: The TAU Team
VBUF Memory Usage With CVAR

% export TAU_TRACK_MPI_T_PVARS=1
% export TAU_MPI_T_CVAR_METRICS=MPIR_CVAR_VBUF_POOL_SIZE
% export TAU_MPI_T_CVAR_VALUES=16
% mpirun -np 1024 tau_exec -T mvapich2 ./a.out

Courtesy: The TAU Team
## MVAPICH2 Software Family

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI with IB, iWARP and RoCE</td>
<td>MVAPICH2</td>
</tr>
<tr>
<td>Advanced MPI, OSU INAM, PGAS and MPI+PGAS with IB and RoCE</td>
<td>MVAPICH2-X</td>
</tr>
<tr>
<td>MPI with IB &amp; GPU</td>
<td>MVAPICH2-GDR</td>
</tr>
<tr>
<td>Energy-aware MPI with IB, iWARP and RoCE</td>
<td>MVAPICH2-EA</td>
</tr>
<tr>
<td>MPI Energy Monitoring Tool</td>
<td>OEMT</td>
</tr>
<tr>
<td>InfiniBand Network Analysis and Monitoring</td>
<td>OSU INAM</td>
</tr>
<tr>
<td>Microbenchmarks for Measuring MPI and PGAS Performance</td>
<td>OMB</td>
</tr>
</tbody>
</table>
MVAPICH2-X for Hybrid MPI + PGAS Applications

**High Performance Parallel Programming Models**

<table>
<thead>
<tr>
<th>MPI</th>
<th>PGAS</th>
<th>Hybrid --- MPI + X</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message Passing Interface</td>
<td>(UPC, OpenSHMEM, CAF, UPC++)</td>
<td>(MPI + PGAS + OpenMP/Cilk)</td>
</tr>
</tbody>
</table>

**High Performance and Scalable Unified Communication Runtime**

**Diverse APIs and Mechanisms**

- Optimized Point-to-point Primitives
- Remote Memory Access
- Active Messages
- Collectives Algorithms (Blocking and Non-Blocking)
- Scalable Job Startup
- Fault Tolerance
- Introspection & Analysis with OSU INAM

**Support for Modern Networking Technologies**

- (InfiniBand, iWARP, RoCE, Omni-Path...)

**Support for Modern Multi-/Many-core Architectures**

- (Intel-Xeon, OpenPower...)

- **Current Model – Separate Runtimes for OpenSHMEM/UPC/UPC++/CAF and MPI**
  - Possible deadlock if both runtimes are not progressed
  - Consumes more network resource

- **Unified communication runtime for MPI, UPC, UPC++, OpenSHMEM, CAF**
  - Available with since 2012 (starting with MVAPICH2-X 1.9)
  - [http://mvapich.cse.ohio-state.edu](http://mvapich.cse.ohio-state.edu)
UPC++ Support in MVAPICH2-X

- Full and native support for hybrid MPI + UPC++ applications
- Better performance compared to IBV and MPI conduits
- OSU Micro-benchmarks (OMB) support for UPC++
- Available since MVAPICH2-X (2.2rc1)

More Details in Student Poster Presentation
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

J. Jose, K. Kandalla, M. Luo and D. K. Panda, Supporting Hybrid MPI and OpenSHMEM over InfiniBand: Design and Performance Evaluation, Int'l Conference on Parallel Processing (ICPP '12), September 2012
OpenSHMEM Application kernels

- AVX-512 vectorization and MCDRAM based optimizations for MVAPICH2-X
  - Will be available in future MVAPICH2-X release

- AVX-512 vectorization showed up to 3X improvement over default
- KNL showed on-par performance as Broadwell on Node-by-node comparison

Implicit On-Demand Paging (ODP)

- Introduced by Mellanox to avoid pinning the pages of registered memory regions
- ODP-aware runtime could reduce the size of pin-down buffers while maintaining performance

### MVAPICH2 Software Family

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI with IB, iWARP and RoCE</td>
<td>MVAPICH2</td>
</tr>
<tr>
<td>Advanced MPI, OSU INAM, PGAS and MPI+PGAS with IB and RoCE</td>
<td>MVAPICH2-X</td>
</tr>
<tr>
<td>MPI with IB &amp; GPU</td>
<td>MVAPICH2-GDR</td>
</tr>
<tr>
<td>Energy-aware MPI with IB, iWARP and RoCE</td>
<td>MVAPICH2-EA</td>
</tr>
<tr>
<td>MPI Energy Monitoring Tool</td>
<td>OEMT</td>
</tr>
<tr>
<td>InfiniBand Network Analysis and Monitoring</td>
<td>OSU INAM</td>
</tr>
<tr>
<td>Microbenchmarks for Measuring MPI and PGAS Performance</td>
<td>OMB</td>
</tr>
</tbody>
</table>
GPU-Aware (CUDA-Aware) MPI Library: MVAPICH2-GPU

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

At Sender:

MPI_Send(s_devbuf, size, …);

At Receiver:

MPI_Recv(r_devbuf, size, …);

High Performance and High Productivity
CUDA-Aware MPI: MVAPICH2-GDR 1.8-2.3 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
- Unified memory
Presentation Overview

- Support for Efficient Small Message Communication with GPUDirect RDMA
- Multi-rail Support
- Support for Efficient Intra-node Communication using CUDA IPC
- MPI Datatype Support
Enhanced MPI Design with GPUDirect RDMA

- Current MPI design using GPUDirect RDMA uses Rendezvous protocol
  - Has higher latency for small messages
- Can eager protocol be supported to improve performance for small messages?
- Two schemes proposed and used
  - Loopback (using network adapter to copy data)
  - Fastcopy/GDRCOPY (using CPU to copy data)

Optimized MVAPICH2-GDR Design

**GPU-GPU Inter-node Latency**

- MV2-(NO-GDR)
- MV2-GDR-2.3a

**GPU-GPU Inter-node Bandwidth**

- MV2-(NO-GDR)
- MV2-GDR-2.3a

**Graphical Observations**

- GPU-GPU Inter-node Latency:
  - MV2-(NO-GDR) vs MV2-GDR-2.3a:
    - 1.88us improvement with 10x speedup.

- GPU-GPU Inter-node Bandwidth:
  - MV2-(NO-GDR) vs MV2-GDR-2.3a:
    - 9x speedup at 10x improvement.

**System Configuration**

- Intel Haswell (E5-2687W @ 3.10 GHz) node - 20 cores
- NVIDIA Volta V100 GPU
- Mellanox Connect-X4 EDR HCA
- CUDA 9.0
- Mellanox OFED 4.0 with GPU-Direct-RDMA

**Observations**

- MVAPICH2-GDR-2.3a demonstrates significant improvements in both latency and bandwidth compared to the NO-GDR version.
**MVAPICH2-GDR: Performance on OpenPOWER (NVLink + Pascal)**

- **Intra-node Latency**: 14.6 us (without GPUDirectRDMA)

- **Intra-node Bandwidth**: 33.9 GB/sec (NVLink)

- **Inter-node Latency**: 23.8 us (without GPUDirectRDMA)

- **Inter-node Bandwidth**: 11.9 GB/sec (EDR)

Available in MVAPICH2-GDR 2.3a

Platform: OpenPOWER (ppc64le) nodes equipped with a dual-socket CPU, 4 Pascal P100-SXM GPUs, and EDR InfiniBand Inter-connect
### Tuning GDRCOPY Designs in MVAPICH2-GDR

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Significance</th>
<th>Default</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MV2_USE_GPUDIRECT_GDRCOPY</td>
<td>• Enable / Disable GDRCOPY-based designs</td>
<td>1 (Enabled)</td>
<td>• Always enable</td>
</tr>
<tr>
<td>MV2_GPUDIRECT_GDR_COPY_LIMIT</td>
<td>• Controls messages size until which GDRCOPY is used</td>
<td>8 KByte</td>
<td>• Tune for your system</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• GPU type, host architecture. Impacts the eager performance</td>
</tr>
<tr>
<td>MV2_GPUDIRECT_GDR_COPY_LIB</td>
<td>• Path to the GDRCOPY library</td>
<td>Unset</td>
<td>• Always set</td>
</tr>
<tr>
<td>MV2_USE_GPUDIRECT_D2H_GDRCOPY_LIMIT</td>
<td>• Controls messages size until which GDRCOPY is used at sender</td>
<td>16Bytes</td>
<td>• Tune for your systems</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• CPU and GPU type</td>
</tr>
</tbody>
</table>

- Refer to Tuning and Usage Parameters section of MVAPICH2-GDR user guide for more information
- [http://mvapich.cse.ohio-state.edu/userguide/gdr/#_tuning_and_usage_parameters](http://mvapich.cse.ohio-state.edu/userguide/gdr/#_tuning_and_usage_parameters)
## Tuning Loopback Designs in MVAPICH2-GDR

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Significance</th>
<th>Default</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MV2_USE_GPUDIRECT_LOOPBACK</td>
<td>• Enable / Disable LOOPBACK-based designs</td>
<td>1 (Enabled)</td>
<td>• Always enable</td>
</tr>
</tbody>
</table>
| MV2_GPUDIRECT_LOOPBACK_LIMIT | • Controls messages size until which LOOPBACK is used | 8 KByte | • Tune for your system  
• GPU type, host architecture and HCA. Impacts the eager performance  
• Sensitive to the P2P issue |

- Refer to [Tuning and Usage Parameters](http://mvapich.cse.ohio-state.edu/userguide/gdr/#_tuning_and_usage_parameters) section of MVAPICH2-GDR user guide for more information
## Tuning GPUDirect RDMA (GDR) Designs in MVAPICH2-GDR

<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 (Enabled)</td>
<td>• Always enable</td>
</tr>
</tbody>
</table>
| MV2_GPUDIRECT_LIMIT | Controls messages size until which GPUDirect RDMA is used | 8 KByte | • Tune for your system  
• GPU type, host architecture and  
CUDA version: impact pipelining overheads and P2P bandwidth bottlenecks |
| MV2_USE_GPUDIRECT_RECEIVE_LIMIT | Controls messages size until which 1 hop design is used (GDR Write at the receiver) | 256KBytes | • Tune for your system  
• GPU type, HCA type and configuration |

- Refer to Tuning and Usage Parameters section of MVAPICH2-GDR user guide for more information
- [http://mvapich.cse.ohio-state.edu/userguide/gdr/#_tuning_and_usage_parameters](http://mvapich.cse.ohio-state.edu/userguide/gdr/#_tuning_and_usage_parameters)
Application-Level Evaluation (HOOMD-blue)

64K Particles

256K Particles

- Platform: Wilkes (Intel Ivy Bridge + NVIDIA Tesla K20c + Mellanox Connect-IB)
- HoomDBlue Version 1.0.5
  - 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
Presentation Overview

- Support for Efficient Small Message Communication with GPUDirect RDMA
- Multi-rail Support
- Support for Efficient Intra-node Communication using CUDA IPC
- MPI Datatype Support
Tuning Multi-rail Support in MVAPICH2-GDR

- Automatic rail and CPU binding depending on the GPU selection
  - User selects the GPU and MVAPICH2-GDR selects the best HCA (avoids the P2P bottleneck)
  - Multi-rail selection for large message size for better Bandwidth utilization (pipeline design)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Significance</th>
<th>Default</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MV2_RAIL_SHARING_POLICY</td>
<td>• How the Rails are bind/selected by processes</td>
<td>Shared</td>
<td>• Sharing gives the best performance for pipeline design</td>
</tr>
<tr>
<td>MV2_PROCESS_TO_RAIL_MAPP ING</td>
<td>• Explicit binding of the HCAs to the CPU</td>
<td>First HCA</td>
<td>• Use this parameter to manually select a different parameter only if default binding seems to perform poorly</td>
</tr>
</tbody>
</table>

- Refer to Tuning and Usage Parameters section of MVAPICH2-GDR user guide for more information
- [http://mvapich.cse.ohio-state.edu/userguide/gdr/#_tuning_and_usage_parameters](http://mvapich.cse.ohio-state.edu/userguide/gdr/#_tuning_and_usage_parameters)
Performance of MVAPICH2-GDR with GPU-Direct RDMA and Multi-Rail Support

MVAPICH2-GDR-2.1 and MVAPICH2-GDR 2.1 RC2
Intel Ivy Bridge (E5-2680 v2) node - 20 cores, NVIDIA Tesla K40c GPU
Mellanox Connect-IB Dual-FDR HCA CUDA 7
Mellanox OFED 2.4 with GPU-Direct-RDMA
Presentation Overview

- Support for Efficient Small Message Communication with GPUDirect RDMA
- Multi-rail Support
  - **Support for Efficient Intra-node Communication using CUDA IPC**
  - MPI Datatype Support
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
Tuning IPC designs in MVAPICH2-GDR

- Works between GPUs within the same socket or IOH
- Leads to significant benefits in appropriate scenarios

<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>
<tr>
<td>MV2_CUDA_SMP_IPC</td>
<td>• Enable / Disable CUDA IPC fastpath design for short messages</td>
<td>0 (Disabled)</td>
<td>• Benefits Device-to-Device transfers</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Hurts Device-to-Host/Host-to-Device transfers</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Always set to 1 if application involves only Device-to-Device transfers</td>
</tr>
<tr>
<td>MV2_IPC_THRESHOLD</td>
<td>• Message size where IPC code path will be used</td>
<td>16 KBytes</td>
<td>• Tune for your system</td>
</tr>
</tbody>
</table>
Alternative Designs

• Double Buffering schemes
  – Uses intermediate buffers (IPC Pinned)
  – Control information through Host memories
    • Exchange the handlers through the host for IPC completion
  – Works for all CUDA versions (Since 5.5)
  – Memory Overhead

• Cache based design
  – Rendezvous based design
  – Cache the IPC handlers at the source and destination (through the control messages)
  – With Cache hit => direct data movement
  – Requires CUDA 6.5 and onwards
  – High Performance and memory efficiency
### Tuning IPC designs in MVAPICH2-GDR

- 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>
</table>
| MV2_CUDA_ENABLE_IPC_CACHE                      | • Enable / Disable CUDA IPC_CACHE-based designs   | 1 (Enabled) | • Always leave set to 1  
• Best performance  
• Enables one-sided semantics |
| MV2_CUDA_IPC_BUFFEREAD                        | • Enable / Disable CUDA IPC_BUFFERED design       | 1 (Enabled) | • Used for subset of operations  
• Backup for the IPC-Cache design  
• Uses double buffering schemes  
• Used for efficient Managed support |
| MV2_CUDA_IPC_MAX_CACHE_ENTRIES                 | • Number of entries in the cache                  | 64      | • Tuned for your application  
• Depends on the communication patterns  
• Increase the value for irregular applications |
| MV2_CUDA_IPC_STAGE_BUF_SIZE                    | • The size of the staging buffers in the double buffering schemes |         | • Tune this value only if degradation is observed with IPC transfers |
• Two Processes sharing the same K80 GPUs
• Proposed designs achieve 4.7X improvement in latency
• 7.8X improvement is delivered for Bandwidth
• Available with the latest release of MVAPICH2-GDR 2.2
Presentation Overview

- Support for Efficient Small Message Communication with GPUDirect RDMA
- Multi-rail Support
- Support for Efficient Intra-node Communication using CUDA IPC
- **MPI Datatype Support**
Non-contiguous 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 Processing (Computation Optimization)

- **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
Performance of Stencil3D (3D subarray)

Stencil3D communication kernel on 2 GPUs with various X, Y, Z dimensions using MPI_Isend/Irecv

- DT: Direct Transfer, TR: Targeted Kernel
- Optimized design gains up to 15%, 15% and 22% compared to TR, and more than 86% compared to DT on X, Y and Z respectively.

![Graph showing latency vs. size for Stencil3D communication kernel on 2 GPUs with various X, Y, Z dimensions using MPI_Isend/Irecv. The graph compares Direct Transfer (DT), Targeted (TR), and Enhanced designs, indicating performance gains up to 15%, 15%, and 22% compared to TR, and more than 86% compared to DT on X, Y and Z respectively.]
Common Scenario

MPI_Isend (A,.. Datatype,...)
MPI_Isend (B,.. Datatype,...)
MPI_Isend (C,.. Datatype,...)
MPI_Isend (D,.. Datatype,...)
...

MPI_Waitall (...);

*A, B...contain non-contiguous MPI Datatype
Application-Level Evaluation (Cosmo) and Weather Forecasting in Switzerland

Wilkes GPU Cluster

CSCS GPU cluster

- 2X improvement on 32 GPUs nodes
- 30% improvement on 96 GPU nodes (8 GPUs/node)

On-going collaboration with CSCS and MeteoSwiss (Switzerland) in co-designing MV2-GDR and Cosmo Application


Cosmo model: http://www2.cosmo-model.org/content/tasks/operational/meteoSwiss/
Enhanced Support for GPU Managed Memory

- CUDA Managed => no memory pin down
  - No IPC support for intranode communication
  - No GDR support for Internode communication
- Significant productivity benefits due to abstraction of explicit allocation and cudaMemcpy()
- Initial and basic support in MVAPICH2-GDR
  - For both intra- and inter-nodes use “pipeline through” host memory
- Enhance intranode managed memory to use IPC
  - Double buffering pair-wise IPC-based scheme
  - Brings IPC performance to Managed memory
  - High performance and high productivity
  - 2.5 X improvement in bandwidth
- OMB extended to evaluate the performance of point-to-point and collective communications using managed buffers

D. S. Banerjee, K Hamidouche, and D. K Panda, Designing High Performance Communication Runtime for GPUManaged Memory: Early Experiences, GPGPU-9 Workshop, held in conjunction with PPoPP ’16
## MVAPICH2 Software Family

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI with IB, iWARP and RoCE</td>
<td>MVAPICH2</td>
</tr>
<tr>
<td>Advanced MPI, OSU INAM, PGAS and MPI+PGAS with IB and RoCE</td>
<td>MVAPICH2-X</td>
</tr>
<tr>
<td>MPI with IB &amp; GPU</td>
<td>MVAPICH2-GDR</td>
</tr>
<tr>
<td>Energy-aware MPI with IB, iWARP and RoCE</td>
<td>MVAPICH2-EA</td>
</tr>
<tr>
<td>MPI Energy Monitoring Tool</td>
<td>OEMT</td>
</tr>
<tr>
<td>InfiniBand Network Analysis and Monitoring</td>
<td>OSU INAM</td>
</tr>
<tr>
<td>Microbenchmarks for Measuring MPI and PGAS Performance</td>
<td>OMB</td>
</tr>
</tbody>
</table>
Energy-Aware MVAPICH2 & OSU Energy Management Tool (OEMT)

- MVAPICH2-EA 2.1 (Energy-Aware)
  - A white-box approach
  - New Energy-Efficient communication protocols for pt-pt and collective operations
  - Intelligently apply the appropriate Energy saving techniques
  - Application oblivious energy saving

- OEMT
  - A library utility to measure energy consumption for MPI applications
  - Works with all MPI runtimes
  - PRELOAD option for precompiled applications
  - Does not require ROOT permission:
    - A safe kernel module to read only a subset of MSRs
Designing Energy-Aware (EA) MPI Runtime

Overall application Energy Expenditure

- Energy Spent in Communication Routines
- Energy Spent in Computation Routines

- Point-to-point Routines
- Collective Routines
- RMA Routines

MVAPICH2-EA Designs

- MPI Two-sided and collectives (ex: MVAPICH2)
- One-sided runtimes (ex: ComEx)
- Other PGAS Implementations (ex: OSHMPI)

Impact

- MPI-3 RMA Implementations (ex: MVAPICH2)
MVAPICH2-EA: Application Oblivious Energy-Aware-MPI (EAM)

- An energy efficient runtime that provides energy savings without application knowledge
- Uses automatically and transparently the best energy lever
- Provides guarantees on maximum degradation with 5-41% savings at \( \leq 5\% \) degradation
- Pessimistic MPI applies energy reduction lever to each MPI call

MPI-3 RMA Energy Savings with Proxy-Applications

- MPI_Win_fence dominates application execution time in graph500
- Between 128 and 512 processes, EAM-RMA yields between 31% and 46% savings with no degradation in execution time in comparison with the default optimistic MPI runtime
• SCF (self-consistent field) calculation spends nearly 75% total time in MPI_Win_unlock call
• With 256 and 512 processes, EAM-RMA yields 42% and 36% savings at 11% degradation (close to permitted degradation $\rho = 10\%$)
• 128 processes is an exception due 2-sided and 1-sided interaction
• MPI-3 RMA Energy-efficient support will be available in upcoming MVAPICH2-EA release
## MVAPICH2 Software Family

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI with IB, iWARP and RoCE</td>
<td>MVAPICH2</td>
</tr>
<tr>
<td>Advanced MPI, OSU INAM, PGAS and MPI+PGAS with IB and RoCE</td>
<td>MVAPICH2-X</td>
</tr>
<tr>
<td>MPI with IB &amp; GPU</td>
<td>MVAPICH2-GDR</td>
</tr>
<tr>
<td>Energy-aware MPI with IB, iWARP and RoCE</td>
<td>MVAPICH2-EA</td>
</tr>
<tr>
<td>MPI Energy Monitoring Tool</td>
<td>OEMT</td>
</tr>
<tr>
<td>InfiniBand Network Analysis and Monitoring</td>
<td>OSU INAM</td>
</tr>
<tr>
<td>Microbenchmarks for Measuring MPI and PGAS Performance</td>
<td>OMB</td>
</tr>
</tbody>
</table>
Overview of OSU INAM

- A network monitoring and analysis tool that is capable of analyzing traffic on the InfiniBand network with inputs from the MPI runtime
  - [http://mvapich.cse.ohio-state.edu/tools/osu-inam/](http://mvapich.cse.ohio-state.edu/tools/osu-inam/)
- Monitors IB clusters in real time by querying various subnet management entities and gathering input from the MPI runtimes
- OSU INAM v0.9.3 released on 03/06/18
  - Enhance INAMD to query end nodes based on command line option
  - Add a web page to display size of the database in real-time
  - Enhance interaction between the web application and SLURM job launcher for increased portability
  - Improve packaging of web application and daemon to ease installation
  - Enhance web interface to improve the user experience
  - Improve debugging and logging support in daemon and web application
- Significant enhancements to user interface to enable scaling to clusters with thousands of nodes
- Improve database insert times by using ‘bulk inserts’
- Capability to look up list of nodes communicating through a network link
- Capability to classify data flowing over a network link at job level and process level granularity in conjunction with MVAPICH2-X 2.2rc1
- “Best practices " guidelines for deploying OSU INAM on different clusters
- Capability to analyze and profile node-level, job-level and process-level activities for MPI communication
  - Point-to-Point, Collectives and RMA
- Ability to filter data based on type of counters using “drop down” list
- Remotely monitor various metrics of MPI processes at user specified granularity
- "Job Page" to display jobs in ascending/descending order of various performance metrics in conjunction with MVAPICH2-X
- Visualize the data transfer happening in a “live” or “historical” fashion for entire network, job or set of nodes
OSU INAM Features

Comet@SDSC --- Clustered View

(1,879 nodes, 212 switches, 4,377 network links)

- Show network topology of large clusters
- Visualize traffic pattern on different links
- Quickly identify congested links/links in error state
- See the history unfold – play back historical state of the network
OSU INAM Features (Cont.)

- **Job level view**
  - Show different network metrics (load, error, etc.) for any live job
  - Play back historical data for completed jobs to identify bottlenecks

- **Node level view - details per process or per node**
  - CPU utilization for each rank/node
  - Bytes sent/received for MPI operations (pt-to-pt, collective, RMA)
  - Network metrics (e.g. XmitDiscard, RcvError) per rank/node

Estimated Process Level Link Utilization

- **Estimated Link Utilization view**
  - Classify data flowing over a network link at different granularity in conjunction with MVAPICH2-X 2.2rc1
    - Job level and
    - Process level
  - More Details in Tutorial/Demo

Session Tomorrow

Visualizing a Job (5 Nodes)
OSU Microbenchmarks

• Available since 2004

• Suite of microbenchmarks to study communication performance of various programming models

• Benchmarks available for the following programming models
  – Message Passing Interface (MPI)
  – Partitioned Global Address Space (PGAS)
    • Unified Parallel C (UPC)
    • Unified Parallel C++ (UPC++)
    • OpenSHMEM

• Benchmarks available for multiple accelerator based architectures
  – Compute Unified Device Architecture (CUDA)
  – OpenACC Application Program Interface

• Part of various national resource procurement suites like NERSC-8 / Trinity Benchmarks

• Continuing to add support for newer primitives and features

• Please visit the following link for more information
  – http://mvapich.cse.ohio-state.edu/benchmarks/
Applications-Level Tuning: Compilation of Best Practices

- MPI runtime has many parameters
- Tuning a set of parameters can help you to extract higher performance
- Compiled a list of such contributions through the MVAPICH Website
  - [http://mvapich.cse.ohio-state.edu/best_practices/](http://mvapich.cse.ohio-state.edu/best_practices/)
- Initial list of applications
  - Amber
  - HoomDBlue
  - HPCG
  - Lulesh
  - MILC
  - Neuron
  - SMG2000
  - Cloverleaf
  - SPEC (LAMMPS, POP2, TERA_TF, WRF2)
- Soliciting additional contributions, send your results to mvapich-help at cse.ohio-state.edu.
- We will link these results with credits to you.
Amber: Impact of Tuning Eager Threshold

- Tuning the Eager threshold has a significant impact on application performance by avoiding the synchronization of rendezvous protocol and thus yielding better communication computation overlap
- 19% improvement in overall execution time at 256 processes
- Library Version: MVAPICH2 2.2
- MVAPICH Flags used
  - MV2_IBA_EAGER_THRESHOLD=131072
  - MV2_VBUF_TOTAL_SIZE=131072
- Input files used
  - Small: MDIN
  - Large: PMTOP

Data Submitted by: Dong Ju Choi @ UCSD
MiniAMR: Impact of Tuning Eager Threshold

- Tuning the Eager threshold has a significant impact on application performance by avoiding the synchronization of rendezvous protocol and thus yielding better communication computation overlap.
- 8% percent reduction in total communication time.
- Library Version: MVAPICH2 2.2.
- MVAPICH Flags used:
  - MV2_IBA_EAGER_THRESHOLD=32768
  - MV2_VBUF_TOTAL_SIZE=32768

Data Submitted by Karen Tomko @ OSC and Dong Ju Choi @ UCSD
UD-based transport protocol selection benefits the SMG2000 application.

- 22% and 6% on 1,024 and 4,096 cores, respectively.

- Library Version: MVAPICH2 2.1
- MVAPICH Flags used:
  - MV2_USE_ONLY_UD=1

- System Details:
  - Stampede@ TACC
  - Sandybridge architecture with dual 8-cores nodes and ConnectX-3 FDR network

SMG2000: Impact of Tuning Transport Protocol

Data Submitted by Jerome Vienne @ TACC
UD-based transport protocol selection benefits the SMG2000 application

- 15% and 27% improvement is seen for 768 and 1,024 processes respectively
- Library Version: MVAPICH2 2.2
- MVAPICH Flags used
  - `MV2_USE_ONLY_UD=1`
- Input File
  - `YuEtAl2012`
- System Details
  - Comet@SDSC
  - Haswell nodes with dual 12-cores socket per node and Mellanox FDR (56 Gbps) network.

Data Submitted by Mahidhar Tatineni @ SDSC
HPCG: Impact of Collective Tuning for MPI+OpenMP Programming Model

- Partial subscription nature of hybrid MPI+OpenMP programming requires a new level of collective tuning
  - For PPN=2 (Processes Per Node), the tuned version of MPI_Reduce shows 51% improvement on 2,048 cores

- 24% improvement on 512 cores
  - 8 OpenMP threads per MPI processes

- Library Version: MVAPICH2 2.1

- MVAPICH Flags used
  - The tuning parameters for hybrid MPI+OpenMP programming models is on by default from MVAPICH2-2.1 onward

- System Details
  - Stampede@ TACC
  - Sandybridge architecture with dual 8-cores nodes and ConnectX-3 FDR network

Data Submitted by Jerome Vienne and Carlos Rosales-Fernandez @ TACC
Partial subscription nature of hybrid MPI+OpenMP programming requires a new level of collective tuning:

- For PPN=2 (Processes Per Node), the tuned version of MPI_Reduce shows 51% improvement on 2,048 cores.
- 4% improvement on 512 cores.
- 8 OpenMP threads per MPI processes.

Library Version: MVAPICH2 2.1

MVAPICH Flags used:
- The tuning parameters for hybrid MPI+OpenMP programming models is on by default from MVAPICH2-2.1 onward.

System Details:
- Stampede@ TACC
- Sandybridge architecture with dual 8-cores nodes and ConnectX-3 FDR network.

Data Submitted by Jerome Vienne and Carlos Rosales-Fernandez @ TACC.
Non-contiguous data processing is very common on HPC applications. MVAPICH2 offers efficient designs for MPI Datatype support using novel hardware features such as UMR.

UMR-based protocol selection benefits the MILC application.
- 4% and 6% improvement in execution time at 512 and 640 processors, respectively.

Library Version: MVAPICH2-X 2.2

MVAPICH Flags used
- MV2_USE_UMR=1

System Details
- The experimental cluster consists of 32 Ivy Bridge Compute nodes interconnected by Mellanox FDR.
- The Intel Ivy Bridge processors consist of Xeon dual ten-core sockets operating at 2.80GHz with 32GB RAM and Mellanox OFED version 3.2-1.0.1.1.

Data Submitted by Mingzhe Li @ OSU
HOOMD-blue: Impact of GPUDirect RDMA Based Tuning

- HOOMD-blue is a Molecular Dynamics simulation using a custom force field.
- GPUDirect specific features selection and tuning significantly benefit the HOOMD-blue application. We observe a factor of 2X improvement on 32 GPU nodes, with both 64K and 256K particles
- Library Version: MVAPICH2-GDR 2.2
- MVAPICH-GDR Flags used
  - MV2_USE_CUDA=1
  - MV2_USE_GPUDIRECT=1
  - MV2_GPUDIRECT_GDRCOPY=1
- System Details
  - Wilkes@Cambridge
  - 128 Ivybridge nodes, each node is a dual 6-cores socket with Mellanox FDR

Data Submitted by Khaled Hamidouche @ OSU

### 64K Particles

<table>
<thead>
<tr>
<th>Number of Processes</th>
<th>Default</th>
<th>Tuned</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Average Time Steps per second (TPS)

### 256K Particles

<table>
<thead>
<tr>
<th>Number of Processes</th>
<th>MV2</th>
<th>MV2+GDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Average Time Steps per second (TPS)
Application Scalability on Skylake and KNL with Omni-Path

MiniFE (1300x1300x1300 ~ 910 GB)

Execution Time (s)

No. of Processes (Skylake: 48ppn)

No. of Processes (KNL: 64ppn)

NEURON (YuEtAl2012)

Execution Time (s)

No. of Processes (Skylake: 48ppn)

No. of Processes (KNL: 64ppn)

Cloverleaf (bm64) MPI+OpenMP,
NUM_OMP_THREADS = 2

Execution Time (s)

No. of Processes (Skylake: 48ppn)

No. of Processes (KNL: 64ppn)

No. of Processes (KNL: 68ppn)

Courtesy: Mahidhar Tatineni @SDSC, Dong Ju (DJ) Choi@SDSC, and Samuel Khuvis@OSC ---- Testbed: TACC Stampede2 using MVAPICH2-2.3b

Runtime parameters: MV2_SMPI_LENGTH_QUEUE=524288 PSM2_MQ_RNDV_SHM_THRESH=128K PSM2_MQ_RNDV_HFI_THRESH=128K
Performance of SPEC MPI 2007 Benchmarks (KNL + Omni-Path)

Mvapich2 outperforms Intel MPI by up to 10%

448 processes on 7 KNL nodes of TACC Stampede2 (64 ppn)
Performance of SPEC MPI 2007 Benchmarks (Skylake + Omni-Path)

MVAPICH2 outperforms Intel MPI by up to 38%

480 processes on 10 Skylake nodes of TACC Stampede2 (48 ppn)
MVAPICH2 – Plans for Exascale

• Performance and Memory scalability toward 1-10M cores
• Hybrid programming (MPI + OpenSHMEM, MPI + UPC, MPI + CAF ...)
  • MPI + Task*
• Enhanced Optimization for GPU Support and Accelerators
• Taking advantage of advanced features of Mellanox InfiniBand
  • Tag Matching*
  • Adapter Memory*
• Enhanced communication schemes for upcoming architectures
  • Knights Landing with MCDRAM*
  • NVLINK*
  • CAPI*
• Extended topology-aware collectives
• Extended Energy-aware designs and Virtualization Support
• Extended Support for MPI Tools Interface (as in MPI 3.0)
• Extended FT support
• Support for * features will be available in future MVAPICH2 Releases
Thank You!

Network-Based Computing Laboratory
http://nowlab.cse.ohio-state.edu/

The MVAPICH2 Project
http://mvapich.cse.ohio-state.edu/