MVAPICH2 2.3 User Guide

MVAPICH Team

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

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

Last revised: July 23, 2018

Contents

1 Overview of the MVAPICH Project

InfiniBand, Omni-Path, Ethernet/iWARP and RDMA over Converged Ethernet (RoCE) are emerging as high-performance networking technologies to deliver low latency and high bandwidth. They are also achieving widespread acceptance due to their *open standards*.

MVAPICH (pronounced as "em-vah-pich") is an *open-source* MPI software to exploit the novel features and mechanisms of these networking technologies and deliver best performance and scalability to MPI applications. This software is developed in the Network-Based Computing Laboratory (NBCL), headed by Prof. Dhabaleswar K. (DK) Panda.

The MVAPICH2 MPI library supports MPI-3 semantics. This *open-source* MPI software project started in 2001 and a first high-performance implementation was demonstrated at SuperComputing '02 conference. After that, this software has been steadily gaining acceptance in the HPC, InfiniBand, Omni-Path, Ethernet/iWARP and RoCE communities. As of July 2018, more than 2,925 organizations (National Labs, Universities and Industry) world-wide (in 86 countries) have registered as MVAPICH users at MVAPICH project web site. There have also been more than 481,000 (0.48 million) downloads of this software from the MVAPICH project site directly. In addition, many InfiniBand, Omni-Path, Ethernet/iWARP and RoCE vendors, server vendors, systems integrators and Linux distributors have been incorporating MVAPICH2 into their software stacks and distributing it. MVAPICH2 distribution is available under BSD licensing.

Several InfiniBand systems using MVAPICH2 have obtained positions in the TOP 500 ranking. The June '18 list includes the following systems: 2nd, 10,649,600-core (Sunway TaihuLight) at National Supercomputing Center in Wuxi, China 12th, 556,104 cores (Oakforest-PACS) in Japan 15th, 367,024 cores (Stampede2) at TACC 24th, 241,108-core (Pleiades) at NASA 62nd, 76,032-core (Tsubame 2.5) at Tokyo Institute of Technology More details on MVAPICH software, users list, mailing lists, sample performance numbers on a wide range of platforms and interconnects, a set of OSU benchmarks, related publications, and other InfiniBand-, RoCE, Omni-Path, and iWARP-related projects (High-Performance Big Data and High-Performance Deep Learning) can be obtained from our website:http://mvapich.cse.ohio-state.edu.

This document contains necessary information for MVAPICH2 users to download, install, test, use, tune and troubleshoot MVAPICH2 2.3. We continuously fix bugs and update update this document as per user feedback. Therefore, we strongly encourage you to refer to our web page for updates.

2 How to use this User Guide?

This guide is designed to take the user through all the steps involved in configuring, installing, running and tuning MPI applications over InfiniBand using MVAPICH2 2.3.

In Section 3 we describe all the features in MVAPICH2 2.3. As you read through this section, please note our new features (highlighted as NEW) compared to version 2.2. Some of these features are designed in order to optimize specific type of MPI applications and achieve greater scalability. Section 4 describes in detail the configuration and installation steps. This section enables the

user to identify specific compilation flags which can be used to turn some of the features on or off. Basic usage of MVAPICH2 is explained in Section 5. Section 6 provides instructions for running MVAPICH2 with some of the advanced features. Section 7 describes the usage of the OSU Benchmarks. In Section 8 we suggest some tuning techniques for multi-thousand node clusters using some of our new features. If you have any problems using MVAPICH2, please check Section 9 where we list some of the common problems people face. Finally, in Sections 11 and 12, we list all important run time parameters, their default values and a short description.

3 MVAPICH2 2.3 Features

MVAPICH2 (MPI-3 over InfiniBand) is an MPI-3 implementation based on MPICH ADI3 layer. MVAPICH2 2.3 is available as a single integrated package (with MPICH 3.2.1). The current release supports ten different underlying transport interfaces, as shown in Figure 1.

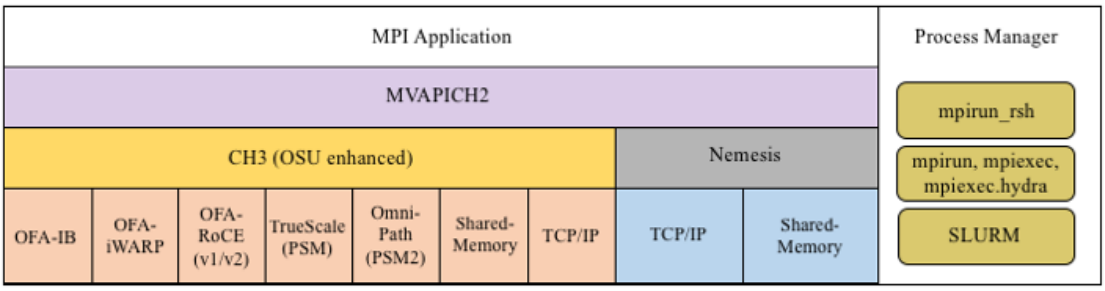

Figure 1: Overview of different available interfaces of the MVAPICH2 library

- OFA-IB-CH3: This interface supports all InfiniBand compliant devices based on the OpenFabrics layer. This interface has the most features and is most widely used. For example, this interface can be used over all Mellanox InfiniBand adapters, IBM eHCA adapters and TrueScale adapters.
- OFA-iWARP-CH3: This interface supports all iWARP compliant devices supported by OpenFabrics. For example, this layer supports Chelsio T3 adapters with the native iWARP mode.
- OFA-RoCE-CH3: This interface supports the emerging RoCE (RDMA over Converged Ethernet) interface for Mellanox ConnectX-EN adapters with 10/40GigE switches. It provides support for RoCE v1 and v2.
- TrueScale (PSM-CH3): This interface provides native support for TrueScale adapters from Intel over PSM interface. It provides high-performance point-to-point communication for both one-sided and two-sided operations.
- Omni-Path (PSM2-CH3): This interface provides native support for Omni-Path adapters from Intel over PSM2 interface. It provides high-performance point-to-point communication for both one-sided and two-sided operations.

- Shared-Memory-CH3: This interface provides native shared memory support on multi-core platforms where communication is required only within a node. Such as SMP-only systems, laptops, etc.
- TCP/IP-CH3: The standard TCP/IP interface (provided by MPICH) to work with a range of network adapters supporting TCP/IP interface. This interface can be used with IPoIB (TCP/IP over InfiniBand network) support of InfiniBand also. However, it will not deliver good performance/scalability as compared to the other interfaces.
- TCP/IP-Nemesis: The standard TCP/IP interface (provided by MPICH Nemesis channel) to work with a range of network adapters supporting TCP/IP interface. This interface can be used with IPoIB (TCP/IP over InfiniBand network) support of InfiniBand also. However, it will not deliver good performance/scalability as compared to the other interfaces.
- Shared-Memory-Nemesis: This interface provides native shared memory support on multi-core platforms where communication is required only within a node. Such as SMP-only systems, laptops, etc.
- (DEPRECATED) OFA-IB-Nemesis: This interface supports all InfiniBand compliant devices based on the OpenFabrics layer with the emerging Nemesis channel of the MPICH stack. This interface can be used by all Mellanox InfiniBand adapters.

MVAPICH2 2.3 is compliant with MPI 3 standard. In addition, MVAPICH2 2.3 provides support and optimizations for NVIDIA GPU, multi-threading and fault-tolerance (Checkpoint-restart, Job-pause-migration-resume). A complete set of features of MVAPICH2 2.3 are indicated below. New features compared to 2.2 are indicated as NEW.

- NEW Based on and ABI compatible with MPICH-3.2
- MPI-3 standard compliance
  - Nonblocking collectives
  - Neighborhood collectives
  - MPI_Comm_split_type support
  - Support for MPI_Type_create_hindexed_block
  - NEW Enhance support for MPI_T PVARs and CVARs
  - NEW Enhanced performance for Allreduce, Reduce_scatter_block, Allgather, Allgatherv through new algorithms
  - Nonblocking communicator duplication routine MPI_Comm_idup (will only work for single-threaded programs)
  - MPI_Comm_create_group support
  - Support for matched probe functionality
  - Support for "Const" (disabled by default)
- CH3-level design for scaling to multi-thousand cores with highest performance and reduced memory usage.

- Support for MPI-3 RMA in OFA-IB-CH3, OFA-IWARP-CH3, OFA-RoCE-CH3, TrueScale (PSM-CH3) and Omni-Path (PSM2-CH3)
- Support for Omni-Path architecture
  * Introduction of a new PSM2-CH3 channel for Omni-Path
- Support for OpenPOWER architecture
- NEW Support Cavium ARM (ThunderX) systems
- NEW Support Intel Skylake architecture
- Support for Intel Knights Landing architecture
- NEW Support for executing MPI jobs in Singularity
  * Optimized inter-node and intra-node communication
- Exposing several performance and control variables to MPI-3 Tools information interface (MPIT)
  * Enhanced PVAR support
- Enable support for multiple MPI initializations
- Enhanced performance for small messages
- Flexibility to use internal communication buffers of different size
- Enhanced performance for MPI_Comm_split through new bitonic algorithm
- Tuning internal communication buffer size for performance
- Improve communication performance by removing locks from critical path
- Enhanced communication performance for small/medium message sizes
- Reduced memory footprint
- Multi-rail support for UD-Hybrid channel
- Support for InfiniBand hardware UD-Multicast based collectives
- HugePage support
- Integrated Hybrid (UD-RC/XRC) design to get best performance on large-scale systems with reduced/constant memory footprint
- Support for running with UD only mode
- Support for MPI-2 Dynamic Process Management on InfiniBand Clusters
- eXtended Reliable Connection (XRC) support
  * Enable XRC by default at configure time
- Multiple CQ-based design for Chelsio 10GigE/iWARP
- Multi-port support for Chelsio 10GigE/iWARP
- Enhanced iWARP design for scalability to higher process count
- Support iWARP interoperability between Intel NE020 and Chelsio T4 adapters
- Support for 3D torus topology with appropriate SL settings
- Quality of Service (QoS) support with multiple InfiniBand SL
- NEW Capability to run MPI jobs across multiple InfiniBand subnets

- Enabling support for intra-node communications in RoCE mode without shared memory
- On-demand Connection Management: This feature enables InfiniBand connections to be setup dynamically, enhancing the scalability of MVAPICH2 on clusters of thousands of nodes.
  * Support for backing on-demand UD CM information with shared memory for minimizing memory footprint
  * Improved on-demand InfiniBand connection setup
  * On-demand connection management support with IB CM (RoCE Interface)
  * Native InfiniBand Unreliable Datagram (UD) based asynchronous connection management for OpenFabrics-IB interface.
  * RDMA CM based on-demand connection management for OpenFabrics-IB and OpenFabrics-iWARP interfaces.
  * NEW Support to automatically detect IP address of IB/RoCE interfaces when RDMA CM is enabled without relying on mv2.conf file
- Message coalescing support to enable reduction of per Queue-pair send queues for reduction in memory requirement on large scale clusters. This design also increases the small message messaging rate significantly. Available for OFA-IB-CH3 interface.
- RDMA Read utilized for increased overlap of computation and communication for OpenFabrics device. Available for OFA-IB-CH3 and OFA-IB-iWARP-CH3 interfaces.
- Shared Receive Queue (SRQ) with flow control. This design uses significantly less memory for MPI library. Available for OFA-IB-CH3 interface.
- Adaptive RDMA Fast Path with Polling Set for low-latency messaging. Available for OFA-IB-CH3 and OFA-iWARP-CH3 interfaces.
- Header caching for low-latency
- CH3 shared memory channel for standalone hosts (including SMP-only systems and laptops) without any InfiniBand adapters
- Unify process affinity support in OFA-IB-CH3, PSM-CH3 and PSM2-CH3 channels
- Support to enable affinity with asynchronous progress thread
- Allow processes to request MPI_THREAD_MULTIPLE when socket or NUMA node level affinity is specified
- Reorganized HCA-aware process mapping
- Dynamic identification of maximum read/atomic operations supported by HCA
- Enhanced scalability for RDMA-based direct one-sided communication with less communication resource. Available for OFA-IB-CH3 and OFA-iWARP-CH3 interfaces.
- Removed libibumad dependency for building the library
- Option to disable signal handler setup
- Tuned thresholds for various architectures
- Option for selecting non-default gid-index in a loss-less fabric setup in RoCE mode
- Option to use IP address as a fallback if hostname cannot be resolved
- NEW Improved job-startup performance

- NEW Gracefully handle RDMA_CM failures during job-startup
- Enhanced startup time for UD-Hybrid channel
- Provided a new runtime variable MV2_HOMOGENEOUS_CLUSTER for optimized startup on homogeneous clusters
- Improved debug messages and error reporting
- Supporting large data transfers (>2GB)

- Support for MPI communication from NVIDIA GPU device memory
  - NEW Improved performance for Host buffers when CUDA is enabled
  - Support for MPI_Scan and MPI_Exscan collective operations from GPU buffers
  - Multi-rail support for GPU communication
  - Support for non-blocking streams in asynchronous CUDA transfers for better overlap
  - Dynamic CUDA initialization. Support GPU device selection after MPI_Init
  - Support for running on heterogeneous clusters with GPU and non-GPU nodes
  - Tunable CUDA kernels for vector datatype processing for GPU communication
  - Optimized sub-array data-type processing for GPU-to-GPU communication
  - Added options to specify CUDA library paths
  - Efficient vector, hindexed datatype processing on GPU buffers
  - Tuned MPI performance on Kepler GPUs
  - Improved intra-node communication with GPU buffers using pipelined design
  - Improved inter-node communication with GPU buffers with non-blocking CUDA copies
  - Improved small message communication performance with CUDA IPC design
  - Improved automatic GPU device selection and CUDA context management
  - Optimal communication channel selection for different GPU communication modes (DD, HH and HD) in different configurations (intra-IOH a and inter-IOH)
  - Provided option to use CUDA library call instead of CUDA driver to check buffer pointer type
  - 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)
  - Enhanced designs for Alltoall and Allgather collective communication from GPU device buffers
  - Optimized and tuned support for collective communication from GPU buffers
  - Non-contiguous datatype support in point-to-point and collective communication from GPU buffers
  - Updated to sm_20 kernel optimizations for MPI Datatypes

- Taking advantage of CUDA IPC (available in CUDA 4.1) in intra-node communication for multiple GPU adapters/node
- Efficient synchronization mechanism using CUDA Events for pipelined device data transfers

- OFA-IB-Nemesis interface design NEW (Deprecated)
  - OpenFabrics InfiniBand network module support for MPICH Nemesis modular design
  - Optimized adaptive RDMA fast path with Polling Set for high-performance inter-node communication
  - Shared Receive Queue (SRQ) support with flow control, uses significantly less memory for MPI library
  - Header caching for low-latency
  - Support for additional features (such as hwloc, hierarchical collectives, one-sided, multi-threading, etc.), as included in the MPICH Nemesis channel
  - Support of Shared-Memory-Nemesis interface on multi-core platforms requiring intra-node communication only (SMP-only systems, laptops, etc.)
  - Support for 3D torus topology with appropriate SL settings
  - Quality of Service (QoS) support with multiple InfiniBand SL
  - Automatic inter-node communication parameter tuning based on platform and adapter detection
  - Flexible HCA selection
  - Checkpoint-Restart support
  - Run-through stabilization support to handle process failures
  - Enhancements to handle IB errors gracefully

- Flexible process manager support
  - Support for PMI-2 based startup with SLURM
  - Enhanced startup performance with SLURM
    - Support for PMIX_Iallgather and PMIX_Ifence
  - Enhanced startup performance and reduced memory footprint for storing InfiniBand end-point information with SLURM
    - Support for shared memory based PMI operations
  - NEW On-demand connection management for PSM-CH3 and PSM2-CH3 channels
  - Improved job startup performance with mpirun_rsh
  - Improved startup performance for TrueScale (PSM-CH3) channel
  - NEW Improved job startup time for OFA-IB-CH3, PSM-CH3, and PSM2-CH3
  - Improved hierarchical job startup performance
  - Enhanced hierarchical ssh-based robust mpirun_rsh framework to work with any interface (CH3 and Nemesis channel-based) including OFA-IB-Nemesis, TCP/IP-CH3 and TCP/IP-Nemesis to launch jobs on multi-thousand core clusters

  - Introduced option to export environment variables automatically with mpirun_rsh
  - Support for automatic detection of path to utilities(rsh, ssh, xterm, TotalView) used by mpirun_rsh during configuration
  - Support for launching jobs on heterogeneous networks with mpirun_rsh
  - MPMD job launch capability
  - Hydra process manager to work with any of the ten interfaces (CH3 and Nemesis channel-based) including OFA-IB-CH3, OFA-iWARP-CH3, OFA-RoCE-CH3 and TCP/IP-CH3
  - Improved debug message output in process management and fault tolerance functionality
  - Better handling of process signals and error management in mpispawn
  - Flexibility for process execution with alternate group IDs
  - Using in-band IB communication with MPD
  - SLURM integration with mpiexec.mpirun_rsh to use SLURM allocated hosts without specifying a hostfile
  - Support added to automatically use PBS_NODEFILE in Torque and PBS environments
  - Support for suspend/resume functionality with mpirun_rsh framework
  - Exporting local rank, local size, global rank and global size through environment variables (both mpirun_rsh and hydra)
- Support for various job launchers and job schedulers (such as SGE and OpenPBS/Torque)
- Configuration file support (similar to one available in MVAPICH). Provides a convenient method for handling all runtime variables through a configuration file.
- Fault-tolerance support
  - Checkpoint-Restart Support with DMTCP (Distributed MultiThreaded CheckPointing)
  - Enable hierarchical SSH-based startup with Checkpoint-Restart
  - Enable the use of Hydra launcher with Checkpoint-Restart for OFA-IB-CH3 and OFA-IB-Nemesis interfaces
  - Checkpoint/Restart using LLNL's Scalable Checkpoint/Restart Library (SCR)
    - Support for application-level checkpointing
    - Support for hierarchical system-level checkpointing
  - Checkpoint-restart support for application transparent systems-level fault tolerance. BLCR-based support using OFA-IB-CH3 and OFA-IB-Nemesis interfaces
    - Scalable Checkpoint-restart with mpirun_rsh framework
    - Checkpoint-restart with Fault-Tolerance Backplane (FTB) framework (FTB-CR)
    - Checkpoint-restart with intra-node shared memory (user-level) support
    - Checkpoint-restart with intra-node shared memory (kernel-level with LiMIC2) support
    - Checkpoint-restart support with pure SMP mode
    - Allows best performance and scalability with fault-tolerance support

    - Run-through stabilization support to handle process failures using OFA-IB-Nemesis interface
    - Enhancements to handle IB errors gracefully using OFA-IB-Nemesis interface
  - Application-initiated system-level checkpointing is also supported. User application can request a whole program checkpoint synchronously by calling special MVAPICH2 functions.
    - Flexible interface to work with different files systems. Tested with ext3 (local disk), NFS and PVFS2.
  - Network-Level fault tolerance with Automatic Path Migration (APM) for tolerating intermittent network failures over InfiniBand.
  - Fast Checkpoint-Restart support with aggregation scheme
  - Job Pause-Migration-Restart Framework for Pro-active Fault-Tolerance
    - Enable signal-triggered (SIGUSR2) migration
  - Fast process migration using RDMA
  - Support for new standardized Fault Tolerant Backplane (FTB) Events for Checkpoint-Restart and Job Pause-Migration-Restart Framework

- Enhancement to software installation
  - Revamped Build system
    - Uses automake instead of simplemake,
    - Allows for parallel builds ("make -j8" and similar)
  - Full autoconf-based configuration
  - Automatically detects system architecture and adapter types and optimizes MVAPICH2 for any particular installation.
  - A utility (mpiname) for querying the MVAPICH2 library version and configuration information
  - Automatically builds and installs OSU Benchmarks for end-user convenience

- Optimized intra-node communication support by taking advantage of shared-memory communication. Available for all interfaces (IB and iWARP).
  - Enhanced intra-node SMP performance
  - Tuned SMP eager threshold parameters
  - New shared memory design for enhanced intra-node small message performance
  - Support for single copy intra-node communication using Linux supported CMA (Cross Memory Attach)
    - Enabled by default
  - Kernel-level single-copy intra-node communication solution based on LiMIC2
    - Upgraded to LiMIC2 version 0.5.6 to support unlocked ioctl calls
    - LiMIC2 is designed and developed by jointly by The Ohio State University and System Software Laboratory at Konkuk University, Korea.

- Efficient Buffer Organization for Memory Scalability of Intra-node Communication
- Multi-core optimized
- Adjust shared-memory communication block size at runtime
- NEW Enhanced intra-node and inter-node tuning for PSM-CH3 and PSM2-CH3 channels
- NEW Added logic to detect heterogeneous CPU/HFI configurations in PSM-CH3 and PSM2-CH3 channels
- Automatic intra-node communication parameter tuning based on platform
- Efficient connection set-up for multi-core systems
- NEW Portable Hardware Locality (hwloc v1.11.9) support for defining CPU affinity
- NEW Efficient CPU binding policies (spread, bunch, and scatter) to specify CPU binding per job for modern multi-core platforms with SMT support
- Enhanced support for CPU binding with socket and numanode level granularity
- Enhance MV2_SHOW_CPU_BINDING to enable display of CPU bindings on all nodes
- Improve performance of architecture detection
- Enhance process mapping support for multi-threaded MPI applications
  * NEW Introduce MV2_CPU_BINDING_POLICY=hybrid
  * NEW Introduce MV2_HYBRID_BINDING_POLICY
  * NEW Introduce MV2_THREADS_PER_PROCESS
- Improved usability of process to CPU mapping with support of delimiters (',' , '-') in CPU listing
- Also allows user-defined CPU binding
- Optimized for Bus-based SMP and NUMA-Based SMP systems.
- Efficient support for diskless clusters

- Optimized collective communication operations. Available for OFA-IB-CH3, OFA-iWARP-CH3, and OFA-RoCE-CH3 interfaces
  - NEW Enhanced small message performance for Alltoallv
  - NEW Support collective offload using Mellanox's SHArP for Allreduce
  - NEW Enhance collective tuning for OpenPOWER (POWER8 and POWER9), Intel Skylake and Cavium ARM (ThunderX) systems
  - NEW Enhance collective tuning for Intel Knights Landing and Intel Omni-path
  - Optimized collectives (bcast, reduce, and allreduce) for 4K processes
  - Optimized and tuned blocking and non-blocking collectives for OFA-IB-CH3, OFA-IB-Nemesis and TrueScale (PSM-CH3) channels
  - Enhanced MPI_Bcast, MPI_Reduce, MPI_Scatter, MPI_Gather performance
  - Hardware UD-Multicast based designs for collectives - Bcast, Allreduce and Scatter
  - Intra-node Zero-Copy designs for MPI_Gather collective (using LiMIC2)
  - Enhancements and optimizations for point-to-point designs for Broadcast, Allreduce collectives

  - Improved performance for shared-memory based collectives - Broadcast, Barrier, Allreduce, Reduce
  - Performance improvements in Scatterv and Gatherv collectives for CH3 interface
  - Enhancements and optimizations for collectives (Alltoallv, Allgather)
  - Tuned Bcast, alltoall, Scatter, Allgather, Allgatherv, Reduce, Reduce_Scatter, Allreduce collectives

- Integrated multi-rail communication support. Available for OFA-IB-CH3 and OFA-iWARP-CH3 interfaces.
  - Supports multiple queue pairs per port and multiple ports per adapter
  - Supports multiple adapters
  - Support to selectively use some or all rails according to user specification
  - Support for both one-sided and point-to-point operations
  - Reduced stack size of internal threads to dramatically reduce memory requirement on multi-rail systems
  - Dynamic detection of multiple InfiniBand adapters and using these by default in multi-rail configurations (OFA-IB-CH3, OFA-iWARP-CH3 and OFA-RoCE-CH3 interfaces)
  - Support for process-to-rail binding policy (bunch, scatter and user-defined) in multi-rail configurations (OFA-IB-CH3, OFA-iWARP-CH3 and OFA-RoCE-CH3 interfaces)
  - NEW Enhance HCA detection to handle cases where node has both IB and RoCE HCAs

- Support for InfiniBand Quality of Service (QoS) with multiple lanes

- Multi-threading support. Available for all interfaces (IB and iWARP), including TCP/IP.
  - Enhanced support for multi-threaded applications

- High-performance optimized and scalable support for one-sided communication: Put, Get and Accumulate. Supported synchronization calls: Fence, Active Target, Passive (lock and unlock). Available for all interfaces.
  - Support for handling very large messages in RMA
  - Enhanced direct RDMA based designs for MPI_Put and MPI_Get operations in OFA-IB-CH3 channel
  - Optimized communication when using MPI_Win_allocate for OFA-IB-CH3 channel
  - Direct RDMA based One-sided communication support for OpenFabrics Gen2-iWARP and RDMA CM (with Gen2-IB)
  - Shared memory backed Windows for one-sided communication

- Two modes of communication progress
  - Polling
  - Blocking (enables running multiple MPI processes/processor). Available for OpenFabrics (IB and iWARP) interfaces.

- Advanced AVL tree-based Resource-aware registration cache
- Adaptive number of registration cache entries based on job size
- Automatic detection and tuning for 24-core Haswell architecture
- Automatic detection and tuning for 28-core Broadwell architecture
- Automatic detection and tuning for Intel Knights Landing architecture
- Automatic tuning based on both platform type and network adapter
- Remove verbs dependency when building the PSM-CH3 and PSM2-CH3 channels
- Progress engine optimization for TrueScale (PSM-CH3) interface
- Improved performance for medium size messages for TrueScale (PSM-CH3) channel
- Multi-core-aware collective support for TrueScale (PSM-CH3) channel
- Collective optimization for TrueScale (PSM-CH3) channel
- Memory Hook Support provided by integration with ptmalloc2 library. This provides safe release of memory to the Operating System and is expected to benefit the memory usage of applications that heavily use malloc and free operations.
- Warn and continue when ptmalloc fails to initialize
- Support for TotalView debugger with mpirun_rsh framework
- Support for linking Intel Trace Analyzer and Collector
- Shared library support for existing binary MPI application programs to run.
- Enhanced debugging config options to generate core files and back-traces
- Use of gfortran as the default F77 compiler
- ROMIO Support for MPI-IO.
  - Optimized, high-performance ADIO driver for Lustre
- Single code base for the following platforms (Architecture, OS, Compilers, Devices and InfiniBand adapters)
  - Architecture: Knights Landing, OpenPOWER(POWER8 and POWER9), ARM, EM64T, x86_64 and x86
  - Operating Systems: (tested with) Linux
  - Compilers: GCC, Intel, PGI, and Open64
  - Devices: OFA-IB-CH3, OFA-iWARP-CH3, OFA-RoCE-CH3, TrueScale (PSM-CH3), Omni-Path (PSM2-CH3), TCP/IP-CH3, OFA-IB-Nemesis and TCP/IP-Nemesis
  - InfiniBand adapters (tested with):

    - Mellanox InfiniHost adapters (SDR and DDR)
    - Mellanox ConnectX (DDR and QDR with PCIe2)
    - Mellanox ConnectX-2 (QDR with PCIe2)
    - Mellanox ConnectX-3 (FDR with PCIe3)
    - Mellanox Connect-IB (Dual FDR ports with PCIe3)
    - Mellanox Connect-4 (EDR with PCIe3)
    - Mellanox Connect-5 (EDR with PCIe3)
    - Intel TrueScale adapter (SDR)
    - Intel TrueScale adapter (DDR and QDR with PCIe2)
  - Intel Omni-Path adapters (tested with):
    - Intel Omni-Path adapter (100 Gbps with PCIe3)
  - 10GigE (iWARP and RoCE) adapters:
    - (tested with) Chelsio T3 and T4 adapter with iWARP support
    - (tested with) Mellanox ConnectX-EN 10GigE adapter
    - (tested with) Intel NE020 adapter with iWARP support
  - 40GigE RoCE adapters:
    - (tested with) Mellanox ConnectX-EN 40GigE adapter

The MVAPICH2 2.3 package and the project also includes the following provisions:

- Public SVN access of the code-base
- A set of micro-benchmarks (including multi-threading latency test) for carrying out MPI-level performance evaluation after the installation
- Public mvapich-discuss mailing list for mvapich users to
  - Ask for help and support from each other and get prompt response
  - Enable users and developers to contribute patches and enhancements

4 Installation Instructions

The MVAPICH2 installation process is designed to enable the most widely utilized features on the target build OS by default. The other interfaces, as indicated in Figure 1, can also be selected on Linux. This installation section provides generic instructions for building from a tarball or our latest sources.

In order to obtain best performance and scalability while having flexibility to use a large number of features, the MVAPICH team strongly recommends the use of following interfaces for different adapters: 1) OFA-IB-CH3 interface for all Mellanox InfiniBand adapters, 2) TrueScale (PSM-CH3) interface for all Intel InfiniBand adapters, 3) OFA-RoCE-CH3 interface for all RoCE adapters, 4) OFA-iWARP-CH3 for all iWARP adapters and 5) Shared-Memory-CH3 for single node SMP system and laptop.

Please see the appropriate subsection for specific configuration instructions for the interface-adapter you are targeting.

4.1 Building from a tarball

The MVAPICH2 2.3 source code package includes MPICH 3.2.1. All the required files are present as a single tarball. Download the most recent distribution tarball from:
http://mvapich.cse.ohio-state.edu/downloads

Unpack the tarball and use the standard GNU procedure to compile:

```
 $ tar -xzf mvapich2-2.3.tgz
$ cd mvapich2-2.3
$ ./configure
$ make
$ make install
```

We now support parallel make and you can use the -j<num threads> option to speed up the build process. You can use the following example to spawn 4 threads instead of the preceding make step.

```
$ make -j 4
```

In order to install a debug build, please use the following configuration option. *Please note that using debug builds may impact performance.*

```
 $ ./configure --enable-g=all --enable-error-messages=all
$ make
$ make install
```

4.2 Obtaining and Building the Source from SVN repository

These instructions assume you have already installed subversion.

The MVAPICH2 SVN repository is available at:
https://scm.nowlab.cse.ohio-state.edu/svn/mpi/mvapich2/

Please keep in mind the following guidelines before deciding which version to check out:

- "tags/2.3" is the exact version released with no updates for bug fixes or new features.
  - To obtain the source code from tags/2.3:
    ```
    $ svn co
    https://scm.nowlab.cse.ohio-state.edu/svn/mpi/mvapich2/tags/2.3
    mvapich2
    ```
- "trunk" will contain the latest source code as we enhance and improve MVAPICH2. It may contain newer features and bug fixes, but is lightly tested.
  - To obtain the source code from trunk:
    ```
    $ svn co https://scm.nowlab.cse.ohio-state.edu/svn/mpi/mvapich2/trunk
    mvapich2
    ```

The mvapich2 directory under your present working directory contains a working copy of the MVAPICH2 source code. Now that you have obtained a copy of the source code, you need to update the files in the source tree:

```
$ cd mvapich2
$ ./autogen.sh
```

This script will generate all of the source and configuration files you need to build MVAPICH2. You will need `autoconf` version >= 2.67, `automake` version >= 1.12.3, `libtool` version >= 2.4

```
$ ./configure
$ make
$ make install
```

4.3 Selecting a Process Manager

MVAPICH2 provides the mpirun_rsh/mpispawn framework from MVAPICH distribution. Using mpirun_rsh should provide the fastest startup of your MPI jobs. More details can be found in Section 5.2.1. In addition, MVAPICH2 also includes the Hydra process manager from MPICH-3.2.1. For more details on using Hydra, please refer to Section 5.2.2.

By default, *mpiexec* uses the Hydra process launcher. Please note that neither mpirun_rsh, nor Hydra require you to start daemons in advance on the nodes used for a MPI job. Both mpirun_rsh and Hydra can be used with any of the eight interfaces of this MVAPICH2 release, as indicated in Figure 1.

4.3.1 Customizing Commands Used by mpirun_rsh

Usage: ./configure [OPTION]... [VAR=VALUE]...

To assign environment variables (e.g., CC, CFLAGS...), specify them as VAR=VALUE. See below for descriptions of some of the useful variables.

RSH_CMD	path to rsh command
SSH_CMD	path to ssh command
ENV_CMD	path to env command
DBG_CMD	path to debugger command
XTERM_CMD	path to xterm command
SHELL_CMD	path to shell command
TOTALVIEW_CMD	path to totalview command

4.3.2 Using SLURM

If you'd like to use SLURM to launch your MPI programs please use the following configure options.

To configure MVAPICH2 to use PMI-1 support in SLURM:

```
$ ./configure --with-pmi=pmi1 --with-pm=slurm
```

To configure MVAPICH2 to use PMI-2 support in SLURM:

```
$ ./configure --with-pmi=pmi2 --with-pm=slurm
```

4.3.3 Using SLURM with support for PMI Extensions

MVAPICH2 automatically detects and uses PMI extensions if available from the process manager. To build and install SLURM with PMIX support, please follow these steps:

Download the SLURM source tarball for SLURM-15.08.8 from
http://slurm.schedmd.com/download.html.

Download the patch to add PMI Extensions in SLURM from
http://mvapich.cse.ohio-state.edu/download/mvapich/osu-shmempmi-slurm-15.08.8.patch.

```
$ tar -xzf slurm-15.08.8.tar.gz
$ cd slurm-15.08.8
$ patch -p1 < osu-shmempmi-slurm-15.08.8.patch
$ ./configure --prefix=/path/to/slurm/install
--disable-pam
$ make -j4 && make install && make install-contrib
```

To configure MVAPICH2 with the modified SLURM, please use:

```
$ ./configure --with-pm=slurm --with-pmi=pmi2
--with-slurm=/path/to/slurm/install
```

Please refer to Section 5.2.3 for information on how to run MVAPICH2 using SLURM.

4.4 Configuring a build for OFA-IB-CH3/OFA-iWARP-CH3/OFA-RoCE-CH3

OpenFabrics (OFA) IB/iWARP/RoCE with the CH3 channel is the default interface on Linux. It can be explicitly selected by configuring with:

```
$ ./configure --with-device=ch3:mrail --with-rdma=gen2
```

Both static and shared libraries are built by default. In order to build with static libraries only, configure as follows:

```
$ ./configure --with-device=ch3:mrail --with-rdma=gen2
--disable-shared
```

To enable use of the TotalView debugger, the library needs to be configured in the following manner:

```
$ ./configure --with-device=ch3:mrail --with-rdma=gen2 --enable-g=dbg
--enable-debuginfo
```

Configuration Options for OpenFabrics IB/iWARP/RoCE

- Configuring with Shared Libraries
  - Default: Enabled
  - Enable: `--enable-shared`
  - Disable: `--disable-shared`
- Configuring with TotalView support
  - Default: Disabled
  - Enable: `--enable-g=dbg --enable-debuginfo`
- Path to OpenFabrics Header Files
  - Default: Your PATH
  - Specify: `--with-ib-include=path`
- Path to OpenFabrics Libraries
  - Default: The systems search path for libraries.
  - Specify: `--with-ib-libpath=path`
- Support for Hybrid UD-RC/XRC transports
  - Default: Disabled
  - Enable: `--enable-hybrid`
- Support for RDMA CM
  - Default: enabled, except when BLCR support is enabled

  - Disable: `--disable-rdma-cm`

- Support for RoCE
  - Default: enabled

- Registration Cache
  - Default: enabled
  - Disable: `--disable-registration-cache`

- ADIO driver for Lustre:
  - When compiled with this support, MVAPICH2 will use the optimized driver for Lustre. In order to enable this feature, the flag
    `--enable-romio --with-file-system=lustre`
    should be passed to `configure` (`--enable-romio` is optional as it is enabled by default). You can add support for more file systems using
    `--enable-romio --with-file-system=lustre+nfs+pvfs2`

- LiMIC2 Support
  - Default: disabled
  - Enable:
    `--with-limic2[=<path to LiMIC2 installation>]`
    `--with-limic2-include=<path to LiMIC2 headers>`
    `--with-limic2-libpath=<path to LiMIC2 library>`

- CMA Support
  - Default: enabled
  - Disable: `--without-cma`

- Header Caching
  - Default: enabled
  - Disable: `--disable-header-caching`

- MPI Tools Information Interface (MPI-T) Support
  - Default: disabled
  - Enable: `--enable-mpit-pvars`

- Checkpoint/Restart
  - Option name: `--enable-ckpt`
  - Require: Berkeley Lab Checkpoint/Restart (BLCR)
  - Default: disabled

The Berkeley Lab Checkpoint/Restart (BLCR) installation is automatically detected if installed in the standard location. To specify an alternative path to the BLCR installation, you can either use:

```
--with-blcr=<path/to/blcr/installation>
```
or
```
--with-blcr-include=<path/to/blcr/headers>
--with-blcr-libpath=<path/to/blcr/library>
```

- Checkpoint Aggregation
  - Option name: `--enable-ckpt-aggregation` or `--disable-ckpt-aggregation`
  - Automatically enable Checkpoint/Restart
  - Require: Filesystem in Userspace (FUSE)
  - Default: enabled (if Checkpoint/Restart enabled and FUSE is present)

  The Filesystem in Userspace (FUSE) installation is automatically detected if installed in the standard location. To specify an alternative path to the FUSE installation, you can either use:

  ```
  --with-fuse=<path/to/fuse/installation>
  ```
  or
  ```
  --with-fuse-include=<path/to/fuse/headers>
  --with-fuse-libpath=<path/to/fuse/library>
  ```

- Application-Level and Transparent System-Level Checkpointing with SCR
  - Option name: `--with-scr`
  - Default: disabled

  SCR caches checkpoint data in storage on the compute nodes of a Linux cluster to provide a fast, scalable checkpoint / restart capability for MPI codes.

- Process Migration
  - Option name: `--enable-ckpt-migration`
  - Automatically enable Checkpoint/Restart
  - Require: Fault Tolerance Backplane (FTB)
  - Default: disabled

  The Fault Tolerance Backplane (FTB) installation is automatically detected if installed in the standard location. To specify an alternative path to the FTB installation, you can either use:

  ```
  --with-ftb=<path/to/ftb/installation>
  ```
  or
  ```
  --with-ftb-include=<path/to/ftb/headers>
  --with-ftb-libpath=<path/to/ftb/library>
  ```

- eXtended Reliable Connection
  - Default: enabled (if OFED installation supports it)
  - Enable: `--enable-xrc`
  - Disable: `--disable-xrc`
- HWLOC Support (Affinity)
  - Default: enabled
  - Disable: `--without-hwloc`
- Support for 64K or greater number of cores
  - Default: 64K or lower number of cores
  - Enable: `--with-ch3-rank-bits=32`

4.5 Configuring a build for NVIDIA GPU with OFA-IB-CH3

This section details the configuration option to enable GPU-GPU communication with the OFA-IB-CH3 interface of the MVAPICH2 MPI library. For more options on configuring the OFA-IB-CH3 interface, please refer to Section 4.4.

- Default: disabled
- Enable: `--enable-cuda`

The CUDA installation is automatically detected if installed in the standard location. To specify an alternative path to the CUDA installation, you can either use:
`--with-cuda=<path/to/cuda/installation>`
or
`--with-cuda-include=<path/to/cuda/include>`
`--with-cuda-libpath=<path/to/cuda/libraries>`

In addition to these we have added the following variables to help account for libraries being installed in different locations:
`--with-libcuda=<path/to/directory/containing/libcuda>`
`--with-libcudart=<path/to/directory/containing/libcudart`

Note: If using the PGI compiler, you will need to add the following to your `CPPFLAGS` and `CFLAGS`. You'll also need to use the `--enable-cuda=basic` configure option to build properly. See the example below.

Example:
```
./configure --enable-cuda=basic CPPFLAGS="-D__x86_64
    -D__align__\(n\)=__attribute__\(\(aligned\(n\)\)\)
    -D__location__\(a\)=__annotate__\(a\)
    -DCUDARTAPI="
    CFLAGS="-ta=tesla:nordc"
```

4.6 Configuring a build to support running jobs across multiple InfiniBand subnets

The support for running jobs across multiple subnets in MVAPICH2 can be enabled at configure time as follows:

```
$ ./configure --enable-multi-subnet
```

MVAPICH2 relies on RDMA_CM module to establish connections with peer processes. The RDMA_CM modules shipped some older versions of OFED (like OFED-1.5.4.1), do not have the necessary support to enable communication across multiple subnets. MVAPICH2 is capable of automatically detecting such OFED installations at configure time. If the OFED installation present on the system does not support running across multiple subnets, the configure step will detect this and exit with an error message.

4.7 Configuring a build for Shared-Memory-CH3

The default CH3 channel provides native support for shared memory communication on stand alone multi-core nodes that are not equipped with InfiniBand adapters. The steps to configure CH3 channel explicitly can be found in Section 4.4. Dynamic Process Management (5.2.5) is currently not supported on stand-alone nodes without InfiniBand adapters.

4.8 Configuring a build for OFA-IB-Nemesis

The Nemesis sub-channel for OFA-IB is now deprecated. It can be built with:

```
$ ./configure --with-device=ch3:nemesis:ib
```

Both static and shared libraries are built by default. In order to build with static libraries only, configure as follows:

```
$ ./configure --with-device=ch3:nemesis:ib --disable-shared
```

To enable use of the TotalView debugger, the library needs to be configured in the following manner:

```
$ ./configure --with-device=ch3:nemesis:ib  --enable-g=dbg
--enable-debuginfo
```

Configuration options for OFA-IB-Nemesis:

- Configuring with Shared Libraries
  - Default: Enabled
  - Enable: `--enable-shared`
  - Disable: `--disable-shared`
- Configuring with TotalView support

  - Default: Disabled
  - Enable: `--enable-g=dbg`
    `--enable-debuginfo`

- Path to IB Verbs
  - Default: System Path
  - Specify: `--with-ibverbs=<path>` or
    `--with-ibverbs-include=<path>` and `--with-ibverbs-lib=<path>`

- Registration Cache
  - Default: enabled
  - Disable: `--disable-registration-cache`

- Header Caching
  - Default: enabled
  - Disable: `--disable-header-caching`

- Support for 64K or greater number of cores
  - Default: 64K or lower number of cores
  - Enable: `--with-ch3-rank-bits=32`

- Checkpoint/Restart
  - Default: disabled
  - Enable: `--enable-checkpointing` and `--with-hydra-ckpointlib=blcr`
  - Require: Berkeley Lab Checkpoint/Restart (BLCR)

  The Berkeley Lab Checkpoint/Restart (BLCR) installation is automatically detected if installed in the standard location. To specify an alternative path to the BLCR installation, you can either use:
  `--with-blcr=<path/to/blcr/installation>`
  or
  `--with-blcr-include=<path/to/blcr/headers>`
  `--with-blcr-libpath=<path/to/blcr/library>`

4.9 Configuring a build for Intel TrueScale (PSM-CH3)

The TrueScale (PSM-CH3) interface needs to be built to use MVAPICH2 on Intel TrueScale adapters. It can built with:

```
$ ./configure --with-device=ch3:psm
```

Both static and shared libraries are built by default. In order to build with static libraries only, configure as follows:

```
$ ./configure --with-device=ch3:psm --disable-shared
```

To enable use of the TotalView debugger, the library needs to be configured in the following manner:

```
$ ./configure --with-device=ch3:psm  --enable-g=dbg --enable-debuginfo
```

Configuration options for Intel TrueScale PSM channel:

- Configuring with Shared Libraries
  - Default: Enabled
  - Enable: `--enable-shared`
  - Disable: `--disable-shared`
- Configuring with TotalView support
  - Default: Disabled
  - Enable: `--enable-g=dbg  --enable-debuginfo`
- Path to Intel TrueScale PSM header files
  - Default: The systems search path for header files
  - Specify: `--with-psm-include=path`
- Path to Intel TrueScale PSM library
  - Default: The systems search path for libraries
  - Specify: `--with-psm-lib=path`
- Support for 64K or greater number of cores
  - Default: 64K or lower number of cores
  - Enable: `--with-ch3-rank-bits=32`

4.10 Configuring a build for Intel Omni-Path (PSM2-CH3)

The Omni-Path (PSM2-CH3) interface needs to be built to use MVAPICH2 on Intel Omni-Path adapters. It can built with:

```
$ ./configure --with-device=ch3:psm
```

Both static and shared libraries are built by default. In order to build with static libraries only, configure as follows:

```
$ ./configure --with-device=ch3:psm --disable-shared
```

To enable use of the TotalView debugger, the library needs to be configured in the following manner:

```
$ ./configure --with-device=ch3:psm  --enable-g=dbg --enable-debuginfo
```

Configuration options for Intel Omni-Path PSM2 channel:

- Configuring with Shared Libraries
  - Default: Enabled
  - Enable: `--enable-shared`
  - Disable: `--disable-shared`
- Configuring with TotalView support
  - Default: Disabled
  - Enable: `--enable-g=dbg  --enable-debuginfo`
- Path to Intel Omni-Path PSM2 header files
  - Default: The systems search path for header files
  - Specify: `--with-psm2-include=path`
- Path to Intel Omni-Path PSM2 library
  - Default: The systems search path for libraries
  - Specify: `--with-psm2-lib=path`
- Support for 64K or greater number of cores
  - Default: 64K or lower number of cores
  - Enable: `--with-ch3-rank-bits=32`

4.11 Configuring a build for TCP/IP-Nemesis

The use of TCP/IP with Nemesis channel requires the following configuration:

```
$ ./configure --with-device=ch3:nemesis
```

Both static and shared libraries are built by default. In order to build with static libraries only, configure as follows:

```
$ ./configure --with-device=ch3:nemesis --disable-shared
```

To enable use of the TotalView debugger, the library needs to be configured in the following manner:

```
$ ./configure --with-device=ch3:nemesis  --enable-g=dbg
--enable-debuginfo
```

Additional instructions for configuring with TCP/IP-Nemesis can be found in the MPICH documentation available at: http://www.mcs.anl.gov/research/projects/mpich2/documentation/index.php?s=docs

- Configuring with Shared Libraries
  - Default: Enabled
  - Enable: `--enable-shared`
  - Disable: `--disable-shared`
- Configuring with TotalView support
  - Default: Disabled
  - Enable: `--enable-g=dbg --enable-debuginfo`
- Support for 64K or greater number of cores
  - Default: 64K or lower number of cores
  - Enable: `--with-ch3-rank-bits=32`

4.12 Configuring a build for TCP/IP-CH3

The use of TCP/IP requires the explicit selection of a TCP/IP enabled channel. The recommended channel is TCP/IP Nemesis (described in Section 4.13). The alternative ch3:sock channel can be selected by configuring with:

```
$ ./configure --with-device=ch3:sock
```

Both static and shared libraries are built by default. In order to build with static libraries only, configure as follows:

```
$ ./configure --with-device=ch3:sock --disable-shared
```

To enable use of the TotalView debugger, the library needs to be configured in the following manner:

```
$ ./configure --with-device=ch3:sock  --enable-g=dbg --enable-debuginfo
```

- Configuring with Shared Libraries
  - Default: Enabled
  - Enable: `--enable-shared`
  - Disable: `--disable-shared`
- Configuring with TotalView support
  - Default: Disabled
  - Enable: `--enable-g=dbg --enable-debuginfo`
- Support for 64K or greater number of cores

  - Default: 64K or lower number of cores
  - Enable: `--with-ch3-rank-bits=32`

Additional instructions for configuring with TCP/IP can be found in the MPICH documentation available at:

http://www.mpich.org/documentation/guides/

4.13 Configuring a build for OFA-IB-Nemesis and TCP/IP Nemesis (unified binary)

MVAPICH2 supports a unified binary for both OFA and TCP/IP communication through the Nemesis interface.

In order to configure MVAPICH2 for unified binary support, perform the following steps:

```
$ ./configure --with-device=ch3:nemesis:ib,tcp
```

You can use mpicc as usual to compile MPI applications. In order to run your application on OFA:

```
$ mpiexec -f hosts ./a.out -n 2
```

To run your application on TCP/IP:

```
$ MPICH_NEMESIS_NETMOD=tcp mpiexec -f hosts ./osu_latency -n 2
```

4.14 Configuring a build for Shared-Memory-Nemesis

The use of Nemesis shared memory channel requires the following configuration.

```
$ ./configure --with-device=ch3:nemesis
```

Both static and shared libraries are built by default. In order to build with static libraries only, configure as follows:

```
$ ./configure --with-device=ch3:nemesis --disable-shared
```

To enable use of the TotalView debugger, the library needs to be configured in the following manner:

```
$ ./configure --with-device=ch3:nemesis  --enable-g=dbg
--enable-debuginfo
```

Additional instructions for configuring with Shared-Memory-Nemesis can be found in the MPICH documentation available at:
http://www.mcs.anl.gov/research/projects/mpich2/documentation/index.php?s=docs

- Configuring with Shared Libraries
  - Default: Enabled

  - Enable: `--enable-shared`
  - Disable: `--disable-shared`

- Configuring with TotalView support
  - Default: Disabled
  - Enable: `--enable-g=dbg --enable-debuginfo`

4.15 Configuration and Installation with Singularity

MVAPICH2 can be configured and installed with Singularity in the following manner. Note that the following prerequisites must be fulfilled before this step.

- Singularity must be installed and operational
- Singularity image must be created as appropriate

Sample Configuration and Installation with Singularity

```
# Clone the MVAPICH2 in current directory (on host)
$ git@scm.mvapich.cse.ohio-state.edu:mvapich2.git
$ cd mvapich2

# Build MVAPICH2 in the working directory,
# using the tool chain within the container
$ singularity exec Singularity-Centos-7.img ./autogen.sh
$ singularity exec Singularity-Centos-7.img ./configure --prefix=/usr/local
$ singularity exec Singularity-Centos-7.img make

# Install MVAPICH2 into the container as root and writeable option
$ sudo singularity exec -w Singularity-Centos-7.img make install
```

5 Basic Usage Instructions

5.1 Compile Applications

MVAPICH2 provides a variety of MPI compilers to support applications written in different programming languages. Please use `mpicc, mpif77, mpiCC`, or `mpif90` to compile applications. The correct compiler should be selected depending upon the programming language of your MPI application.

These compilers are available in the `MVAPICH2_HOME/bin` directory. MVAPICH2 installation directory can also be specified by modifying `$PREFIX`, then all the above compilers will also be present in the `$PREFIX/bin` directory.

5.2 Run Applications

This section provides instructions on how to run applications with MVAPICH2. Please note that on new multi-core architectures, process-to-core placement has an impact on performance. Please refer to Section 6.5 to learn about running MVAPICH2 library on multi-core nodes.

5.2.1 Run using `mpirun_rsh`

The MVAPICH team suggests users using this mode of job start-up for all interfaces (including OFA-IB-CH3, OFA-IB-Nemesis, OFA-iWARP-CH3, OFA-RoCE-CH3, TrueScale (PSM-CH3), Omni-Path (PSM2-CH3), Shared memory-CH3, TCP/IP-CH3 and TCP/IP-Nemesis) This ***mpirun_rsh*** *scheme provides fast and scalable job start-up. It scales to multi-thousand node clusters.*

Prerequisites:

- Either `ssh` or `rsh` should be enabled between the front nodes and the computing nodes. In addition to this setup, you should be able to login to the remote nodes without any password prompts.
- All host names should resolve to the same IP address on all machines. For instance, if a machine's host names resolves to 127.0.0.1 due to the default /etc/hosts on some Linux distributions it leads to incorrect behavior of the library.

Examples of running programs using `mpirun_rsh`:

```
$ mpirun_rsh -np 4 n0 n0 n1 n1 ./cpi
```

This command launches `cpi` on nodes n0 and n1, two processes per node. By default `ssh` is used.

```
$ mpirun_rsh -rsh -np 4 n0 n0 n1 n1 ./cpi
```

This command launches `cpi` on nodes n0 and n1, two processes per each node using `rsh` instead of `ssh`.

MPIRUN_RSH Hostfile:

```
$ mpirun_rsh -np 4 -hostfile hosts ./cpi
```

A list of target nodes must be provided in the file `hosts` one per line. MPI ranks are assigned in order of the hosts listed in the hosts file or in the order they are passed to mpirun_rsh. i.e., if the nodes are listed as n0 n1 n0 n1, then n0 will have two processes, rank 0 and rank 2; whereas n1 will have rank 1 and 3. This rank distribution is known as "cyclic". If the nodes are listed as n0 n0 n1 n1, then n0 will have ranks 0 and 1; whereas n1 will have ranks 2 and 3. This rank distribution is known as "block".

Hostfile Format

The mpirun_rsh hostfile format allows for users to specify hostnames, one per line, optionally with a multiplier, and HCA specification.

The multiplier allows you to save typing by allowing you to specify blocked distribution of MPI ranks using one line per hostname. The HCA specification allows you to force an MPI rank to use a particular HCA.

The optional components are delimited by a ':'. Comments and empty lines are also allowed. Comments start with '#' and continue to the next newline.

Sample hostfile

```
$ cat hosts
# sample hostfile for mpirun_rsh
host1           # rank 0 will be placed on host1
host2:2         # rank 1 and 2 will be placed on host 2
host3:hca1      # rank 3 will be on host3 and will use hca1
host4:4:hca2    # ranks 4 through 7 will be on host4 and use hca2

# if the number of processes specified for this job is greater than 8
# then the additional ranks will be assigned to the hosts in a cyclic
# fashion.  For example, rank 8 will be on host1 and ranks 9 and 10 will
# be on host2.
```

Specifying Environmental Variables

Many parameters of the MPI library can be configured at run-time using environmental variables. In order to pass any environment variable to the application, simply put the variable names and values just before the executable name, like in the following example:

```
$ mpirun_rsh -np 4 -hostfile hosts ENV1=value ENV2=value ./cpi
```

Note that the environmental variables should be put immediately before the executable.

Alternatively, you may also place environmental variables in your shell environment (e.g. `.bashrc`). These will be automatically picked up when the application starts executing.

Note that mpirun_rsh is sensitive to the ordering of the command-line arguments.

There are many different parameters which could be used to improve the performance of applications depending upon their requirements from the MPI library. For a discussion on how to identify such parameters, see Section 8.

Job Launch using MPMD

The mpirun_rsh framework also supports job launching using MPMD mode. It permits the use of heterogeneous jobs using multiple executables and command line arguments. The following format needs to be used:

```
$ mpirun_rsh -config configfile -hostfile hosts
```

A list of different group of executables must be provided to the job launcher in the file `configfile`, one per line. The `configfile` can contain comments. Lines beginning with "#" are considered comments.

For example:

```
#Config file example

#Launch 4 copies of exe1 with arguments arg1 and arg2

-n 4 :  exe1 arg1 arg2

#Launch 2 copies of exe2

-n 2 :  exe2
```

A list of target nodes must be provided in the file `hosts` one per line and the allocation policy previously described is used.

Please note that this section only gives general information on how to run applications using mpirun_rsh. Please refer to the following sections for more information on how to run the application over various interfaces such as iWARP and RoCE.

Other Options

Other options of `mpirun_rsh` can be obtained using

```
$ mpirun_rsh --help
```

5.2.2 Run using `Hydra (mpiexec)`

Hydra is the default process manager for MPICH. MVAPICH2 also distributes Hydra along with with mpirun_rsh. Hydra can be used either by using `mpiexec` or `mpiexec.hydra`. All interfaces of

MVAPICH2 will work using Hydra. The following is an example of running a program using it:

```
$ mpiexec -f hosts -n 2 ./cpi
```

The Hydra process manager can be used to launch MPMD jobs. For example the following command:

```
$ mpiexec -genv FOO=1 -env BAR=1 -n 2 ./exec1 :  -env BAR=2 -n 2 ./exec2
```

The environment variable FOO=1 passed to "-genv" is applied the environment to all executables (i.e. exec1 and exec2). The values BAR=1 applies to exec1 and BAR=2 applies to only exec2.

This process manager has many features. Please refer to the following web page for more details.

http://wiki.mcs.anl.gov/mpich2/index.php/Using_the_Hydra_Process_Manager

5.2.3 Run using SLURM

SLURM is an open-source resource manager designed by Lawrence Livermore National Laboratory and maintained by SchedMD. SLURM software package and its related documents can be downloaded from: http://www.schedmd.com/

Once SLURM is installed and the daemons are started, applications compiled with MVAPICH2 can be launched by SLURM, e.g.

```
$ srun -n 2 ./a.out
```

The use of SLURM enables many good features such as explicit CPU and memory binding. For example, if you have two processes and want to bind the first process to CPU 0 and Memory 0, and the second process to CPU 4 and Memory 1, then it can be achieved by:

```
$ srun --cpu_bind=v,map_cpu:0,4 --mem_bind=v,map_mem:0,1 -n2 --mpi=none
./a.out
```

To use PMI-2 with SLURM, please use:

```
$ srun --mpi=pmi2 -n 2 ./a.out
```

If PMI-2 is selected and the installed version of SLURM supports PMI extensions, MVAPICH2 will automatically use the extensions.

For more information about SLURM and its features please visit http://www.schedmd.com/

5.2.4 Run on PBS/Torque Clusters

Both mpirun_rsh and mpiexec can take information from the PBS/Torque environment to launch jobs (i.e. launch on nodes found in PBS_NODEFILE).

You can also use MVAPICH2 in a tightly integrated manner with PBS. To do this you can install mvapich2 by adding the –with-pbs option to mvapich2. Below is a snippet from ./configure –help of the hydra process manager (mpiexec) that you will use with PBS/Torque.

–with-pbs=PATH specify path where pbs include directory and lib directory can be found –with-pbs-include=PATH specify path where pbs include directory can be found –with-pbs-lib=PATH specify path where pbs lib directory can be found

For more information on using hydra, please visit the following URL: http://wiki.mpich.org/mpich/index.php/Using_the_Hydra_Process_Manager

5.2.5 Run with Dynamic Process Management support

MVAPICH2 (OFA-IB-CH3 interface) provides MPI-2 dynamic process management. This feature allows MPI applications to spawn new MPI processes according to MPI-2 semantics. The following commands provide an example of how to run your application.

- To run your application using mpirun_rsh
  `$ mpirun_rsh -np 2 -hostfile hosts MV2_SUPPORT_DPM=1 ./spawn1`
  Note: It is necessary to provide the hostfile when running dynamic process management applications using mpirun_rsh.
- To run your application using mpiexec (Hydra)
  `$ mpiexec -n 2 -f hosts -env MV2_SUPPORT_DPM 1 ./spawn1`

Please refer to Section 11.72 for information about the MV2_SUPPORT_DPM environment variable.

5.2.6 Run with mpirun_rsh using OFA-iWARP Interface

The MVAPICH2 library can automatically detect iWARP cards and use them with the appropriate settings at run time. This feature deprecates the use of the environment variable `MV2_USE_IWARP_MODE` which was being used earlier to enable the use of iWARP devices at run time.

All the systems to be used need the following one time setup for enabling RDMA CM usage.

- **Setup the RDMA CM interface:** RDMA CM interface needs to be setup, configured with an IP address and connected to the network.
- **Setup the Local Address File:** Create the file (`/etc/mv2.conf`) with the local IP address to be used by RDMA CM. (Multiple IP addresses can be listed (one per line) for multi-rail configurations).
  `$ echo 10.1.1.1 >> /etc/mv2.conf`

Programs can be executed as follows:

`$ mpirun_rsh -np 2 n0 n1 prog`

The iWARP interface also provides TotalView debugging and shared library support. Please refer to Section 4.4.

5.2.7 Run with mpirun_rsh using OFA-RoCE Interface

RDMA over Converged Ethernet (RoCE) is supported with the use of the run time environment variable MV2_USE_RoCE.

Programs can be executed as follows:

```
$ mpirun_rsh -np 2 MV2_USE_RoCE=1 prog
```

RoCE requires loss-less Ethernet fabric. This requires to configure Ethernet switch to treat RoCE traffic as loss-less. A separate VLAN interface needs to be created on RoCE NIC on all compute nodes and assign a private IP address

In loss-less fabric setup, MVAPICH2 can be run in RoCE mode in following two ways

- Put private VLAN IP addresses in /etc/mv2.conf and run in RDMA CM mode
  ```
  $ mpirun_rsh -np 2 MV2_USE_RoCE=1 MV2_USE_RDMA_CM=1 prog
  ```

- All VLAN interfaces will appear as additional GID indexes (starting from 1) on the InfiniBand HCA side of the RoCE adapter. User can select non-default GID index using run-time parameter
  MV2_DEFAULT_GID_INDEX(11.83) and RoCE priority service level using MV2_DEFAULT_SERVICE_L
  ```
  $ mpirun_rsh -np 2 MV2_USE_RoCE=1 MV2_DEFAULT_GID_INDEX=1
  MV2_DEFAULT_SERVICE_LEVEL=<RoCE_Service_Level> prog
  ```

5.2.8 Run using IPoIB with mpirun_rsh or mpiexec

You would like to run an MPI job using IPoIB but your IB card is not the default interface for IP traffic. Assume that you have a cluster setup as the following:

#hostname	Eth Addr	IPoIB Addr
compute1	192.168.0.1	192.168.1.1
compute2	192.168.0.2	192.168.1.2
compute3	192.168.0.3	192.168.1.3
compute4	192.168.0.4	192.168.1.4

The Ethernet addresses are assigned to eth0 and the IPoIB addresses are assigned to the ib0 interface. The host names resolve to the 192.168.0.* addresses.

The most direct way to use the IPoIB network is to populate your hosts file with the IP addresses of the ib0 interfaces.

Example:

```
$ cat - > hosts
192.168.1.1
192.168.1.2
192.168.1.3
192.168.1.4

$ mpirun_rsh -hostfile hosts -n 4 ./app1
```

or

```
$ mpiexec -f hosts -n 4 ./app1
```

Another way to achieve this is to use the *-iface* option of hydra. This allows you to have your hosts file to use the host names even though they resolve to the eth0 interface.

Example:

```
$ cat - > hosts
compute1
compute2
compute3
compute4

$ mpiexec -f hosts -iface ib0 -n 4 ./app1
```

More information can be found at the following link.

5.2.9 Run using ADIO driver for Lustre

MVAPICH2 contains optimized Lustre ADIO support for the OFA-IB-CH3 interface. The Lustre directory should be mounted on all nodes on which MVAPICH2 processes will be running. Compile MVAPICH2 with ADIO support for Lustre as described in Section 4. If your Lustre mount is /mnt/datafs on nodes n0 and n1, on node n0, you can compile and run your program as follows:

```
$ mpicc -o perf romio/test/perf.c
$ mpirun_rsh -np 2 n0 n1 <path to perf>/perf -fname /mnt/datafs/testfile
```

If you have enabled support for multiple file systems, append the prefix "lustre:" to the name of the file. For example:

```
$ mpicc -o perf romio/test/perf.c
$ mpirun_rsh -np 2 n0 n1 ./perf -fname lustre:/mnt/datafs/testfile
```

5.2.10 Run using TotalView Debugger Support

MVAPICH2 provides TotalView support. The following commands provide an example of how to build and run your application with TotalView support. Note: running TotalView requires correct setup in your environment, if you encounter any problem with your setup, please check with your system administrator for help.

- 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...
  `$ 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`

5.2.11 Run using a profiling library

All MPI2-functions of MVAPICH2 support the MPI profiling interface. This allows MVAPICH2 to be used by a variety of profiling libraries for MPI applications.

Two use of profiling libraries will be describe below, Scalasca and mpiP;

- To use Scalasca, you should configure Scalasca by supplying the '`--mpi=mpich2`' option like shows below:
  `$./configure --mpi=mpich2`

  Once the installation is done, you will be able to use Scalasca with MVAPICH2.

  For more information about Scalasca and its features please visit Scalasca website.
- To use mpiP, you should build your program with the required libraries as described in the following command:
  `$ mpicc -g -o pgm.o pgm.c -L${mpiP_root}/lib -lmpiP -lm -lbfd -liberty -lunwind`
  Simply run your MPI application as usual. On running a mpi application, a file with mpiP extension gets created which contains the following information
  - Header
  - MPI Time (seconds)
  - Callsites
  - Aggregate Time (top twenty, descending, milliseconds)
  - Aggregate Sent Message Size (top twenty, descending, bytes)
  - Callsite Time statistics (all, milliseconds)

– Callsite Message Sent statistics (all, sent bytes)

Prerequisites to install mpiP library: mpiP library has the build dependency on following libraries. Usually, these libraries are installed along with the Linux installation. You may also download these libraries from the specified URLs and install them according to their README file.

– GNU binutils.
– iberty
– unwind

Alternative to GNU binutils, libelf and libdwarf can be used for source lookup.

The sample configure command to build mpiP library is given below. Please refer to the mpiP web page for commands specific to your system environment.

```
./configure LDFLAGS=-L/usr/lib64 LIBS=‘‘-lbfd -liberty’’
--enable-collective-report-default --enable-demangling=GNU
--with-cc=mpicc --with-cxx=mpiCC --with-f77=mpif77
```

5.3 Compile and Run Applications with Singularity

To compile the file example.c, use the mpicc built in the Singularity environment as described in Section 4.15.

```
$ singularity exec Singularity-Centos-7.img mpicc /path/to/example.c -o
example
```

The MPI binary (example) can be installed into the container at /usr/bin as follows.

```
$ sudo singularity copy Singularity-Centos-7.img ./example /usr/bin/
```

Run the MPI program within the container by using the mpirun_rsh job launcher on the host. As with the native MVAPICH2 version, different features can be enabled through the use of runtime variables.

```
$ mpirun_rsh -np #procs -hostfile hostfile MV2_IBA_HCA=mlx4_0 singularity
exec Singularity-Centos-7.img /usr/bin/example
```

6 Advanced Usage Instructions

In this section, we present the usage instructions for advanced features provided by MVAPICH2.

6.1 Running on Customized Environments

In MVAPICH2 2.3, run-time variables are used to switch various optimization schemes on and off. Following is a list of optimizations schemes and the control environmental variables, for a full list please refer to the section 11:

- **Extended Reliable Connection:** Use the XRC InfiniBand transport available with Mellanox ConnectX adapters. Default: off; to enable:
  `$ mpirun_rsh -np 2 n0 n1 MV2_USE_XRC=1 prog`
  or
  `$ mpiexec -n 2 -hosts n0,n1 -env MV2_USE_XRC 1 prog`

- **Enable use of multiple communication channels:** Indicates the number of queue pairs per port to be used for communication on an end node. Default: 1; to change default value:
  `$ mpirun_rsh -np 2 n0 n1 MV2_NUM_QP_PER_PORT=2 prog`
  or
  `$ mpiexec -n 2 -hosts n0,n1 -env MV2_NUM_QP_PER_PORT 2 prog`

- **Adaptive RDMA fast path:** using RDMA write to enhance performance for short messages. Default: on; to disable:
  `$ mpirun_rsh -np 2 n0 n1 MV2_USE_RDMA_FAST_PATH=0 prog`
  or
  `$ mpiexec -n 2 -hosts n0,n1 -env MV2_USE_RDMA_FAST_PATH 0 prog`

6.2 Export Environment

Traditionally with mpirun_rsh you have to specify all environment variables that you want visible to the remote MPI processes on the command line. With the –export option of mpirun_rsh this is no longer necessary.

6.2.1 Sample Use

Traditional Method

```
$ mpirun_rsh -n 2 compute1 compute2 FOO=1 BAR=baz ./osu_latency
```

With export option

```
$ export FOO=1 BAR=baz
$ mpirun_rsh -export -n 2 compute1 compute2 ./osu_latency
```

Please note that the -export option does not overwrite variables that are normally already set when you first ssh into the remote node. If you want to export all variables including the ones that are already set you can use the -export-all option.

With export-all option

```
$ export PATH=$PATH:/some/special/path
$ mpirun_rsh -export-all -n 2 compute1 compute2 ./osu_latency
```

6.3 Configuration File Processing

MVAPICH2 supports the use of configuration values to ease the burden of users when they would like to set and repeatedly use many different environment variables. These can be stored in a configuration file with statements of the form "VARIABLE = VALUE". Full line comments are supported and begin with the "#" character.

The system configuration file can be placed at /etc/mvapich2.conf while user configuration files are located at "`~/.mvapich2.conf`" by default. The user configuration file can be specified at runtime by MV2_USER_CONFIG if the user would like to have mvapich2 read from a different location.

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

6.3.1 Sample Use

Run with blocking mode enabled

```
$ cat ~/.mvapich2.conf
# Enable blocking mode
MV2_USE_BLOCKING = 1
$ mpirun_rsh -n 2 compute1 compute2 ./osu_latency
```

Do not use user configuration file

```
$ mpirun_rsh -n 2 compute1 compute2 MV2_IGNORE_USER_CONFIG=1 ./osu_latency
```

6.4 Suspend/Resume Support

MVAPICH2 can be suspended and resumed when using a process launcher that catches and forwards the appropriate signals.

For example, when using mpirun_rsh you can type Ctrl-Z (or send the SIGTSTP signal) at the terminal and the job will suspend. You can then later send the SIGCONT signal to the job and it will continue.

6.5 Running with Efficient CPU (Core) Mapping

MVAPICH2-CH3 interfaces support architecture specific CPU mapping through the Portable Hardware Locality (hwloc) software package. By default, the HWLOC sources are compiled and built while the MVAPICH2 library is being installed. Users can choose the "–disable-hwloc" parameter while configuring the library if they do not wish to have the HWLOC library installed. However, in such cases, the MVAPICH2 library will not be able to perform any affinity related operations.

There are two major schemes as indicated below. To take advantage of any of these schemes, the jobs need to run with CPU affinity (MV2_ENABLE_AFFINITY) and shared memory (MV2_USE_SHARED_MEM) turned on (default). If users choose to set these run-time parameters to 0, then the kernel takes care of mapping processes to cores and none of these schemes will be enabled.

To report the process mapping, users can set the environment variable MV2_SHOW_CPU_BINDING to 1 (Section 10.7).

6.5.1 Using HWLOC for CPU Mapping

Under this scheme, the HWLOC tool will be used at job-launch time to detect the processor's micro-architecture, and then generate a suitable cpu mapping string based. Three policies are currently implemented: "bunch", "scatter", and "hybrid". By default, we choose to use the "hybrid" mapping. However, we also allow users to choose a binding policy through the run-time variable, MV2_CPU_BINDING_POLICY. (Section 11.14)

For example, if you want to run 4 processes per node and utilize "bunch" policy on each node, you can specify:

```
$ mpirun_rsh -np 4 -hostfile hosts MV2_CPU_BINDING_POLICY=bunch ./a.out
```

The CPU binding will be set as shown in Figure 2.

If you want to run 4 processes per node and utilize "scatter" policy on each node, you can specify:

```
$ mpirun_rsh -np 4 -hostfile hosts MV2_CPU_BINDING_POLICY=scatter ./a.out
```

The CPU binding will be set as shown in Figure 3.

If two applications with four processes each need to share a given node (with eight cores) at the

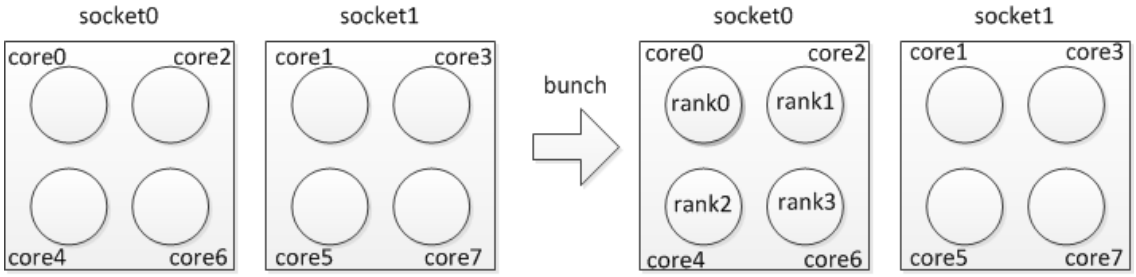

Figure 2: Process placement with "bunch" CPU binding policy

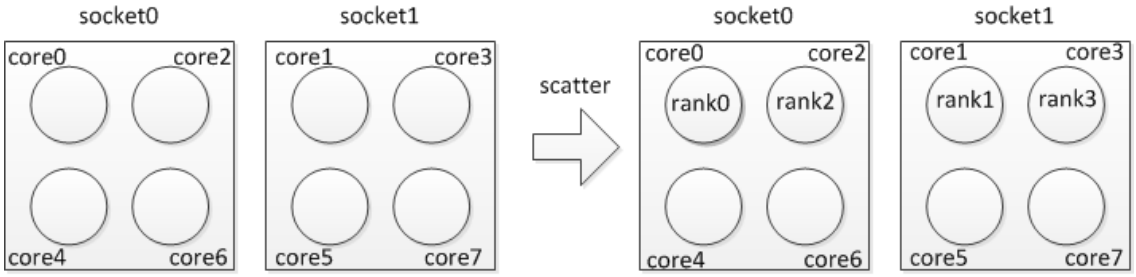

Figure 3: Process placement with "Scatter" CPU binding policy

same time with "bunch" policy, you can specify:

```
$ mpirun_rsh -np 4 -hostfile hosts MV2_CPU_BINDING_POLICY=bunch ./a.out
$ mpirun_rsh -np 4 -hostfile hosts MV2_CPU_BINDING_POLICY=bunch ./b.out
```

The CPU binding will be set as shown in Figure 4.

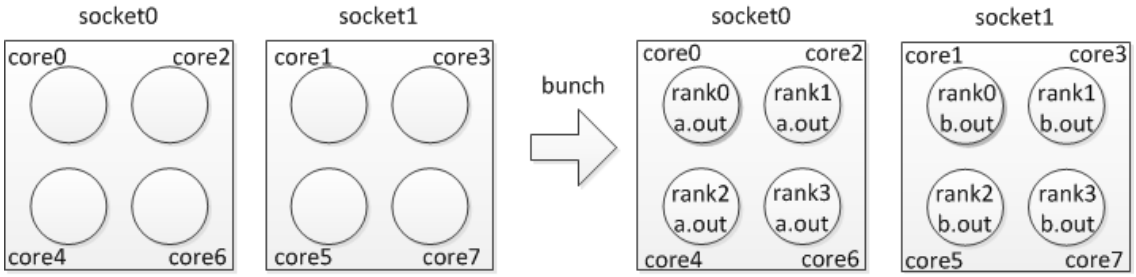

Figure 4: Process placement with two applications using the "bunch" CPU binding policy

If two applications with four processes each need to share a given node (with eight cores) at the same time with "scatter" policy, you can specify:

```
$ mpirun_rsh -np 4 -hostfile hosts MV2_CPU_BINDING_POLICY=scatter ./a.out
$ mpirun_rsh -np 4 -hostfile hosts MV2_CPU_BINDING_POLICY=scatter ./b.out
```

The CPU binding will be set as shown in Figure 5.

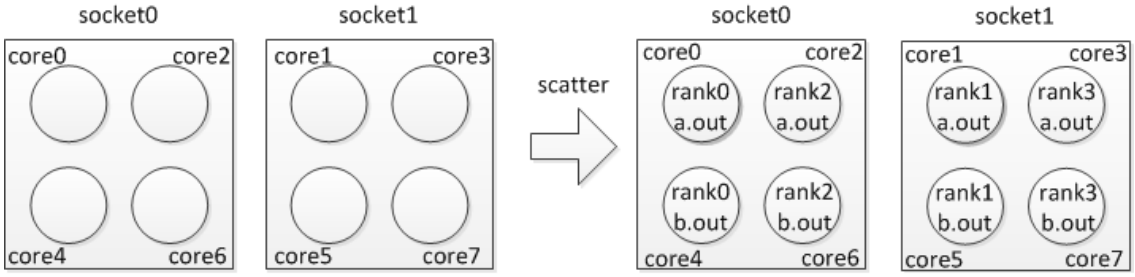

Figure 5: Process placement with two applications using the "scatter" CPU binding policy

The aforementioned binding is based on the core level, meaning each MPI process will be bound to a specific core. Actually, we provide different process binding level. There are three binding levels: "core", "socket", and "numanode" (which is designed for some multicore processor with NUMA node unit). We use the "core" as the default binding level, and we also allow users to choose a binding level through the run-time variable, MV2_CPU_BINDING_LEVEL. (Section 11.16) For example, if you want to run 4 processes per node and utilize "socket" as the binding level on each node, you can specify:

```
$ mpirun_rsh -np 4 -hostfile hosts MV2_CPU_BINDING_LEVEL=socket ./a.out
```

The CPU binding will be set as shown in Figure 6. Note: because we use "bunch" as the default binding policy, all four processes will be bound to the first socket and each of them can use all four cores in this socket. When the binding policy is "bunch" and the binding level is "socket", processes will be bound to the same socket until the process number is larger than the core number in the socket.

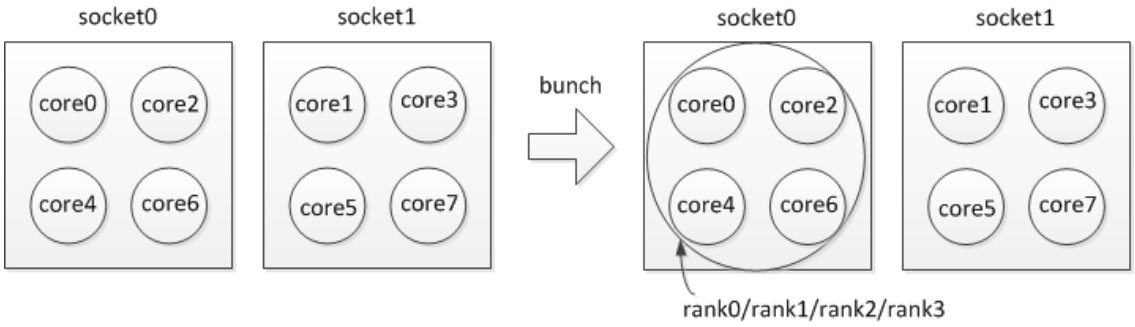

Figure 6: Process placement with the "bunch" CPU binding policy and "socket" binding level

If you want to run 4 processes per node, utilize "socket" as the binding level and "scatter" as the binding policy, you can specify:

```
$ mpirun_rsh -np 4 -hostfile hosts MV2_CPU_BINDING_LEVEL=socket
MV2_CPU_BINDING_POLICY=scatter ./a.out
```

The CPU binding will be set as shown in Figure 7.

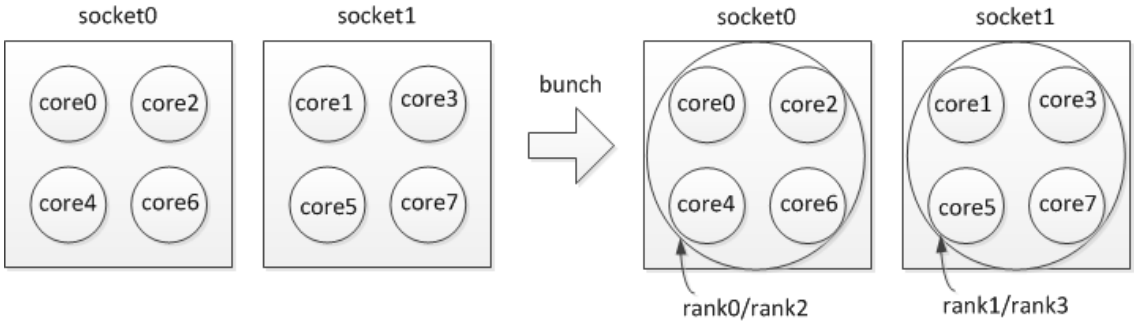

Figure 7: Process placement with the "scatter" CPU binding policy and "socket" binding level

6.5.2 Mapping Policies for Hybrid (MPI+Threads) Applications

We have introduced one additional policy called "hybrid" for MV2_CPU_BINDING_POLICY variable. This policy can also be used for hybrid MPI+Threads (OpenMP, pthreads, etc.) applications where each MPI rank additionally spawns multiple threads. The detailed usage of this variable and any additional variables you might need is discussed in Section 6.19.

In addition to MV2_CPU_BINDING_POLICY, we have also provided a new environment variable called MV2_HYBRID_BINDING_POLICY to specify thread specific binding policies. The detailed description and usage of this variable is explained in Section 6.20.

6.5.3 User defined CPU Mapping

Under the second scheme, users can also use their own mapping to bind processes to CPU's on modern multi-core systems. The feature is especially useful on multi-core systems, where performance may be different if processes are mapped to different cores. The mapping can be specified by setting the environment variable `MV2_CPU_MAPPING` (Section 11.13).

For example, if you want to run 4 processes per node and utilize cores 0, 1, 4, 5 on each node, you can specify:

```
$ mpirun_rsh -np 64 -hostfile hosts MV2_CPU_MAPPING=0:1:4:5 ./a.out
```

or

```
$ mpiexec -n 64 -f hosts -env MV2_CPU_MAPPING 0:1:4:5 ./a.out
```

In this way, process 0 on each node will be mapped to core 0, process 1 will be mapped to core 1, process 2 will be mapped to core 4, and process 3 will be mapped to core 5. For each process, the mapping is separated by a single ":".

MVAPICH2 supports binding one process to multiple cores in the same node with "," or "-". For example:

```
$ mpirun_rsh -np 64 -hostfile hosts MV2_CPU_MAPPING=0,2,3,4:1:5:6 ./a.out
```

or

```
$ mpirun_rsh -np 64 -hostfile hosts MV2_CPU_MAPPING=0,2-4:1:5:6 ./a.out
```

In this way, process 0 on each node will be mapped to core 0, core 2, core 3, and core 4; process 1 will be mapped to core 1, process 2 will be mapped to core 5, and process 3 will be mapped to core 6. This feature is designed to support the case that one rank process will spawn multiple threads and set thread binding in the program.

6.5.4 Performance Impact of CPU Mapping

Here we provide a table with latency performance of 0 byte and 8KB messages using different CPU mapping schemes. The results show how process binding can affect the benchmark performance. We strongly suggest the consideration of best CPU mapping on multi-core platforms when carrying out benchmarking and performance evaluation with MVAPICH2.

The following measurements were taken on the machine with the dual quad-core 2.53GHz Intel Xeon processors with 12MB L3 shared cache (among cores in one socket). MVAPICH2-2.3 was built with gcc-4.4.6 and default configure arguments:

	Message Latency		
Core Pair	0-byte	8k-byte	Notes
1,2	0.17 us	1.83 us	same socket, shared L3 cache, best performance
0,1	0.17 us	1.87 us	same socket, shared L3 cache, but core 0 handles interrupts
1,5	0.41 us	3.16 us	different sockets
0,4	0.42 us	3.17 us	different sockets, but core 0 handles interrupts

6.6 Running with LiMIC2

MVAPICH2 CH3-based interfaces support LiMIC2 for intra-node communication for medium and large messages to get higher performance. LiMIC2 is also used to optimize intra-node one-sided communication in OFA-IB-CH3 and OFA-iWARP-CH3 interfaces. It is disabled by default because it depends on the LiMIC2 package to be previously installed. As a convenience we have distributed the latest LiMIC2 package (as of this release) with our sources.

To install this package, please take the following steps.

- Navigate to the LiMIC2 to source
  ```
  $ cd limic2-0.5.6
  ```
- Configure and build the source
  ```
  limic2-0.5.6$ ./configure --enable-module --prefix=/usr --sysconfdir=/etc
  && make
  ```
- Install
  ```
  limic2-0.5.6$ sudo make install
  ```

Before using LiMIC2 you'll need to load the kernel module. If you followed the instructions above you can do this using the following command (LSB init script).

- `$ /etc/init.d/limic start`

Please note that supplying '`--sysconfdir=/etc`' in the configure line above told the package to install the init script and an udev rule in the standard location for system packages. Supplying '`--prefix=/usr`' will also install the headers and libraries in the system path. These are optional but recommended.

Now you can use LiMIC2 with MVAPICH2 by simply supplying the '`--with-limic2`' option when configuring MVAPICH2. You can run your applications as normal and LiMIC2 will be used by default. To disable it at run time, use the env variable:

```
$ mpirun_rsh -np 64 -hostfile hosts MV2_SMP_USE_LIMIC2=0 ./a.out
```

6.7 Running with Shared Memory Collectives

In MVAPICH2, support for shared memory based collectives has been enabled for MPI applications running over OFA-IB-CH3, OFA-iWARP-CH3, TrueScale (PSM-CH3) and Omni-Path (PSM2-CH3) interfaces. Currently, this support is available for the following collective operations:

- MPI_Allreduce
- MPI_Reduce
- MPI_Barrier
- MPI_Bcast

Optionally, these feature can be turned off at run time by using the following parameters:

- `MV2_USE_SHMEM_COLL` (section 11.94)
- `MV2_USE_SHMEM_ALLREDUCE` (section 11.91)
- `MV2_USE_SHMEM_REDUCE` (section 11.95)
- `MV2_USE_SHMEM_BARRIER` (section 11.92)
- `MV2_USE_SHMEM_BCAST` (section 11.93)

Please refer to Section 11 for further details.

6.8 Running Collectives with Hardware based Multicast support

In MVAPICH2, support for multicast based collectives has been enabled for MPI applications running over OFA-IB-CH3 interface. Currently, this support is available for the following collective operations:

- MPI_Bcast
- MPI_Allreduce
- MPI_Scatter

This feature is disabled by default. This can be turned on at runtime by using parameter MV2_USE_MCAST (11.120). This feature is effective when the MPI job is running on more than the threshold
MV2_MCAST_NUM_NODES_THRESHOLD (11.121) number of nodes.

This feature requires the cluster to be installed with libibumad and libibmad libraries. If there are not installed, this feature can be disabled with –disable-mcast configure flag.

This feature requires to have read/write permission for users on /dev/infiniband/umad0

```
$ ls -l /dev/infiniband/umad0
crw-rw-rw- 1 root root 231, 0 Jul  8 19:47 /dev/infiniband/umad0
```

6.9 Running MPI_Gather collective with intra-node Zero-Copy designs (using LiMIC2)

In MVAPICH2, we offer new intra-node Zero-Copy designs (using LiMIC2) for the MPI_Gather collective operation based on the LiMIC2 feature. This feature can be used, when the library has been configured to use LiMIC2(6.6). This feature is disabled by default and can be turned on at runtime by using the parameter MV2_USE_LIMIC_GATHER (11.119).

6.10 Running with scalable UD transport

MVAPICH2 has scalable design with InfiniBand connection less transport Unreliable Datagram (UD). Applications can use UD only transport by simply configuring MVAPICH2 with the –enable-hybrid and setting the environment variable MV2_USE_ONLY_UD(11.117). In this mode, library not uses any reliable RC connections. This feature eliminates all the overheads associated with RC connections and reduces the memory footprint at large scale.

6.11 Running with Integrated Hybrid UD-RC/XRC design

MVAPICH2 has integrated hybrid transport support for OFA-IB-CH3. This provides the capability to use Unreliable Datagram (UD), Reliable Connection (RC) and eXtended Reliable Connection (XRC) transports of InfiniBand. This hybrid transport design is targeted at emerging clusters with multi-thousand core clusters to deliver best possible performance and scalability with constant memory footprint.

Applications can use Hybrid transport by simply configuring MVAPICH2 with the –enable-hybrid option. In this configuration, MVAPICH2 seamlessly uses UD and RC/XRC connections by

default. The use of UD transport can be disabled at run time by setting the environment variable MV2_USE_UD_HYBRID(11.116) to Zero.

MV2_HYBRID_ENABLE_THRESHOLD (11.111) defines the threshold for enabling the hybrid transport. Hybrid mode will be used when the size of the job is greater than or equal to the threshold. Otherwise, it uses default RC/XRC connections.

For a full list of Hybrid environment variables, please refer Section 11.

6.12 Running with Multiple-Rail Configurations

MVAPICH2 has integrated multi-rail support for OFA-IB-CH3 and OFA-iWARP-CH3 interfaces. Run-time variables are used to specify the control parameters of the multi-rail support; number of adapters with MV2_NUM_HCAS (section 11.33), number of ports per adapter with MV2_NUM_PORTS (section 11.35), and number of queue pairs per port with MV2_NUM_QP_PER_PORT (section 11.37). Those variables are default to 1 if you do not specify them.

Large messages are striped across all HCAs. The threshold for striping is set according to the following formula:
(MV2_VBUF_TOTAL_SIZE × MV2_NUM_PORTS × MV2_NUM_QP_PER_PORT × MV2_NUM_HCAS).
In addition, there is another parameter MV2_STRIPING_THRESHOLD (section 11.71) which users can utilize to set the striping threshold directly.

MVAPICH2 also gives the flexibility to balance short message traffic over multiple HCAs in a multi-rail configuration. The run-time variable MV2_SM_SCHEDULING can be used to choose between the various load balancing options available. It can be set to USE_FIRST (Default) or ROUND_ROBIN. In the USE_FIRST scheme, the HCA in slot 0 is always used to transmit the short messages. If ROUND_ROBIN is chosen, messages are sent across all HCAs alternately.

In the following example, we can use multi-rail support with two adapters, using one port per adapter and one queue pair per port:

```
$ mpirun_rsh -np 2 n0 n1 MV2_NUM_HCAS=2 MV2_NUM_PORTS=1
MV2_NUM_QP_PER_PORT=1 prog
```

Using the Hydra process manager, the same can be accomplished by:

```
$ mpiexec -n 2 -hosts n0,n1 -env MV2_NUM_HCAS 2 -env MV2_NUM_PORTS 1 -env
MV2_NUM_QP_PER_PORT 1 prog
```

Note that the default values of `MV2_NUM_PORTS` and `MV2_NUM_QP_PER_PORT` are 1, so they can be omitted.

```
$ mpirun_rsh -np 2 n0 n1 MV2_NUM_HCAS=2 prog
```

Using the Hydra process launcher, the following command can be used:

```
$ mpiexec -n 2 -hosts n0,n1 -env MV2_NUM_HCAS 2 prog
```

The user can also select the particular network card(s) that should be used by using the MV2_IBA_HCA environment variable specified in section 11.25. The following is an example of how to run MVAPICH2 in this mode. (In the example "mlx4_0" is the name of the InfiniBand card as displayed by the output of the "ibstat" command).

```
$ mpirun_rsh -np 2 n0 n1 MV2_IBA_HCA=mlx4_0 prog
```

If there are multiple HCAs in the system, the user can also selectively use some or all of these HCAs for network operations by using the MV2_IBA_HCA environment variable. Different HCAs are delimited by colons ":". An example is shown below. In the example "mlx4_0" and "mlx4_1" are the names of the InfiniBand card as displayed by the output of the "ibstat" command. There can be other HCAs in the system as well.

```
$ mpirun_rsh -np 2 n0 n1 MV2_IBA_HCA=mlx4_0:mlx4_1 prog
```

6.13 Enhanced design for Multiple-Rail Configurations

MVAPICH2 now features an enhanced design for multiple rail configurations for OFA-IB-CH3 and OFA-iWARP-CH3 interfaces. It can broadly be explained by the figure given below. In addition to the earlier design where the rails were shared among processes at run time (as depicted under the Rail Sharing banner in the figure below), MVAPICH2 now features a new RAIL BINDING policy which will dedicate a particular rail to a particular process.

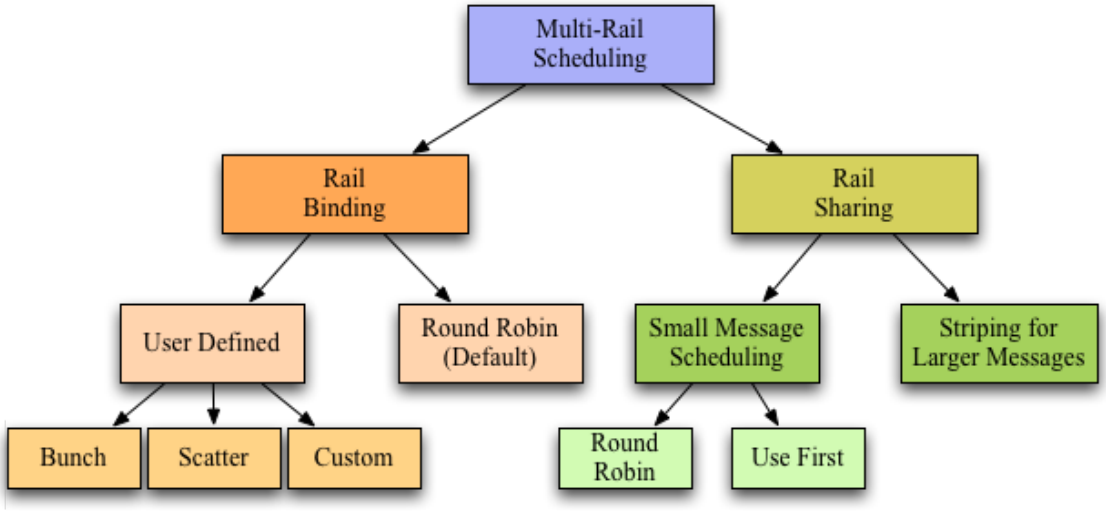

Figure 8: Multi-rail Scheduling policies

The scheduling policies are broadly classified into 2 basic types. Rail Binding and Rail Sharing.

- The Rail Sharing policy has been kept the same as in earlier versions and can be set using the MV2_RAIL_SHARING_POLICY (section 11.38) (previously MV2_SM_SCHEDULING) parameter. It can take the values USE_FIRST or ROUND_ROBIN. However as opposed to the earlier versions, the number of HCAs of the same type on a system get detected automatically and used. The user can still make use of the MV2_NUM_HCAS parameter and set the number to a custom number of rails if required. If this parameter is not set the maximum number gets used.

- In the Rail Binding policy only that HCAs is made available to the process which has been assigned to it using some scheme. The default policy used is 'Rail Binding in a round-robin manner': If the parameter MV2_RAIL_SHARING_POLICY is not specified, MVAPICH2 assigns HCAs to the processes in a round-robin manner. The Rail Binding policy can either be default as described above or user defined. For the user defined mode the following parameter MV2_RAIL_SHARING_POLICY=FIXED_MAPPING should be set.

  The user defined policies can be set in the following manner by giving appropriate values to the parameter MV2_PROCESS_TO_RAIL_MAPPING (section 11.40)

  - MV2_PROCESS_TO_RAIL_MAPPING=BUNCH : The HCAs are assigned in a block manner. e.g. For 4 rails and 16 processes the mapping will be 0:0:0:0:1:1:1:1:2:2:2:2:3:3:3:3
  - MV2_PROCESS_TO_RAIL_MAPPING=SCATTER : The HCAs are assigned in a cyclic manner. e.g. For 4 rails and 16 processes the mapping will be 0:1:2:3:0:1:2:3:0:1:2:3:0:1:2:3
  - The third option is a custom list which can be passed as a string to MV2_PROCESS_TO_RAIL_MAPPING separated by a colon(:) as shown above.

- In addition to these parameters the other ENVS that can be set are MV2_VBUF_TOTAL_SIZE, MV2_NUM_QP_PER_PORT, MV2_RAIL_SHARING_LARGE_MSG_THRESHOLD (previously MV2_STRIPING_THRESHOLD), MV2_IBA_HCA. Please refer to the usage described for these parameters as well as the ones described above in (section 11).

6.14 Running with Fault-Tolerance Support

6.14.1 System-Level Checkpoint/Restart

MVAPICH2 provides system-level rollback-recovery capability based on a coordinated Checkpoint-Restart protocol involving all the application processes.

The following section (6.14.1) provides instructions for basic checkpoint/restart operations with MVAPICH2. These require the Berkeley Lab Checkpoint/Restart (BLCR) library in order perform checkpoints of local processes. Then, advanced features are presented. Section 6.14.1 details the usage of the fast-checkpointing scheme based on aggregation. Section 6.14.1 present the support of the new standardized Fault Tolerance Backplane (FTB).

Basic Checkpoint/Restart Scheme: BLCR is a library that allows to take checkpoint of individual processes. Its usage is mandatory to take advantage of the checkpoint/restart functionality in MVAPICH2. Here are the steps that allows the usage of BLCR with MVAPICH2.

- Download and install the BLCR (Berkeley Lab's Checkpoint/Restart) package. The packages can be downloaded from this web page.
- Make sure the BLCR packages are installed on every node and the `LD_LIBRARY_PATH` must contain the path to the shared library of BLCR, usually `$BLCR_HOME/lib`.
- MVAPICH2 needs to be compiled with checkpoint/restart support, see section 4.4.
- BLCR kernel modules must be loaded on all the compute nodes.
- Make sure the PATH contains the path to the executables of BLCR, usually `$BLCR_HOME/bin`.

Users are strongly encouraged to read the Administrators guide of BLCR, and test the BLCR on the target platform, before using the checkpointing feature of MVAPICH2.

Checkpointing operation

Now, your system is set up to use the Checkpoint/Restart features of MVAPICH2. Several parameters are provided by MVAPICH2 for flexibility in configuration and using the Checkpoint / Restart features. If mpiexec is used as the job start up mechanism, these parameters need to be set in the user's environment through the BASH shell's `export` command, or the equivalent command for other shells. If mpirun_rsh is used as the job start up mechanism, these parameters need to be passed to mpirun_rsh through the command line.

- MV2_CKPT_FILE: This parameter specifies the path and the base file name for checkpoint files of MPI processes. Please note that File System performance is critical to the performance of checkpointing. This parameter controls which file system will be used to store the checkpoint files. For example, if your PVFS2 is mounted at `/mnt/pvfs2`, using `MV2_CKPT_FILE=/mnt/pvfs2/ckptfile` will let the checkpoint files being stored in pvfs2 file system. See Section 11.4 for details.
- MV2_CKPT_INTERVAL: This parameter (in minutes) can be used to enable automatic checkpointing. See Section 11.5 for details.
- MV2_CKPT_MAX_SAVE_CKPTS: This parameter is used to limit the number of checkpoints saved on file system. See Section 11.6 for details.
- MV2_CKPT_NO_SYNC: This parameter is used to control whether the program forces the checkpoint files being synced to disk or not before it continues execution. See Section 11.7 for details.

In order to provide maximum flexibility to end users who wish to use the checkpoint/restart features of MVAPICH2, we have provided three different methods that can be used to take checkpoints during the execution of the MPI application. These methods are described as follows:

- Manual Checkpointing: In this mode, the user simply launches an MPI application and chooses when to checkpoint the application. This mode can be primarily used for experimentation

during deployment stages. In order to use this mode, the MPI application is launched normally using `mpiexec` or `mpirun_rsh`. When the user decides to take a checkpoint, the users can issue a BLCR command namely "`cr_checkpoint`" in the following manner:

```
cr_checkpoint -p <PID>
```

where PID is the process id of the `mpiexec` or `mpirun_rsh` process. In order to simplify the process, the script `mv2_checkpoint` can be used. This script is available in the same directory as `mpiexec` and `mpirun_rsh`.

- Automated Checkpointing: In this mode, the user can launch the MPI application normally using `mpiexec` or `mpirun_rsh`. However, instead of manually issuing checkpoints as described in the above bullet, a parameter (`MV2_CKPT_INTERVAL`) can be set to automatically take checkpoints and user-defined intervals. Please refer to Section 11.5 for a complete usage description of this variable. This mode can be used to take checkpoints of a long running application, for example every 1 hour, 2 hours etc. based on user's choice.

- Application Initiated Synchronous Checkpointing: In this mode, the MPI application which is running can itself request for a checkpoint. Application can request a whole program checkpoint synchronously by calling `MVAPICH2_Sync_Checkpoint`. Note that it is a collective operation, and this function must be called from all processes to take the checkpoint. This mode is expected to be used by applications that can be modified and have well defined synchronization points. These points can be effectively used to take checkpoints. An example of how this mode can be activated is given below.

```
#include "mpi.h"
#include <unistd.h>
#include <stdio.h>

int main(int argc,char *argv[])
{
    MPI_Init(&argc,&argv);
    printf("Computation\n");
    sleep(5);
    MPI_Barrier(MPI_COMM_WORLD);
    MVAPICH2_Sync_Checkpoint();
    MPI_Barrier(MPI_COMM_WORLD);
    printf("Computation\n");
    sleep(5);
    MPI_Finalize();
    return 0;
}
```

Restart operation

To restart a job from a **manual checkpoint**, users need to issue another command of BLCR, "`cr_restart`" with the checkpoint file name of the MPI job console as the parameter. Usually, this file is named
`context.<pid>`. The checkpoint file name of the MPI job console can be specified when issuing the checkpoint (see the "`cr_checkpoint --help`" for more information). Please note that the names of checkpoint files of the MPI processes will be assigned according to the environment variable MV2_CKPT_FILE,
(`$MV2_CKPT_FILE.<number of checkpoint>.<process rank>`).

To restart a job from an **automatic checkpoint**, use `cr_restart $MV2_CKPT_FILE.<number of checkpoint>.auto`.

If the user wishes to restart the MPI job on a different set of nodes, the host file that was specified along with the "`-hostfile`" option during job launch phase should be modified accordingly before trying to restart a job with "`cr_restart`". This modified "`hostfile`" must be at the same location and with the same file name as the original hostfile. The mpirun_rsh framework parses the host file again when trying to restart from a checkpoint, and launches the job on the corresponding nodes. This is possible as long as the nodes in which the user is trying to restart has the exact same environment as the one in which the checkpoint was taken (including shared NFS mounts, kernel versions, and user libraries).

For this to function correctly, the user should disable pre-linking on both the source and the destination node. See the FAQ Section of the BLCR userguide for more information.

Please refer to the Section 9.6 for troubleshooting with Checkpoint/Restart.

Write-Aggregation based Fast Checkpointing Scheme: MVAPICH2 provides an enhanced technique that allows fast checkpoint and restart. This scheme, named Aggregation, relies on the Filesystem in Userspace (FUSE) library.

Although Aggregation is optional, its usage is recommended to achieve best performances during checkpoint and restart operations. That is why, if the FUSE library is detected during configuration step, it will be automatically enabled (see section 4.4). Once enabled at configuration step, aggregation scheme can be disabled at run time by setting the environment variable `MV2_CKPT_USE_AGGREGATION=0` (see section 11.8 for details).

The following steps need to be done to use FUSE library for aggregation scheme.

- Download and install the Filesystem in Userspace (FUSE) library package. The packages can be downloaded from this web page. To get the best performance, users are encouraged to upgrade to kernel version $\geq$ 2.6.26 and use FUSE library $\geq$ 2.8.
- Make sure the FUSE packages are installed on every node and the `LD_LIBRARY_PATH` must contain the path to the shared library of FUSE, usually `$FUSE_HOME/lib`.
- MVAPICH2 needs to be compiled with checkpoint/restart support, see section 4.4.
- FUSE kernel modules must be loaded on all the compute nodes.

- Make sure the PATH contains the path to the executables of FUSE, usually `$FUSE_HOME/bin`.

If write aggregation has been enabled at configuration time, MVAPICH2 will check the FUSE configuration of each node during startup (FUSE module loaded and `fusermount` command in the PATH). If one node is not properly configured, then MVAPICH2 will abort. In this case, you need to fix the FUSE configuration of the nodes, or disable aggregation using `MV2_CKPT_USE_AGGREGATION=0` to run MVAPICH2.

Fault Tolerance Backplane (FTB) support: MVAPICH2 supports the new standardized Fault Tolerance Backplane (FTB). FTB can be used for Checkpoint-Restart and Job Pause-Migration-Restart Frameworks. Activating FTB support is optional to perform checkpoint/restart operations, but it will allow MVAPICH2 to automatically publish and listen to standardized events through the Fault Tolerance Backplane.

FTB has been developed and standardized by the CIFTS project. It enables faults to be handled in a coordinated and holistic manner in the entire system, providing for an infrastructure which can be used by different software systems to exchange fault-related information.

If using the FTB framework for checkpoint/restart, the following steps need to be done in addition to the above nodes.

- Download and install the FTB (Fault Tolerance Backplane) package. The packages can be downloaded from here.
- Make sure the FTB packages are installed on every node and the `LD_LIBRARY_PATH` must contain the path to the shared library of FTB, usually `$FTB_HOME/lib`.
- MVAPICH2 needs to be compiled with checkpoint/restart as well as FTB support, see section 4.4.
- Start FTB Database server (`$FTB_HOME/sbin/ftb_database_server`) on one of the nodes, this node will act as server node for all the FTB agents.
- Start FTB agents (`$FTB_HOME/sbin/ftb_agent`) on all the compute nodes.

Checkpoint/Restart support for the OFA-IB-Nemesis Interface: MVAPICH2 also provides Checkpoint-Restart capabilities for the OFA-IB-Nemesis interface. Currently, this feature can only be used in conjunction with the Hydra process manager. More details on using Hydra to checkpoint an MPI application can be found here.

6.14.2 Multi-Level Checkpointing with Scalable Checkpoint-Restart (SCR)

The Scalable Checkpoint-Restart (SCR) library developed at the Lawrence Livermore National Laboratory (LLNL) enables MPI applications to utilize distributed storage on Linux clusters to attain high file I/O bandwidth for checkpointing and restarting large-scale jobs. With SCR, jobs

run more efficiently, lose less work upon a failure, and reduce load on critical shared resources such as the parallel file system and the network infrastructure.

In the current SCR implementation, application checkpoint files are cached in storage local to the compute nodes, and a redundancy scheme is applied such that the cached files can be recovered even after a failure disables part of the system. SCR supports the use of spare nodes such that it is possible to restart a job directly from its cached checkpoint, provided the redundancy scheme holds and provided there are sufficient spares.

The SCR library implements three redundancy schemes which trade of performance, storage space, and reliability:

- **Local**: Each checkpoint file is written to storage on the local node
- **Partner**: Each checkpoint file is written to storage on the local node, and a full copy of each file is written to storage on a partner node
- **XOR**: Each checkpoint file is written to storage on the local node, XOR parity data are computed using checkpoint files from a set of nodes, and the parity data are stored among the set

SCR is integrated into MVAPICH2 to provide Multi-Level checkpointing capabilities to MPI applications in two modes: application-aware mode and transparent mode. The following subsections illustrate the steps needed to checkpoint an application using these two schemes.

Application-Aware Multi-Level Checkpointing: In this case, it is assumed that the application knows what data to checkpoint and how to read data from a checkpoint during a restart. It is also assumed that each process writes its checkpoint data in a unique file. For an application to actually write a checkpoint using SCR, the following steps need to be followed.

- The application must call `SCR_Start_checkpoint` to define the start boundary of a new checkpoint. It must do this before it opens any file it writes as part of the checkpoint.
- The application must call `SCR_Route_file` for each file it writes as part of the checkpoint to register each file and to determine the full path and file name to use to open each file.
- After the application has registered, written, and closed each of its files for the checkpoint, it must call `SCR_Complete_checkpoint` to define the end boundary of the checkpoint. If a process does not write any files during a checkpoint, it must still call `SCR_Start_checkpoint` and
  `SCR_Complete_checkpoint` as these functions are collective. All files registered through a call to `SCR_Route_file` between a given `SCR_Start_checkpoint` and `SCR_Complete_checkpoint` pair are considered to be part of the same checkpoint file set.
- The initialization and finalization of SCR (`SCR_Init` and `SCR_Finalize` functions) will be implicitly handled by MVAPICH2.

The following code snippet describes the checkpointing function of a sample MPI application that writes checkpoints using SCR:

```
/* Determine whether we need to checkpoint */
int flag;
SCR_Need_checkpoint(&flag);
if (flag) {
    /* Tell SCR that a new checkpoint is starting */
    SCR_Start_checkpoint();

    /* Define the checkpoint file name for this process */
    int rank;
    char name[256];
    MPI_Comm_rank(MPI_COMM_WORLD, &rank );
    sprintf (name, "rank_%d.ckpt", rank);

    /* Register our file, and get the full path to open it */
    char file[SCR MAX FILENAME];
    SCR_Route_file name , file);

    /* Open, write, and close the file */
    int valid = 0;
    FILE *fs = open( file, "w") ;
    if ( fs != NULL ) {
        valid = 1;
        size_t n = fwrite(checkpoint_data, 1, sizeof(checkpoint_data), fs);
        if (n != sizeof(checkpoint data)) {
            valid = 0;
        }
        if ( fclose(fs) != 0 ) {
            valid = 0;
        }
    }

    /* Tell SCR that this process is done writing its checkpoint files */
    SCR_Complete_checkpoint( valid ) ;
}
```

The following code snippet describes the SCR-assisted restart mechanism for a sample MPI application:

```
/* Define the checkpoint filename for this process */
int rank;
char name[256];
```

```
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
sprintf( name, "rank_%d.ckpt", rank);
/* Get the full path to open our file */
char file[SCR_MAX_FILENAME];
if (SCR_Route_file( name, file) == SCR SUCCESS) {
    /* Open, read, and close the file */
    FILE *fs = open( file, "r");
    size_t n = fread(checkpoint_data, 1, sizeof(checkpoint_data), fs);
    fclose(fs);
} else {
    /* There is no existing file to restart from */
}
```

For more instructions on integrating the SCR APIs into an MPI application, and for information about the various runtime parameters that can be set by the user, please refer to the SCR Userguide.

Transparent Multi-Level Checkpointing :

The benefits of SCR can also be leveraged by applications that do not employ their own Checkpoint-Restart mechanism. The SCR-MVAPICH2 integration makes this possible by using SCR to manage the checkpoints generated by BLCR the basic system-level Checkpointing scheme described in Section 6.14.1. Once MVAPICH2 has been configured with SCR, the steps required to checkpoint and restart a job transparently using BLCR and SCR are the same as what is described in Section 6.14.1. MVAPICH2 uses SCR's APIs internally to transparently manage checkpoint storage efficiently.

For information about the various SCR-specific runtime parameters that can be set by the user, and for detailed information about the redundancy schemes employed by SCR, please refer to the SCR Userguide.

6.14.3 Job Pause-Migration-Restart Support

MVAPICH2 provides a node-level Job Pause-Migration-Restart mechanism for the OFA-IB-CH3 interface, which can be used to migrate MPI processes on a given failing node to a healthy node, selected from a pool of spare hosts provided by the user during job-launch.

This Job Migration framework relies on BLCR and FTB libraries. See subsections in 6.14.1 to set up these libraries. The source and target nodes are required to have these libraries installed in the same path.

The Job Migration framework makes use of BLCR to take a checkpoint of all MPI processes running on the failing node in question. Users are strongly recommended to either disable "prelinking" feature, or execute the following command:

```
$ prelink --undo --all
```

on all nodes before starting a job that will later be migrated. Please refer to this BLCR web page for complete information: BLCR FAQ.

During job-launch, the list of spare hosts can be provided as input to mpirun_rsh using the `-sparehosts` option which takes a hostfile as argument.

The following example illustrates a sample MPI job being launched to run with job migration support:

```
$ mpirun_rsh -np 4 -hostfile ./hosts -sparehosts
./spare_hosts ./prog
```

where `spare_hosts` is a file which contains a list of healthy spare hosts that is needed by the job migration framework.

An actual migration of MPI processes from a failing source node to a target spare node can be triggered using one of two methods - using signals or using the `mv2_trigger` utility.

For the signal-triggered method, users can manually initiate the migration protocol by issuing a `SIGUSR2` signal to the `mpispawn` processes running on the source node. This can be done using the following command:

```
$ pkill -SIGUSR2 mpispawn
```

The migration protocol can also be triggered using the simple utility provided with MVAPICH2:

```
$ $PREFIX/bin/mv2_trigger src
```

where `src` is the hostname of the health-deteriorating node from which all MPI processes need to be migrated.

Please note, that the `ftb_agent` daemon will have to be launched on all the compute nodes and the spare nodes before launching the MPI application, in order to successfully migrate MPI processes. See section 6.14.1 for information about setting up the FTB infrastructure.

6.14.4 Run-Through Stabilization

MVAPICH2 provides support for run-through stabilization wherein communication failures are not treated as fatal errors. On enabling this capability, MVAPICH2 returns the appropriate error code to a user-set error handler in the event of a communication failure, instead of terminating the entire job. When a process detects a failure when communicating with another process, it will consider the other process as having failed and will no longer attempt to communicate with that process.

The user can, however, continue making communication calls to other processes. Any outstanding send or receive operations to a failed process, or wild-card receives (i.e., with MPI_ANY_SOURCE) posted to communicators with a failed process, will be immediately completed with an appropriate error code.

Currently, this support is available only for the OFA-IB-Nemesis and TCP/IP-Nemesis interfaces, when the Hydra process manager is used.

It can be enabled at run-time by:

- Setting the environment variable `MV2_RUN_THROUGH_STABILIZATION=1` (see section 12.26), and
- passing the `--disable-auto-cleanup` flag to Hydra launcher.

6.14.5 Network Fault Tolerance with Automatic Path Migration

MVAPICH2 supports network fault recovery by using InfiniBand Automatic Path Migration (APM) mechanism for OFA-IB-CH3 interface. This support is available for MPI applications using OpenFabrics stack and InfiniBand adapters.

To enable this functionality, a run-time variable, MV2_USE_APM (Section 11.73) can be enabled, as shown in the following example:

```
$ mpirun_rsh -np 2 n0 n1 MV2_USE_APM=1 ./cpi
```

or

```
$ mpiexec -n 2 -hosts n0,n1 -env MV2_USE_APM 1 ./cpi
```

MVAPICH2 also supports testing Automatic Path Migration in the subnet in the absence of network faults. This can be controlled by using a run-time variable MV2_USE_APM_TEST (Section 11.74). This should be combined with MV2_USE_APM as follows:

```
$ mpirun_rsh -np 2 n0 n1 MV2_USE_APM=1 MV2_USE_APM_TEST=1 ./cpi
```

or

```
$ mpiexec -n 2 -hosts n0,n1 -env MV2_USE_APM 1 -env MV2_USE_APM_TEST 1
./cpi
```

6.15 Running with RDMA CM support

In MVAPICH2, for using RDMA CM the run time variable `MV2_USE_RDMA_CM` needs to be used as described in 11. This applies to OFA-IB-CH3, OFA-iWARP-CH3 and OFA-RoCE-CH3 interfaces.

In addition to these flags, all the systems to be used need the following one time setup for enabling RDMA CM usage.

- **Setup the RDMA CM interface:** RDMA CM interface needs to be setup, configured with an IP address and connected to the network.

- **Setup the Local Address File:** Create mv2.conf file in a standard location (`/etc/mv2.conf`) or user defined path specified using MV2_RDMA_CM_CONF_FILE_PATH (11.86) with the local IP address to be used by RDMA CM. (Multiple IP addresses can be listed (one per line) for multi-rail configurations).
  `$ echo 10.1.1.1 >> /etc/mv2.conf`

Programs can be executed as follows:

```
$ mpirun_rsh -np 2 n0 n1 MV2_USE_RDMA_CM=1 prog
```

or

```
$ mpiexec -n 2 -hosts n0,n1 -env MV2_USE_RDMA_CM 1 prog
```

6.16 Running jobs across multiple InfiniBand subnets

MVAPICH2 relies on RDMA_CM to establish connections with peer processes.

The environment variables `MV2_RDMA_CM_MULTI_SUBNET_SUPPORT` (described in Section 11.85) and `MV2_USE_RDMA_CM` (described in Section 11.84) must be set to 1 (default value is 0) to enable the multi-subnet support in MVAPICH2.

The multi-subnet support in MVAPICH2 relies on the GID of the underlying InfiniBand HCA to establish communication with peer processes. In some scenarios, the GID may be populated in non-default indices in the GID table resulting in failure when trying to establish communication. In such scenarios, users can take advantage of the `MV2_DEFAULT_GID_INDEX` environment variable (described in Section 11.83) to point to the correct index in the GID table.

Programs can be executed as follows if GID is populated in the default location:

```
$ mpirun_rsh -np 2 n0 n1 MV2_USE_RDMA_CM=1 MV2_RDMA_CM_MULTI_SUBNET_SUPPORT=1
prog
```

If the GID is populated in a non-default location (e.g. 1), programs can be executed as follows:

```
$ mpirun_rsh -np 2 n0 n1 MV2_USE_RDMA_CM=1 MV2_RDMA_CM_MULTI_SUBNET_SUPPORT=1
MV2_DEFAULT_GID_INDEX=1 prog
```

6.17 Running MVAPICH2 in Multi-threaded Environments

MVAPICH2 binds processes to processor cores for optimal performance. Please refer to Section 6.5 for more details. However, in multi-threaded environments, it might be desirable to have each thread compute using a separate processor core. This is especially true for OpenMP+MPI programs.

In MVAPICH2, processor core mapping is turned off in the following way to enable the application in MPI_THREAD_MULTIPLE threading level if user requested it in MPI_Init_thread. Otherwise, applications will run in MPI_THREAD_SINGLE threading level.

```
$ mpirun_rsh -np 2 n0 n1 MV2_ENABLE_AFFINITY=0 ./openmp+mpi_app
```

For QLogic PSM Interface,

```
$ mpirun_rsh -np 2 n0 n1 MV2_ENABLE_AFFINITY=0 IPATH_NO_CPUAFFINITY=1
./openmp+mpi_app
```

6.18 Compiler Specific Flags to enable OpenMP thread binding

Further, to get better performance for applications that use MPI + OpenMP, we recommend binding the OpenMP threads to the processor cores. This can potentially avoid cache effects due to unwanted thread migration. For example, if we consider a Quad-Core processor, with 1 MPI process and 4 OpenMP threads, we recommend binding OpenMP thread 0 to core 0, OpenMP thread 1 to core 1 and so on. This can be achieved by setting the kernel affinity via the following compiler-specific environment variables:

Intel Compilers: On Intel processors, we recommend using the KMP_AFFINITY run-time flag in the following manner:

```
$ mpirun_rsh -hostfile hostfile -np 1 MV2_ENABLE_AFFINITY=0
OMP_NUM_THREADS=4
KMP_AFFINITY=warnings,compact ./a.out
```

For more information, please refer to:
http://software.intel.com/sites/products/documentation/studio/composer/en-us/2011Update/compiler_c/optaps/common/optaps_openmp_thread_affinity.htm

GNU compilers: On both Intel and AMD processors, we recommend using the GOMP_CPU_AFFINITY run-time flag, in the following manner:

```
$ mpirun_rsh -hostfile hostfile -np 1 MV2_ENABLE_AFFINITY=0
OMP_NUM_THREADS=4 GOMP_CPU_AFFINITY="0,1,2,3" ./a.out
```

For more information, please refer to:
(http://gcc.gnu.org/onlinedocs/libgomp/GOMP_005fCPU_005fAFFINITY.html)

6.19 Optimizations Specific to Intel Knights Landing (KNL) Processors

We recommend setting the following environment variables:

```
MV2_CPU_BINDING_POLICY=hybrid
MV2_THREADS_PER_PROCESS=<number of threads per process>
```

If MV2_THREADS_PER_PROCESS is not specified, OMP_NUM_THREADS is used to determine the CPU mapping. One core per thread will be allocated to the process. For good performance, we recommend setting MV2_THREADS_PER_PROCESS equal to or more than OMP_NUM_THREADS.

Example with 8 MPI processes and 4 OpenMP threads per process:

```
$ mpirun_rsh -hostfile hostfile -np 8 OMP_NUM_THREADS=4
MV2_CPU_BINDING_POLICY=hybrid MV2_THREADS_PER_PROCESS=4 ./a.out
```

Knights Landing + Omni-Path Architecture: For systems with Knights Landing and Omni-Path/PSM2 architecture, we recommend setting the following environment variable: PSM2_KASSIST_MODE=none in addition.

Example with 8 MPI processes and 4 OpenMP threads per process:

```
$ mpirun_rsh -hostfile hostfile -np 8 OMP_NUM_THREADS=4
MV2_CPU_BINDING_POLICY=hybrid MV2_THREADS_PER_PROCESS=4
PSM2_KASSIST_MODE=none ./a.out
```

6.20 Thread Specific Binding Policies for MPI+Threads Scenarios

MV2_CPU_BINDING_POLICY is used to bind MPI ranks to processor cores. We have also introduced a new variable called MV2_HYBRID_BINDING_POLICY that offers two new policies for thread binding in MPI+Threads execution. The possible bindings offered by this variable are "bunch", "scatter", "linear", "compact", and "spread".

linear — assigns cores to MPI rank and their corresponding threads linearly. For example, first rank followed by its threads, followed by second rank and its corresponding threads and so on.

Example with 8 MPI processes and 4 OpenMP threads per process on a 68-core KNL processor with "linear" binding policy for threads is shown below:

```
$ mpirun_rsh -hostfile hostfile -np 8 OMP_NUM_THREADS=4
MV2_CPU_BINDING_POLICY=hybrid MV2_THREADS_PER_PROCESS=4
MV2_HYBRID_BINDING_POLICY=linear ./a.out
```

```
-------------CPU AFFINITY-------------
RANK: 0  CPU_SET:    0   1   2   3
RANK: 1  CPU_SET:    4   5   6   7
RANK: 2  CPU_SET:    8   9  10  11
RANK: 3  CPU_SET:   12  13  14  15
RANK: 4  CPU_SET:   16  17  18  19
RANK: 5  CPU_SET:   20  21  22  23
RANK: 6  CPU_SET:   24  25  26  27
RANK: 7  CPU_SET:   28  29  30  31
--------------------------------------
```

This will assign ranks and their corresponding threads linearly.

compact — assigns all MPI ranks to physical cores first, and then starts assigning logical cores or hardware threads to each rank's threads in linear fashion. This policy should only be used when the hyper-/hardware-multithreading is enabled and the user intends to map application threads to hardware threads.

Example with 8 MPI processes and 4 OpenMP threads per process on a 68-core KNL processor with "compact" binding policy for threads is shown below:

```
$ mpirun_rsh -hostfile hostfile -np 8 OMP_NUM_THREADS=4
MV2_CPU_BINDING_POLICY=hybrid MV2_THREADS_PER_PROCESS=4
MV2_HYBRID_BINDING_POLICY=compact ./a.out

-------------CPU AFFINITY-------------
RANK: 0  CPU_SET:    0  68 136 204
RANK: 1  CPU_SET:    1  69 137 205
RANK: 2  CPU_SET:    2  70 138 206
RANK: 3  CPU_SET:    3  71 139 207
RANK: 4  CPU_SET:    4  72 140 208
RANK: 5  CPU_SET:    5  73 141 209
RANK: 6  CPU_SET:    6  74 142 210
RANK: 7  CPU_SET:    7  75 143 211
-------------------------------------
```

This will assign one physical core per MPI rank. Since there are four hardware-threads per physical core in KNL, the four OpenMP threads of each rank will be mapped to the hardware threads of the corresponding core.

Note — The correct use of this environment variable mandates that MV2_CPU_BINDING_POLICY should be set to "hybrid" and MV2_THREADS_PER_PROCESS must be a positive integer.

spread — This policy ensures that no two MPI ranks get bound to the same physical core. Moreover, this policy will equally distribute the resources (physical cores and hw/threads) among the MPI processes. It also ensures correct bindings on some clusters that have non-trivial vendor specific mappings.

Example with 8 MPI processes and 4 OpenMP threads per process on a 68-core KNL processor with "spread" binding policy for threads is shown below:

```
$ mpirun_rsh -hostfile hostfile -np 8 OMP_NUM_THREADS=4
MV2_CPU_BINDING_POLICY=hybrid MV2_THREADS_PER_PROCESS=4
MV2_HYBRID_BINDING_POLICY=spread ./a.out

-------------CPU AFFINITY-------------
RANK: 0  CPU_SET:    0   1   2   3
RANK: 1  CPU_SET:    4   5   6   7
RANK: 2  CPU_SET:    8   9  10  11
RANK: 3  CPU_SET:   12  13  14  15
RANK: 4  CPU_SET:   16  17  18  19
RANK: 5  CPU_SET:   20  21  22  23
RANK: 6  CPU_SET:   24  25  26  27
RANK: 7  CPU_SET:   28  29  30  31
-------------------------------------
```

This will assign one MPI rank per physical core avoiding resource contention caused by multiple MPI ranks getting bound to logical cores.

Note — The correct use of this environment variable mandates that MV2_CPU_BINDING_POLICY should be set to "hybrid" and MV2_THREADS_PER_PROCESS must be a positive integer.

bunch — assigns cores to MPI ranks in the same way as "bunch" option described earlier. It additionally takes care of the vendor specific non-trivial core mappings. Further, it also ensures that the MPI ranks are bound to the physical cores only.

scatter — similar to "bunch", this also assigns cores to MPI ranks in "scatter" option given to MV2_CPU_BINDING_POLICY. It additionally takes care of vendor specific non-trivial core mappings. It also ensures that the MPI ranks are only bound to the physical cores only.

Note — "bunch" and "scatter" options for MV2_HYBRID_BINDING_POLICY are recommended to be used as a solution to vendor specific non-trivial mapping issues. These should be considered an alternate to "bunch" or "scatter" given to CPU_BINDING_POLICY.

6.21 Running with Hot-Spot and Congestion Avoidance

MVAPICH2 supports hot-spot and congestion avoidance using InfiniBand multi-pathing mechanism. This support is available for MPI applications using OFA-IB-CH3 interface.

To enable this functionality, a run-time variable, MV2_USE_HSAM (Section 11.79) can be enabled, as shown in the following example:

```
$ mpirun_rsh -np 2 n0 n1 MV2_USE_HSAM=1 ./cpi
```

or

```
$ mpiexec -n 2 -hosts n0,n1 -env MV2_USE_HSAM 1 ./cpi
```

This functionality automatically defines the number of paths for hot-spot avoidance. Alternatively, the maximum number of paths to be used between a pair of processes can be defined by using a run-time variable MV2_NUM_QP_PER_PORT (Section 11.37).

We expect this functionality to show benefits in the presence of at least partially non-overlapping paths in the network. OpenSM, the subnet manager distributed with OpenFabrics supports LMC mechanism, which can be used to create multiple paths:

```
$ opensm -l4
```

will start the subnet manager with LMC value to four, creating sixteen paths between every pair of nodes.

6.22 Running on Clusters with NVIDIA GPU Accelerators

MVAPICH2 CH3-based interface supports MPI communication using NVIDIA GPU device memory with CUDA versions 4.0 or later. This feature removes the need for the application developer to explicitly move the data from device memory to host memory before using MPI for communication. The new support allows direct MPI communication from device memory to device memory, device

memory to host memory and host memory to device memory. It also supports point-to-point and collective communication using contiguous and non-contiguous MPI datatypes. It takes advantage of CUDA IPC for intra-node GPU-GPU communication (with CUDA 4.1).

For example, without CUDA support in the MPI library, a typical user might be using the following sequence of commands to move data from a device memory to another device memory.

```
...
cudaMemcpy(host_buf, device_buf, size, cudaMemcpyDeviceToDevice);
MPI_Isend(host_buf, size, MPI_CHAR, 1, 100, MPI_COMM_WORLD, req);
...
```

With the support provided in MVAPICH2 and support of CUDA 4.0 (and later), the user can achieve the same data movement operation by explicitly specifying MPI calls on device memory.

```
...
MPI_Isend(device_buf, size, MPI_CHAR, 1, 100, MPI_COMM_WORLD, req);
...
```

This support can be enabled by configuring MVAPICH2 with `--enable-cuda` and setting the environment variable MV2_USE_CUDA (11.122) to 1 during runtime.

To minimize communication overhead, MVAPICH2 divides copies between device and host into chunks. This can be better tuned for internode transfers with a runtime environment variable MV2_CUDA_BLOCK_SIZE (11.123). The default chunk size is 64K (65536). However, higher values of this parameter, such as 256K (262144) and 512K (524288), might deliver better performance if the MPI application uses large messages. The optimal value for this parameter depends on several other factors such as InfiniBand network/adapter speed, GPU adapter characteristics, platform characteristics (processor and memory speed) and amount of memory to be dedicated to the MPI library with GPU support. For different platforms and MPI applications, the users are encouraged to try out different values for this parameter to get best performance.

MVAPICH2 takes advantage of tunable two-dimensional CUDA kernel for packing/unpacking data when MPI vector datatype is used in communication involving GPU memory. The CUDA thread block size and dimensions are automatically tuned based on the dimensions of the vector. Users can also control the number of CUDA threads per block using the runtime parameter: MV2_CUDA_KERNEL_VECTOR_TIDBLK_SIZE (default value is 1024) (11.124). They can adjust the number of threads operating on each data block of the vector using MV2_CUDA_KERNEL_VECTOR_YSIZE (tuned based on vector size) (11.125).

GPU Affinity: When multiple GPUs are present on a node, users might want to set the MPI process affinity to a particular GPU using cuda calls like cudaSetDevice(). This can be done after MPI_Init based on MPI rank of the process or before MPI_Init using the local rank of a process obtained from an environment variable MV2_COMM_WORLD_LOCAL_RANK.

6.23 MPIRUN_RSH compatibility with MPIEXEC

There is now a front end to mpirun_rsh that aims for compatibility with mpiexec usage as recommended by the MPI standard. This front end is can be used by calling mpiexec.mpirun_rsh.

The main differences between this and mpirun_rsh is that this front end doesn't require you to specify hosts or environment variables on the command line. Also, the host file options is -f as opposed to -hostfile.

Run 4 instances of CPI on localhost:

```
$ mpiexec.mpirun_rsh -n 4 ./cpi
```

Run 4 instances of CPI on the nodes specified by hostfile:

```
$ mpiexec.mpirun_rsh -n 4 -f hostfile ./cpi
```

6.23.1 Interaction with SLURM

If the hostfile isn't given and mpiexec.mpirun_rsh finds the appropriate slurm environment variables set, mpiexec will use these to determine which nodes the applications should be launched on.

Please note that processes are assigned in a block fashion by slurm so all the processes will run on the same node if possible.

Run 4 instances of CPI on up to 4 different nodes (but maybe only 1) allocated by "salloc":

```
$ salloc -N 4
$ mpiexec.mpirun_rsh -n 4 ./cpi
```

Same thing as above but only allocating the minimum number of nodes needed to run 4 tasks:

```
$ salloc -n 4
$ mpiexec.mpirun_rsh ./cpi
```

Run 4 instances of CPI on 4 different nodes allocated by "salloc":

```
$ salloc -N 4 --ntasks-per-node=1
$ mpiexec.mpirun_rsh -n 4 ./cpi
```

6.23.2 Interaction with PBS

If the hostfile isn't given and mpiexec.mpirun_rsh finds the PBS_NODEFILE environment variable set, mpiexec will use this to locate the host file.

6.24 Running with Intel Trace Analyzer and Collector

MVAPICH2 supports compiling and running MPI applications with Intel Trace Analyzer and Collector (ITAC), so that the MPI calls in an execution can be traced, and the logs can be used for functionality or performance analysis. The following steps need to be done to make an MPI

application work with ITAC.

- Export the environment variables of ITAC. You can either export the entire set of ITAC variables for your architecture by script:

  ```
  $ source /opt/intel/itac/8.1.4/intel64/bin/itacvars.sh
  ```

  or simply export the only variable of VTune root that needed by MVAPICH2:

  ```
  $ export VT_ROOT=/opt/intel/itac/8.1.4
  ```

- Compile your application using `mpicc`, `mpifort` or other compiling commands with the "`-itac`" option. For example:

  ```
  $ mpicc -itac -o cpi cpi.c
  ```

  If configure/Makefile or other building scripts are employed, you should append "`-itac`" to the compiling and linking options such as "`CFLAGS`". For example:

  ```
  $ ./configure CC=mpicc CFLAGS=-itac && make
  ```

- Run the created binary with an MPI launcher in the normal way. After execution, the ITAC prompt information will be printed on stderr, and some trace files will be created in current directory.

- Launch the GUI tool of ITAC on a desktop environment or through X11 forwarding from the server, load the trace file with the "`.stf`" suffix, and the MPI calls within that execution can be visualized and analyzed.

The advanced usage of ITAC can be found at: https://software.intel.com/en-us/intel-trace-analyzer.

6.25 Running with MCDRAM support on Intel Knights Landing (KNL) processor

This section describes the procedure to take advantage of the High Bandwidth Memory (MCDRAM) available with the Intel Knights Landing processor. The method discussed here is basic and does not require modifications to the user application or MPI library.

Allocating Memory on MCDRAM

In order to do all the memory allocations including application buffers and/or internal buffers of MVAPICH2, one can use the *numactl* system call. This is beneficial when KNL is configured in Flat/Hybrid mode where MCDRAM can be viewed as separate NUMA node.

Step 1: Verify that KNL is configured in Flat/Hybrid mode. One can execute the *numactl -H* command to verify this. A sample output is also given below. As can be seen from the sample output, when KNL is in Flat-mode NUMA node 0 consists of all the processor cores and DDR memory (approximately 96GB in size). NUMA node 1 consists of only the MCDRAM (approximately 16GB in size).

```
$ numactl -H
```

```
[hashmij@knl1 pt2pt]$ numactl -H
available: 2 nodes (0-1)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22
23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46
47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70
71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94
95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113
114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131
132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149
150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167
168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185
186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203
204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221
222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239
240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257
258 259 260 261 262 263 264 265 266 267 268 269 270 271
node 0 size: 98200 MB
node 0 free: 94215 MB
node 1 cpus:
node 1 size: 16384 MB
node 1 free: 15929 MB
node distances:
node   0   1
  0:  10  31
  1:  31  10
```

Step 2: Executing applications to take advantage of MCDRAM.

All memory allocations made by the application as well as MVAPICH2 can be redirected to MCDRAM by prepending *numactl –membind <MCDRAM_numa_node>* to the job invocation. An example of the same is given below.

```
$ numactl --membind 1 mpirun_rsh -np 2 -hostfile hosts
osu_latency
```

Viewing MCDRAM Allocations

To verify that the allocations are indeed being done on the MCDRAM, the *numastat -p <pid>* command can be used where "pid" refers to the process ID of one of the application processes. An example of the same is given below.

Note that the pages are not allocated until the memory is actually "touched" by the application and/or the MVAPICH2 MPI library. Thus, '`numastat`' will only report a difference in the amount of memory after the memory is touched.

```
$ numastat -p 21356

Per-node process memory usage (in MBs) for PID 95610 (osu_latency)
```

```
                    Node 0          Node 1           Total
               --------------- --------------- ---------------
Huge                     0.00            0.00            0.00
Heap                     0.00            1.40            1.40
Stack                    0.00            0.08            0.08
Private                  3.11           73.89           77.00
---------------  --------------- --------------- ---------------
Total                    3.11           75.37           78.48
```

6.26 Running Collectives with Hardware based SHArP support

In MVAPICH2, support for SHArP-based collectives has been enabled for MPI applications running over OFA-IB-CH3 interface. Currently, this support is available for the following collective operations:

- MPI_Allreduce

This feature is disabled by default at configuration time. To use this feature, you need to use `--enable-sharp` configuration option and specify the path to SHArP installation location by adding the paths to LDFLAGS and CFLAGS. Also, `-lsharp` and `-lsharp_coll` flags must be added to LDFLAGS. See the example below.

```
./configure --enable-sharp LDFLAGS="-Wl,-rpath,/opt/hpcx/1.8/sharp/lib
-L/opt/hpcx/1.8/sharp/lib -lsharp -lsharp_coll"
CFLAGS="-I/opt/hpcx/1.8/sharp/include/sharp"
```

Please note that SHArP library is a part of HPCX Software Toolkit and the network must be equipped with Mellanox Switch-IB2. Currently, MVAPICH2 supports SHArP library of HPCX v1.7 and v1.8.

Note that the various SHArP related daemons (SHArP Aggregation Manager - *sharpam* and the local SHArP daemon - *sharpd* must be installed, setup and running for SHArP support in MVAPICH2 to work properly. This can be verified by running the `sharp_hello` program available in the "bin" sub-folder of the SHArP installation directory.

This feature is turned off by default at runtime and can be turned on at runtime by using parameter MV2_ENABLE_SHARP=1 (11.129). Collective tuning has been done to ensure the best performance.

When using HPCX v1.7, we recommend setting SHARP_COLL_ENABLE_GROUP_TRIM=0 environment variable. Note that if you are using `mpirun_rsh` you need to add `-export` option to make sure that environment variable is exported correctly. This environment variable is a part of SHArP library. Please refer to SHArP userguide for more information.

7 OSU Benchmarks

If you have arrived at this point, you have successfully installed MVAPICH2. Congratulations!! The OSU benchmarks should already be built and installed along with MVAPICH2. Look for them under $prefix/libexec/mvapich2. Sample performance numbers for these benchmarks on representative platforms with InfiniBand, iWARP and RoCE adapters are also included on our projects' web page. You are welcome to compare your performance numbers with our numbers. If you see any big discrepancy, please let us know by sending an email to mvapich-discuss@cse.ohio-state.edu.

The benchmarks provided are:

MPI-1, MPI-2 and MPI-3

osu_bibw	Bidirectional Bandwidth Test
osu_bw	Bandwidth Test
osu_latency	Latency Test
osu_mbw_mr	Multiple Bandwidth / Message Rate Test
osu_multi_lat	Multi-pair Latency Test
osu_allgather	MPI_Allgather Latency Test
osu_allgatherv	MPI_Allgatherv Latency Test
osu_allreduce	MPI_Allreduce Latency Test
osu_alltoall	MPI_Alltoall Latency Test
osu_alltoallv	MPI_Alltoallv Latency Test
osu_barrier	MPI_Barrier Latency Test
osu_bcast	MPI_Bcast Latency Test
osu_gather	MPI_Gather Latency Test
osu_gatherv	MPI_Gatherv Latency Test
osu_reduce	MPI_Reduce Latency Test
osu_reduce_scatter	MPI_Reduce_scatter Latency Test
osu_scatter	MPI_Scatter Latency Test
osu_scatterv	MPI_Scatterv Latency Test

MPI-2 and MPI-3 only

osu_acc_latency	Accumulate Latency Test with Active/Passive Synchronization
osu_get_bw	One-Sided Get Bandwidth Test with Active/Passive Synchronization
osu_get_latency	One-Sided Get Latency Test with Active/Passive Synchronization
osu_latency_mt	Multi-threaded Latency Test
osu_put_bibw	One-Sided Put Bidirectional Test with Active Synchronization
osu_put_bw	One-Sided Put Bandwidth Test with Active/Passive Synchronization
osu_put_latency	One-Sided Put Latency Test with Active/Passive Synchronization

MPI-3 only

osu_cas_latency	One-Sided Compare_and_swap Latency Test with Active/Passive Synchronization
osu_fop_latency	One-Sided Fetch_and_op Latency Test with Active/Passive Synchronization
osu_get_acc_latency	One-Sided Get_accumulate Latency Test with Active/Passive Synchronization
osu_iallgather	MPI_Iallgather Latency Test
osu_ialltoall	MPI_Ialltoall Latency Test
osu_ibarrier	MPI_Ibarrier Latency Test
osu_ibcast	MPI_Ibcast Latency Test
osu_igather	MPI_Igather Latency Test
osu_iscatter	MPI_Iscatter Latency Test

More information about the benchmarks can be found at http://mvapich.cse.ohio-state.edu/benchmarks/. You can also check this link for updates to the benchmarks.

7.1 Download and Build Stand-alone OSU Benchmarks Package

The OSU Benchmarks can also be downloaded as a separate package from here. You can build the benchmarks using the following steps if mpicc is in your PATH. For example:

```
$ ./configure --prefix=<path-to-install> && make && make install
```

If mpicc is not in your path or you would like to use another particular version you can explicitly tell configure by setting CC. For example:

```
$ ./configure CC=/path/to/special/mpicc --prefix=<path-to-install> &&
make && make install
```

Configure will detect whether your library supports MPI-2, MPI-3 and compile the corresponding benchmarks. The benchmarks will be installed under $prefix/libexec/osu-micro-benchmarks.

CUDA Extensions to OMB can be enabled by configuring the benchmark suite with –enable-cuda option as shown below. Similarly, OpenACC Extensions can be enabled by specifying the –enable-openacc option. The MPI library used should be able to support MPI communication from buffers in GPU Device memory.

```
$ ./configure CC=/path/to/mpicc
        --enable-cuda
        --with-cuda-include=/path/to/cuda/include
        --with-cuda-lib=/path/to/cuda/lib
$ make
$ make install
```

7.2 Running

The OSU Benchmarks are run in the same manner as other MPI Applications. The following examples will use mpirun_rsh as the process manager. Please see section 5.2 for more information on running with other process managers.

7.2.1 Running OSU Latency and Bandwidth

Inter-node latency and bandwidth: The following example will measure the latency and bandwidth of communication between node1 and node2.

```
$ mpirun_rsh -np 2 node1 node2 ./osu_latency
```

```
$ mpirun_rsh -np 2 node1 node2 ./osu_bw
```

Intra-node latency and bandwidth: The following example will measure the latency and bandwidth of communication inside node1 on different cores. This assumes that you have at least two cores (or processors) in your node.

```
$ mpirun_rsh -np 2 node1 node1 ./osu_latency
```

```
$ mpirun_rsh -np 2 node1 node1 ./osu_bw
```

7.2.2 Running OSU Message Rate Benchmark

The OSU message rate benchmark reports the rate at which messages can be sent between two nodes. It is advised that it should be run in a configuration that utilizes multiple pairs of communicating processes on two nodes. The following example measures the message rate on a system with two nodes, each with four processor cores.

```
$ mpirun_rsh -np 8 -hostfile mf ./osu_mbw_mr
```

Where hostfile “mf” has the following contains:

node1
node1
node1
node1
node2
node2
node2
node2

7.2.3 Running OSU Collective Benchmarks

By default, the OSU collective benchmarks report the average communication latencies for a given collective operation, across various message lengths. Additionally, the benchmarks offer the following options:

1. ”-f” can be used to report additional statistics, such as min and max latencies and the number of iterations.
2. ”-m” option can be used to set the maximum message length to be used in a benchmark. In the default version, the benchmarks report the latencies for up to 1MB message lengths.
3. ”-i” can be used to set the number of iterations to run for each message length.
4. ”-h” can be used to list all the options and their descriptions.
5. ”-v” reports the benchmark version.

If a user wishes to measure the communication latency of a specific collective operation, say, MPI_Alltoall, with 16 processes, we recommend running the osu_alltoall benchmark in the following manner:

```
$ mpirun_rsh -np 16 -hostfile mf ./osu_alltoall
```

Where hostfile “mf” has the following contains:

node1
node1
node1
node1
node1
node1
node1
node1
node1
node2
node2
node2
node2
node2
node2
node2
node2
node2

7.2.4 Running Benchmarks with CUDA/OpenACC Extensions

The following benchmarks have been extended to evaluate performance of MPI communication from and to buffers on NVIDIA GPU devices.

osu_bibw	Bidirectional Bandwidth Test
osu_bw	Bandwidth Test
osu_latency	Latency Test
osu_allgather	MPI_Allgather Latency Test
osu_allgatherv	MPI_Allgatherv Latency Test
osu_allreduce	MPI_Allreduce Latency Test
osu_alltoall	MPI_Alltoall Latency Test
osu_alltoallv	MPI_Alltoallv Latency Test
osu_bcast	MPI_Bcast Latency Test
osu_gather	MPI_Gather Latency Test
osu_gatherv	MPI_Gatherv Latency Test
osu_reduce	MPI_Reduce Latency Test
osu_reduce_scatter	MPI_Reduce_scatter Latency Test
osu_scatter	MPI_Scatter Latency Test
osu_scatterv	MPI_Scatterv Latency Test

Each of the pt2pt benchmarks takes two input parameters. The first parameter indicates the location of the buffers at rank 0 and the second parameter indicates the location of the buffers at rank 1. The value of each of these parameters can be either 'H' or 'D' to indicate if the buffers are to be on the host or on the device, respectively. When no parameters are specified, the buffers are allocated on the host.

The collective benchmarks will use buffers allocated on the device if the -d option is used otherwise the buffers will be allocated on the host.

If both CUDA and OpenACC support is enabled, you can switch between using CUDA and OpenACC to allocate your device buffers by specifying the '-d cuda' or '-d openacc' option to the benchmark. Please use the '-h' option for more info.

GPU affinity for processes is set before MPI_Init is called. The process rank on a node is normally used to do this and different MPI launchers expose this information through different environment variables. The benchmarks use an environment variable called LOCAL_RANK to get this information. The script like get_local_rank provided alongside the benchmarks can be used to export this environment variable when using mpirun_rsh. This can be adapted to work with other MPI launchers and libraries.

Examples:

```
$ mpirun_rsh -np 2 host0 host0 MV2_USE_CUDA=1
get_local_rank ./osu_latency D D
```

In this run, assuming host0 has two GPU devices, the latency test allocates buffers on GPU device 0 at rank 0 and on GPU device 1 at rank 1.

```
$ mpirun_rsh -np 2 host0 host1 MV2_USE_CUDA=1
get_local_rank ./osu_bw D H
```

In this run, the bandwidth test allocates buffers on the GPU device at rank 0 (host0) and on the host at rank 1 (host1).

8 Scalability features and Performance Tuning for Large Scale Clusters

MVAPICH2 provides many different parameters for tuning performance for a wide variety of platforms and applications. This section deals with tuning CH3-based interfaces. These parameters can be either compile time parameters or runtime parameters. Please refer to Section 8 for a complete description of all these parameters. In this section we classify these parameters depending on what you are tuning for and provide guidelines on how to use them.

8.1 Optimizations for homogeneous clusters

MVAPICH2 internally detects the heterogeneity of the cluster in terms of processor and network interface type. Set parameter MV2_HOMOGENEOUS_CLUSTER to 1 to skip this detection, if the user already knows that cluster is homogeneous.

8.2 Improving Job startup performance

MVAPICH2 has several advanced features to speed up launching jobs on HPC clusters. There are several launcher-agnostic and launcher-specific parameters that can be used to get the best job startup performance. More details about these designs can be obtained from: http://mvapich.cse.ohio-state.edu/performance/job-startup/

8.2.1 Configuration Options (Launcher-Agnostic)

- Disabling RDMA_CM
  - Default: Enabled
  - Disable: `--disable-rdma-cm`
  - Disabling RDMA_CM will improve job-startup performance, particularly on nodes with large number of cores.

8.2.2 Runtime Parameters (Launcher-Agnostic)

- MV2_HOMOGENEOUS_CLUSTER
  - Default: 0 (Disabled)
  - Setting MV2_HOMOGENEOUS_CLUSTER to 1 on homogeneous clusters will improve startup performance.
- MV2_ON_DEMAND_THRESHOLD
  - Default: 64 (OFA-IB-CH3), 16 (OFA-iWARP-CH3)
  - Should be enabled for fast startup. See Section 11.43 for details.

- MV2_ON_DEMAND_UD_INFO_EXCHANGE
  - Default: 1 (Enabled)
  - Setting MV2_ON_DEMAND_UD_INFO_EXCHANGE to 1 will enable on-demand Address Handle creation for hybrid mode.

8.2.3 Enabling Optimizations Specific to mpirun_rsh

- **MV2_MT_DEGREE**: MVAPICH2 has a scalable job launcher – mpirun_rsh which uses a tree based mechanism to spawn processes. The degree of this tree is determined dynamically to keep the depth low. For large clusters, it might be beneficial to further flatten the tree by specifying a higher degree. The degree can be overridden with the environment variable MV2_MT_DEGREE (see 13.8).
- **MV2_FASTSSH_THRESHOLD**: MVAPICH2 can use a faster, hierarchical launching mechanism on large clusters. This is enabled manually using MV2_FASTSSH_THRESHOLD (see 13.5).
- **MV2_NPROCS_THRESHOLD**: When the number of nodes involved is beyond 8k, the mpirun_rsh uses a file-based communication scheme to create the hierarchical tree. The default value can be overridden with the environment variable MV2_NPROCS_THRESHOLD (see 13.6).

8.2.4 Enabling Optimizations Specific to SLURM

- **Using OSU optimized SLURM**: For best performance with SLURM, the OSU-optimized PMI2 plugin should be used. This requires applying the appropriate patch to SLURM. Please refer to Section 4.3.3 for more details.
- **Using Default SLURM**: If the SLURM installation cannot be modified, the default PMI2 plugin provided by SLURM should be used. Please see section 4.3.2 for more details.

8.3 Basic QP Resource Tuning

The following parameters affect memory requirements for each QP.

- MV2_DEFAULT_MAX_SEND_WQE
- MV2_DEFAULT_MAX_RECV_WQE
- MV2_MAX_INLINE_SIZE

MV2_DEFAULT_MAX_SEND_WQE and MV2_DEFAULT_MAX_RECV_WQE control the maximum number of WQEs per QP and MV2_MAX_INLINE_SIZE controls the maximum inline size. Reducing the values of these two parameters leads to less memory consumption. They are especially important for large scale clusters with a large amount of connections and multiple rails.

These two parameters are run-time adjustable. Please refer to Sections 11.18 and 11.30 for details.

8.4 RDMA Based Point-to-Point Tuning

The following parameters are important in tuning the memory requirements for adaptive rdma fast path feature.

- MV2_RDMA_FAST_PATH_BUF_SIZE (11.41)
- MV2_NUM_RDMA_BUFFER (11.42)

MV2_RDMA_FAST_PATH_BUF_SIZE is the size of each buffer used in RDMA fast path communication.

MV2_NUM_RDMA_BUFFER is number of buffers used for the RDMA fast path communication.

On the other hand, the product of MV2_RDMA_FAST_PATH_BUF_SIZE and MV2_NUM_RDMA_BUFFER generally is a measure of the amount of memory registered for eager message passing. These buffers are not shared across connections.

8.5 Shared Receive Queue (SRQ) Tuning

The main environmental parameters controlling the behavior of the Shared Receive Queue design are:

- MV2_SRQ_MAX_SIZE (11.69)
- MV2_SRQ_SIZE (11.70)
- MV2_SRQ_LIMIT (11.68)

MV2_SRQ_MAX_SIZE is the maximum size of the Shared Receive Queue (default 4096). You may increase this to value 8192 if the application requires very large number of processors. The application will start by only using MV2_SRQ_SIZE buffers (default 256) and will double this value on every SRQ limit event(upto MV2_SRQ_MAX_SIZE). For long running applications this re-size should show little effect. If needed, the MV2_SRQ_SIZE can be increased to 1024 or higher as needed for applications.
MV2_SRQ_LIMIT defines the low water-mark for the flow control handler. This can be reduced if your aim is to reduce the number of interrupts.

8.6 eXtended Reliable Connection (XRC)

MVAPICH2 now supports the eXtended Reliable Connection (XRC) transport available in recent Mellanox HCAs. This transport helps reduce the number of QPs needed on multi-core systems. Set MV2_USE_XRC (11.102) to use XRC with MVAPICH2.

8.7 Shared Memory Tuning

MVAPICH2 uses shared memory communication channel to achieve high-performance message passing among processes that are on the same physical node. The two main parameters which are used for tuning shared memory performance for small messages are SMPI_LENGTH_QUEUE (Section 11.107), and SMP_EAGER_SIZE (Section 11.106). The two main parameters which are used for tuning shared memory performance for large messages are SMP_SEND_BUF_SIZE (Section 11.109) and
SMP_NUM_SEND_BUFFER (Section 11.108).

SMPI_LENGTH_QUEUE is the size of the shared memory buffer which is used to store outstanding small and control messages. SMP_EAGER_SIZE defines the switch point from Eager protocol to Rendezvous protocol.

Messages larger than SMP_EAGER_SIZE are packetized and sent out in a pipelined manner. SMP_SEND_BUF_SIZE is the packet size, i.e. the send buffer size. SMP_NUM_SEND_BUFFER is the number of send buffers.

8.8 On-demand Connection Management Tuning

MVAPICH2 uses on-demand connection management to reduce the memory usage of MPI library. There are 4 parameters to tune connection manager: MV2_ON_DEMAND_THRESHOLD (Section 11.43),
MV2_CM_RECV_BUFFERS (Section 11.10), MV2_CM_TIMEOUT (Section 11.12), and
MV2_CM_SPIN_COUNT (Section 11.11). The first one applies to OFA-IB-CH3 and OFA-iWARP-CH3 interfaces and the other three only apply to OFA-IB-CH3 interface.

MV2_ON_DEMAND_THRESHOLD defines threshold for enabling on-demand connection management scheme. When the size of the job is larger than the threshold value, on-demand connection management will be used.

MV2_CM_RECV_BUFFERS defines the number of buffers used by connection manager to establish new connections. These buffers are quite small and are shared for all connections, so this value may be increased to 8192 for large clusters to avoid retries in case of packet drops.

MV2_CM_TIMEOUT is the timeout value associated with connection management messages via UD channel. Decreasing this value may lead to faster retries but at the cost of generating duplicate messages.

MV2_CM_SPIN_COUNT is the number of the connection manager polls for new control messages from UD channel for each interrupt. This may be increased to reduce the interrupt overhead when many incoming control messages from UD channel at the same time.

8.9 Scalable Collectives Tuning

MVAPICH2 uses shared memory to optimize the performance for many collective operations: MPI_Allreduce, MPI_Reduce, MPI_Barrier, and MPI_Bcast.

We use shared-memory based collective for most small and medium sized messages and fall back to the default point-to-point based algorithms for very large messages. The upper-limits for shared-memory based collectives are tunable parameters that are specific to each collective operation. We have variables such as MV2_SHMEM_ALLREDUCE_MSG (11.58), MV2_SHMEM_REDUCE_MSG (11.64) and
MV2_SHMEM_BCAST_MSG (11.60), for MPI_Allreduce, MPI_Reduce and MPI_Bcast collective operations. The default values for these variables have been set through experimental analysis on some of our clusters and a few large scale clusters, such as the TACC Ranger. Users can choose to set these variables at job-launch time to tune the collective algorithms on different platforms.

8.9.1 Optimizations for MPI_Bcast

MVAPICH2 supports a 2-level point-to-point tree-based "Knomial" algorithm for small messages for the MPI_Bcast operation. MVAPICH2 also offers improved designs that deliver better performance for medium and large message lengths.

8.9.2 Optimizations for MPI_Reduce and MPI_Allreduce

In this release, we have introduced new 2-level algorithms for MPI_Reduce and MPI_Allreduce operations, along the same lines as MPI_Bcast. Pure Shared-memory based algorithms cannot be used for larger messages for reduction operations, because the node-leader processes become the bottleneck, as they have to perform the reduction operation on the entire data block. We now rely on shared-memory algorithms for MPI_Reduce and MPI_Allreduce for message sizes set by the thresholds MV2_SHMEM_ALLREDUCE_MSG (11.58), MV2_SHMEM_REDUCE_MSG (11.64), the new 2-level algorithms for medium sized messages and the default point-to-point based algorithms for large messages. We have introduced two new run-time variables MV2_ALLREDUCE_2LEVEL_MSG (11.1) and MV2_REDUCE_2LEVEL_MSG (11.54) to determine when to fall back to the default point-to-point based algorithms.

8.9.3 Optimizations for MPI_Gather and MPI_Scatter

MVAPICH2 supports two new optimized algorithms for MPI_Gather and MPI_Scatter operations – the "Direct" and the multi-core aware "2-level" algorithms. Both these algorithms perform significantly better than the default binomial-tree pattern. The "Direct" algorithm is however inherently not very scalable and can be used when the communicator size is less than 1K processes. We switch over to the 2-level algorithms for larger system sizes. For MPI_Gather, we use different algorithms depending of the system size. For small system sizes (up to 386 cores), we use the "2-level" algorithm following by the "Direct" algorithm. For medium system sizes (up to 1k), we use "Binomial" algorithm following by the "Direct" algorithm. It's possible to set the switching point between algorithms using the run-time parameter MV2_GATHER_SWITCH_PT (11.97). For MPI_Scatter, when the system size is lower than 512 cores, we use the "Binomial" algorithm for small message sizes following by the "2-level" algorithm for medium message sizes and the "Direct" algorithm for large message sizes. Users can define the threshold for small and medium message sizes using the run-time parameters MV2_SCATTER_SMALL_MSG (11.98) and MV2_SCATTER_MEDIUM_MSG (11.99).

Users can also choose to use only one of these algorithms by toggling the run-time parameters 11.77 and 11.100 for MPI_Gather and 11.78 and 11.101 for MPI_Scatter.

8.10 Process Placement on Multi-core platforms

Process placement has a significant impact on performance of applications. Depending on your application communication patterns, various process placements can result in performance gains. In Section 6.5, we have described the usage of "bunch" and "scatter" placement modes provided by MVAPICH2. Using these modes, one can control the placement of processes within a particular node. Placement of processes across nodes can be controlled by adjusting the order of MPI ranks. For example, the following command launches jobs in block fashion.

```
$ mpirun_rsh -np 4 n0 n0 n1 n1 MV2_CPU_BINDING_POLICY=bunch ./a.out
```

The following command launches jobs in a cyclic fashion.

```
$ mpirun_rsh -np 4 n0 n1 n0 n1 MV2_CPU_BINDING_POLICY=scatter ./a.out
```

We have noted that the HPL (High-Performance Linpack) benchmark performs better when using block distribution.

8.11 HugePage Support

MVAPICH2 uses HugePages(2MB) by default for communication buffers if they are configured on the system. The run-time variable, MV2_USE_HUGEPAGES(11.110) can be used to control the behavior of this feature.

In order to use HugePages, Make sure HugePages are configured on all nodes. The number of HugePages can be configured by setting vm.nr_hugepages kernel parameter to a suitable value. For example, to allocate a 1GB HugePage pool, execute(as root):

```
$ echo 512 > /proc/sys/vm/nr_hugepages
or
$ sysctl -w vm.nr_hugepages = 512
```

9 FAQ and Troubleshooting with MVAPICH2

Based on our experience and feedback we have received from our users, here we include some of the problems a user may experience and the steps to resolve them. If you are experiencing any other problem, please feel free to contact us by sending an email to mvapich-discuss@cse.ohio-state.edu.

MVAPICH2 can be used over eight underlying interfaces, namely OFA-IB-CH3, OFA-IB-Nemesis, OFA-IWARP-CH3, OFA-RoCE-CH3, TrueScale (PSM-CH3), Omni-Path (PSM2-CH3), TCP/IP-CH3 and TCP/IP-Nemesis. Based on the underlying library being utilized, the troubleshooting steps may be different. We have divided the troubleshooting tips into four sections: General troubleshooting and Troubleshooting over any one of the five transport interfaces.

9.1 General Questions and Troubleshooting

9.1.1 Issues with MVAPICH2 and Python based MPI programs

Using an application written using Python with MVAPICH2 can potentially result in the memory registration cache mechanism in MVAPICH2 being disabled at runtime due to an interaction between the Python memory allocator and the memory registration cache mechanism in MVAPICH2. We are working towards resolving this issue. In the mean time, we recommend that you try the following work arounds:

- Use the LD_PRELOAD environment variable to export the path of the MVAPICH2 shared library object (libmpich.so or libmpi.so). eg: export LD_PRELOAD=/path/to/libmpi.so
- Increase the size of the internal communication buffer being used by MVAPICH2 and the switch point between eager and rendezvous protocol in MVAPICH2 to a larger value. Please refer to Section 9.1.3 for more details.

9.1.2 Issues with MVAPICH2 and Google TCMalloc

Using an application that utilizes the Google TCMalloc library with MVAPICH2 can potentially result in issues at compile time and/or runtime due to an interaction between the Google TCMalloc library and the memory registration cache mechanism in MVAPICH2. We are working towards resolving this issue. In the mean time, we recommend that you disable the memory registration cache mechanism in MVAPICH2 to work-around this issue. MVAPICH2 has the capability to memory disable registration cache support at configure / build time and also at runtime.

If the issue you are facing is at compile time, then you need to re-configure the MVAPICH2 library to disable memory registration cache at configure time using the "–disable-registration-cache" option. Please refer to Section 4.4 of the MVAPICH2 userguide for more details on how to disable registration cache at build time.

If the issue you are facing is at run time, then please re-run your application after setting "MV2_USE_LAZY_MEM_UNREGISTER=0". Please refer to Section 11.81 of the MVAPICH2 userguide for more details on how to disable registration cache at run time.

Please refer to Section 9.1.3 for more details on the impact of disabling memory registration cache on application performance.

9.1.3 Impact of disabling memory registration cache on application performance

Whether disabling registration cache will have a negative effect on application performance depends entirely on the communication pattern of the application. If the application uses mostly small to medium sized messages (approximately less than 16 KB), then disabling registration cache will mostly have no impact on the performance of the application.

However, if the application uses messages of larger size, then there might be an impact depending on the frequency of communication. If this is the case, then it might be useful to increase the size of the internal communication buffer being used by MVAPICH2 (using the "MV2_VBUF_TOTAL_SIZE" environment variable) and the switch point between eager and rendezvous protocol in MVAPICH2 (using the "MV2_IBA_EAGER_THRESHOLD") to a larger value. In this scenario, we recommend that you set both to the same value (possibly slightly greater than the median message size being used by your application). Please refer to Sections 11.24 and 11.105 of the userguide for more information about these two parameters.

9.1.4 MVAPICH2 failed to register memory with InfiniBand HCA

OFED provides network vendor specific kernel module parameters to control the size of Memory translation table(MTT) used to map virtual to physical address. This will limit the amount of physical memory can be registered with InfiniBand device. The following two parameters are provided to control the size of this table.

1. log_num_mtt
2. log_mtts_per_seg

The amount of memory that can be registered is calculated by

`max_reg_mem =` $(2^{log\_num\_mtt})$ `*` $(2^{log\_mtts\_per\_seg})$ `* PAGE_SIZE`

It is recommended to adjust log_num_mtt to allow at least twice the amount of physical memory on your machine. For example, if a node has 64 GB of memory and a 4 KB page size, log_num_mtt should be set to 24 and (assuming log_mtts_per_seg is set to 1)

These parameters are set on the mlx4_core module in /etc/modprobe.conf

```
options mlx4_core log_num_mtt=24
```

9.1.5 Invalid Communicators Error

This is a problem which typically occurs due to the presence of multiple installations of MVAPICH2 on the same set of nodes. The problem is due to the presence of `mpi.h` other than the one, which

is used for executing the program. This problem can be resolved by making sure that the `mpi.h` from other installation is not included.

9.1.6 Are `fork()` and `system()` supported?

`fork()` and `system()` is supported for the OpenFabrics device as long as the kernel is being used is Linux 2.6.16 or newer. Additionally, the version of OFED used should be 1.2 or higher. The environment variable IBV_FORK_SAFE=1 must also be set to enable fork support.

9.1.7 MPI+OpenMP shows bad performance

MVAPICH2 uses CPU affinity to have better performance for single-threaded programs. For multi-threaded programs, e.g. MPI+OpenMP, it may schedule all the threads of a process to run on the same CPU. CPU affinity should be disabled in this case to solve the problem, i.e. set `MV2_ENABLE_AFFINITY` to 0. In addition, please read Section 6.17 on using MVAPICH2 in multi-threaded environments. We also recommend using the compiler/platform specific run-time options to bind the OpenMP threads to processors. Please refer to Section (6.18) for more information.

9.1.8 Error message "No such file or directory" when using Lustre file system

If you are using ADIO support for Lustre, please make sure of the following:
– Check your Lustre setup
– You are able to create, read to and write from files in the Lustre mounted directory
– The directory is mounted on all nodes on which the job is executed
– The path to the file is correctly specified
– The permissions for the file or directory are correctly specified

9.1.9 Program segfaults with "File locking failed in ADIOI_Set_lock"

If you are using ADIO support for Lustre, the recent Lustre releases require an additional mount option to have correct file locks. Please include the following option with your Lustre mount command: "-o localflock".

```
$ mount -o localflock -t lustre xxxx@o2ib:/datafs /mnt/datafs
```

9.1.10 Running MPI programs built with gfortran

MPI programs built with gfortran might not appear to run correctly due to the default output buffering used by gfortran. If it seems there is an issue with program output, the `GFORTRAN_UNBUFFERED_ALL` variable can be set to "y" and exported into the environment before using the `mpiexec` or `mpirun_rsh` command to launch the program, as below:

```
$ export GFORTRAN_UNBUFFERED_ALL=y
```

Or, if using `mpirun_rsh`, export the environment variable as in the example:

```
$ mpirun_rsh -np 2 n1 n2 GFORTRAN_UNBUFFERED_ALL=y ./a.out
```

9.1.11 How do I obtain MVAPICH2 version and configuration information?

The `mpiname` application is provided with MVAPICH2 to assist with determining the MPI library version and related information. The usage of `mpiname` is as follows:

```
$ mpiname [OPTION]
```

Print MPI library information. With no OPTION, the output is the same as -v.

-a print all information

-c print compilers

-d print device

-h display this help and exit

-n print the MPI name

-o print configuration options

-r print release date

-v print library version

9.1.12 How do I compile my MPI application with static libraries, and not use shared libraries?

MVAPICH2 is configured to be built with shared-libraries by default. To link your application to the static version of the library, use the command below when compiling your application:

```
$ mpicc -noshlib -o cpi cpi.c
```

9.1.13 Does MVAPICH2 work across AMD and Intel systems?

Yes, as long as you compile MVAPICH2 and your programs on one of the systems, either AMD or Intel, and run the same binary across the systems. MVAPICH2 has platform specific parameters for performance optimizations and it may not work if you compile MVAPICH2 and your programs on different systems and try to run the binaries together.

9.1.14 I want to enable debugging for my build. How do I do this?

We recommend that you enable debugging when you intend to take a look at back traces of processes in GDB (or other debuggers). You can use the following configure options to enable debugging: `--enable-g=dbg --disable-fast`.

Additionally:

- See parameter `MV2_DEBUG_CORESIZE` (section 10.4) to enable core dumps.
- See parameter `MV2_DEBUG_SHOW_BACKTRACE` (section 10.5) to show a basic backtrace in case of error.

9.1.15 How can I run my application with a different group ID?

You can specify a different group id for your MPI application using the `-sg group` option to `mpirun_rsh`. The following example executes *a.out* on host1 and host2 using *secondarygroup* as their group id.

```
$ mpirun_rsh -sg secondarygroup -np 2 host1 host2 ./a.out
```

9.2 Issues and Failures with Job launchers

9.2.1 /usr/bin/env: mpispawn: No such file or directory

If `mpirun_rsh` fails with this error message, it was unable to locate a necessary utility. This can be fixed by ensuring that all MVAPICH2 executables are in the PATH on all nodes. If PATHs cannot be setup as mentioned, then invoke mpirun_rsh with a path prefix. For example:

```
$ /path/to/mpirun_rsh -np 2 node1 node2 ./mpi_proc
```

9.2.2 TotalView complains that "The MPI library contains no suitable type definition for struct MPIR_PROCDESC"

Ensure that the MVAPICH2 job launcher mpirun_rsh is compiled with debug symbols. Details are available in Section 5.2.10.

9.3 Problems Building MVAPICH2

9.3.1 Unable to convert MPI_SIZEOF_AINT to a hex string

```
configure: error: Unable to convert MPI_SIZEOF_AINT to a hex string.
This is either because we are building on a very strange platform or there is a bug
somewhere in configure.
```

This error can be misleading. The problem is often not that you're building on a strange platform, but that there was some problem running an executable that made configure have trouble determining the size of a datatype. The true problem is often that you're trying to link against a library that is not found in your system's default path for linking at runtime. Please check that you've properly set LD_LIBRARY_PATH or used the correct rpath settings in LDFLAGS.

9.3.2 Cannot Build with the PathScale Compiler

There is a known bug with the PathScale compiler (before version 2.5) when building MVAPICH2. This problem will be solved in the next major release of the PathScale compiler. To work around this bug, use the the "`-LNO:simd=0`" C compiler option. This can be set in the build script similarly to:

```
export CC="pathcc -LNO:simd=0"
```

Please note the use of double quotes. If you are building MVAPICH2 using the PathScale compiler (version below 2.5), then you should add "-g" to your CFLAGS, in order to get around a compiler bug.

9.3.3 nvlink fatal : Unsupported file type '../lib/.libs/libmpich.so'

There have been recent reports of this issue when using the PGI compiler. This may be able to be solved by adding "-ta=tesla:nordc" to your CFLAGS. The following example shows MVAPICH2 being configured with the proper CPPFLAGS and CFLAGS to get around this issue (Note: –enable-cuda=basic is optional).

Example:
```
./configure --enable-cuda=basic CPPFLAGS="-D__x86_64
    -D__align__\(n\)=__attribute__\(\(aligned\(n\)\)\)
    -D__location__\(a\)=__annotate__\(a\)
    -DCUDARTAPI="
    CFLAGS="-ta=tesla:nordc"
```

9.3.4 Libtool has a problem linking with non-GNU compiler (like PGI)

If you are using a compiler that is not recognized by autoconf as a GNU compiler, Libtool uses an default library search path to look for shared objects which is "/lib /usr/lib /usr/local/lib". Then, if your libraries are not in one of these paths, MVAPICH2 may fail to link properly.

You can work around this issue by adding the following configure flags:

```
./configure \
lt_cv_sys_lib_search_path_spec="/lib64 /usr/lib64 /usr/local/lib64"   \
lt_cv_sys_lib_dlsearch_path_spec="/lib64 /usr/lib64 /usr/local/lib64" \
... ...
```

The above example considers that the correct library search path for your system is "/lib64 /usr/lib64 /usr/local/lib64".

9.4 With OFA-IB-CH3 Interface

9.4.1 Cannot Open HCA

The above error reports that the InfiniBand Adapter is not ready for communication. Make sure that the drivers are up. This can be done by executing the following command which gives the path at which drivers are setup.

```
$ locate libibverbs
```

9.4.2 Checking state of IB Link

In order to check the status of the IB link, one of the OFED utilities can be used: `ibstatus`, `ibv_devinfo`.

9.4.3 Creation of CQ or QP failure

A possible reason could be inability to pin the memory required. Make sure the following steps are taken.

1. In `/etc/security/limits.conf` add the following

   ```
   * soft memlock phys_mem_in_KB
   ```

2. After this, add the following to `/etc/init.d/sshd`

   ```
   ulimit -l  phys_mem_in_KB
   ```

3. Restart sshd

With some distros, we've found that adding the ulimit -l line to the sshd init script is no longer necessary. For instance, the following steps work for our RHEL5 systems.

1. Add the following lines to `/etc/security/limits.conf`

   ```
   * soft memlock unlimited
   * hard memlock unlimited
   ```

2. Restart sshd

9.4.4 Hang with Multi-rail Configuration

If your system has multiple HCAs per node, ensure that the number of HCAs per node is the same across all nodes. Otherwise, specify the correct number of HCAs using the parameter 11.33.

If you configure MVAPICH2 with `RDMA_CM` and see this issue, ensure that the different HCAs on the same node are on different subnets.

9.4.5 Hang with the HSAM Functionality

HSAM functionality uses multi-pathing mechanism with LMC functionality. However, some versions of OpenFabrics Drivers (including OpenFabrics Enterprise Distribution (OFED) 1.1) and using the Up*/Down* routing engine do not configure the routes correctly using the LMC mechanism. We strongly suggest to upgrade to OFED 1.2, which supports Up*/Down* routing engine and LMC mechanism correctly.

9.4.6 Failure with Automatic Path Migration

MVAPICH2 (OFA-IB-CH3) provides network fault tolerance with Automatic Path Migration (APM). However, APM is supported only with OFED 1.2 onwards. With OFED 1.1 and prior versions of OpenFabrics drivers, APM functionality is not completely supported. Please refer to Section 11.73 and section 11.74

9.4.7 Error opening file

If you configure MVAPICH2 with `RDMA_CM` and see this error, you need to verify if you have setup up the local IP address to be used by RDMA_CM in the file `/etc/mv2.conf`. Further, you need to make sure that this file has the appropriate file read permissions. Please follow Section 6.15 for more details on this.

9.4.8 RDMA CM Address error

If you get this error, please verify that the IP address specified `/etc/mv2.conf` is correctly specified with the IP address of the device you plan to use RDMA_CM with.

9.4.9 RDMA CM Route error

If see this error, you need to check whether the specified network is working or not.

9.5 With OFA-iWARP-CH3 Interface

9.5.1 Error opening file

If you configure MVAPICH2 with RDMA_CM and see this error, you need to verify if you have setup up the local IP address to be used by RDMA_CM in the file `/etc/mv2.conf`. Further, you need to make sure that this file has the appropriate file read permissions. Please follow Section 5.2.6 for more details on this.

9.5.2 RDMA CM Address error

If you get this error, please verify that the IP address specified `/etc/mv2.conf` is correctly specified with the IP address of the device you plan to use RDMA_CM with.

9.5.3 RDMA CM Route error

If see this error, you need to check whether the specified network is working or not.

9.6 Checkpoint/Restart

9.6.1 Failure during Restart

Please make sure the following things for a successful restart:

- The BLCR modules must be loaded on all the compute nodes and the console node before a restart
- The checkpoint file of MPI job console must be accessible from the console node.
- The corresponding checkpoint files of the MPI processes must be accessible from the compute nodes using the same path as when checkpoint was taken.

The following things can cause a restart to fail:

- The job which was checkpointed is not terminated or the some processes in that job are not cleaned properly. Usually they will be cleaned automatically, otherwise, since the pid can't be used by BLCR to restart, it will fail.
- The processes in the job have opened temporary files and these temporary files are removed or not accessible from the nodes where the processes are restarted on.
- If the processes are restarted on different nodes, then all the nodes must have the exact same libraries installed. In particular, you may be required to disable any "prelinking". Please look at https://upc-bugs.lbl.gov//blcr/doc/html/FAQ.html#prelink for further details.

FAQ regarding Berkeley Lab Checkpoint/Restart (BLCR) can be found at:
http://upc-bugs.lbl.gov/blcr/doc/html/FAQ.html And the user guide for BLCR can be found at http://upc-bugs.lbl.gov/blcr/doc/html/BLCR_Users_Guide.html

If you encounter any problem with the Checkpoint/Restart support, please feel free to contact us at mvapich-discuss@cse.ohio-state.edu.

9.6.2 Errors related to SHArP with multiple concurrent jobs

There is a large, yet finite number of jobs that can run in parallel which takes advantage of SHArP. If this limit is exceeded, one may observe errors like `ERROR sharp_get_job_data_len failed: Job not found`. This is a system limitation.

10 MVAPICH2 General Parameters

10.1 MV2_IGNORE_SYSTEM_CONFIG

- Class: Run time
- Default: 0

If set, the system configuration file is not processed.

10.2 MV2_IGNORE_USER_CONFIG

- Class: Run time
- Default: 0

If set, the user configuration file is not processed.

10.3 MV2_USER_CONFIG

- Class: Run time
- Default: Unset

Specify the path of a user configuration file for mvapich2. If this is not set the default path of " /.mvapich2.conf" is used.

10.4 MV2_DEBUG_CORESIZE

- Class: Run time
- Default: Unset
- Possible values: Positive integer or "unlimited"

Set the limit for the core size resource. It allows to specify the maximum size for a core dump to be generated. It only set the soft limit and it has the respect the hard value set on the nodes.

It is similar to the `ulimit -c <coresize>` that can be run in the shell, but this will only apply to the MVAPICH2 processes (MPI processes, mpirun_rsh, mpispawn).

Examples:

- '`MV2_DEBUG_CORESIZE=0`' will disable core dumps for MVAPICH2 processes.
- '`MV2_DEBUG_CORESIZE=unlimited`' will enable core dumps for MVAPICH2 processes.

10.5 MV2_DEBUG_SHOW_BACKTRACE

- Class: Run time
- Default: 0 (disabled)
- Possible values: 1 to enable, 0 to disable

Show a backtrace when a process fails on errors like "Segmentation faults", "Bus error", "Illegal Instruction", "Abort" or "Floating point exception".

If your application uses the static version of the MVAPICH2 library, you have to link your application with the `-rdynamic` flag in order to see the function names in the backtrace. For more information, see the backtrace manpage.

10.6 MV2_SHOW_ENV_INFO

- Class: Run time
- Default: 0 (disabled)
- Possible values: 1 (short list), 2(full list)

Show the values assigned to the run time environment parameters

10.7 MV2_SHOW_CPU_BINDING

- Class: Run time
- Default: 0 (disabled)
- Possible values: 1 (Show only on node containing rank 0), 2 (Show on all nodes)

If set to 1, it shows the CPU mapping of all processes on node where rank 0 exists. If set to 2, it shows the CPU mapping of all processes on all nodes.

11 MVAPICH2 Parameters (CH3-Based Interfaces)

11.1 MV2_ALLREDUCE_2LEVEL_MSG

- Class: Run Time
- Default: 256K Bytes
- Applicable interface(s): OFA-IB-CH3 and OFA-iWARP-CH3

This parameter can be used to determine the threshold for the 2-level Allreduce algorithm. We now use the shared-memory-based algorithm for messages smaller than the MV2_SHMEM_ALLREDUCE_MSG threshold (11.58), the 2-level algorithm for medium sized messages up to the threshold defined by this parameter. We use the default point-to-point algorithms messages larger than this threshold.

11.2 MV2_CKPT_AGGREGATION_BUFPOOL_SIZE

- Class: Run Time
- Default: 8M
- Applicable interface(s): OFA-IB-CH3

This parameter determines the size of the buffer pool reserved for use in checkpoint aggregation. Note that this variable can be set with suffixes such as 'K'/'k', 'M'/'m' or 'G'/'g' to denote Kilobyte, Megabyte or Gigabyte respectively.

11.3 MV2_CKPT_AGGREGATION_CHUNK_SIZE

- Class: Run Time
- Default: 1M
- Applicable interface(s): OFA-IB-CH3

The checkpoint data that has been coalesced into the buffer pool, is written to the back-end file system, with the value of this parameter as the chunk size. Note that this variable can be set with suffixes such as 'K'/'k', 'M'/'m' or 'G'/'g' to denote Kilobyte, Megabyte or Gigabyte respectively.

11.4 MV2_CKPT_FILE

- Class: Run Time
- Default: /tmp/ckpt

- Applicable interface(s): OFA-IB-CH3

This parameter specifies the path and the base file name for checkpoint files of MPI processes. The checkpoint files will be named as $MV2_CKPT_FILE.<number of checkpoint>.<process rank>, for example, /tmp/ckpt.1.0 is the checkpoint file for process 0's first checkpoint. To checkpoint on network-based file systems, user just need to specify the path to it, such as /mnt/pvfs2/my_ckpt_file.

11.5 MV2_CKPT_INTERVAL

- Class: Run Time
- Default: 0
- Unit: minutes
- Applicable interface(s): OFA-IB-CH3

This parameter can be used to enable automatic checkpointing. To let MPI job console automatically take checkpoints, this value needs to be set to the desired checkpointing interval. A zero will disable automatic checkpointing. Using automatic checkpointing, the checkpoint file for the MPI job console will be named as $MV2_CKPT_FILE.<number of checkpoint>.auto. Users need to use this file for restart.

11.6 MV2_CKPT_MAX_SAVE_CKPTS

- Class: Run Time
- Default: 0
- Applicable interface(s): OFA-IB-CH3

This parameter is used to limit the number of checkpoints saved on file system to save the file system space. When set to a positive value N, only the last N checkpoints will be saved.

11.7 MV2_CKPT_NO_SYNC

- Class: Run Time
- Applicable interface(s): OFA-IB-CH3

When this parameter is set to any value, the checkpoints will not be required to sync to disk. It can reduce the checkpointing delay in many cases. But if users are using local file system, or any parallel file system with local cache, to store the checkpoints, it is recommended not to set this parameter because otherwise the checkpoint files will be cached in local memory and will likely be lost upon failure.

11.8 MV2_CKPT_USE_AGGREGATION

- Class: Run Time
- Default: 1 (if configured with Checkpoint Aggregation support)
- Applicable interface(s): OFA-IB-CH3

This parameter enables/disables Checkpoint aggregation scheme at run time. It is set to '1'(enabled) by default, when the user enables Checkpoint/Restart functionality at configure time, or when the user explicitly configures MVAPICH2 with aggregation support. Please note that, to use aggregation support, each node needs to be properly configured with FUSE library (cf section 6.14.1).

11.9 MV2_DEBUG_FT_VERBOSE

- Class: Run Time
- Type: Null or positive integer
- Default: 0 (disabled)

This parameter enables/disables the debug output for Fault Tolerance features (Checkpoint/Restart and Migration).

Note: All debug output is disabled when MVAPICH2 is configured with the `--enable-fast=ndebug` option.

11.10 MV2_CM_RECV_BUFFERS

- Class: Run Time
- Default: 1024
- Applicable interface(s): OFA-IB-CH3

This defines the number of buffers used by connection manager to establish new connections. These buffers are quite small and are shared for all connections, so this value may be increased to 8192 for large clusters to avoid retries in case of packet drops.

11.11 MV2_CM_SPIN_COUNT

- Class: Run Time
- Default: 5000
- Applicable interface(s): OFA-IB-CH3

This is the number of the connection manager polls for new control messages from UD channel for each interrupt. This may be increased to reduce the interrupt overhead when many incoming control messages from UD channel at the same time.

11.12 MV2_CM_TIMEOUT

- Class: Run Time
- Default: 500
- Unit: milliseconds
- Applicable interface(s): OFA-IB-CH3

This is the timeout value associated with connection management messages via UD channel. Decreasing this value may lead to faster retries but at the cost of generating duplicate messages.

11.13 MV2_CPU_MAPPING

- Class: Run Time
- Default: NA
- Applicable interface(s): OFA-IB-CH3, OFA-iWARP-CH3, PSM

This allows users to specify process to CPU (core) mapping. The detailed usage of this parameter is described in Section 6.5.3. This parameter will not take effect if either *MV2_ENABLE_AFFINITY* or *MV2_USE_SHARED_MEM* run-time parameters are set to 0, or if the library was configured with the "–disable-hwloc" option. MV2_CPU_MAPPING is currently not supported on Solaris.

11.14 MV2_CPU_BINDING_POLICY

- Class: Run Time
- Default: hybrid
- Applicable interface(s): OFA-IB-CH3, OFA-iWARP-CH3, PSM

We have changed the default value of MV2_CPU_BINDING_POLICY to "hybrid" along with MV2_HYBRID_BINDING_POLICY=bunch. It is same as setting MV2_CPU_BINDING_POLICY to bunch. However, it also works well for the systems with hyper-threading enabled or systems that have vendor specific core mappings. This allows users to specify process to CPU (core) mapping with the CPU binding policy. The detailed usage of this parameter is described in Section 6.5.1. This parameter will not take effect: if
MV2_ENABLE_AFFINITY or *MV2_USE_SHARED_MEM* run-time parameters are set to 0; or

MV2_ENABLE_AFFINITY is set to 1 and *MV2_CPU_MAPPING* is set, or if the library was configured with the "–disable-hwloc" option. The value of MV2_CPU_BINDING_POLICY can be "bunch", "scatter", or "hybrid". When this parameter takes effect and its value isn't set, "bunch" will be used as the default policy.

11.15 MV2_HYBRID_BINDING_POLICY

- Class: Run Time
- Default: Linear
- Applicable interface(s): OFA-IB-CH3, OFA-iWARP-CH3, PSM

This allows users to specify binding policies for application thread in MPI+Threads applications. The detailed usage of this parameter is described in Section 6.20. This parameter will not take effect: if
MV2_ENABLE_AFFINITY or *MV2_USE_SHARED_MEM* run-time parameters are set to 0; or
MV2_CPU_BINDING_POLICY is set to "bunch" or "scatter", or
MV2_ENABLE_AFFINITY is set to 1 and *MV2_CPU_MAPPING* is set, or if the library was configured with the "–disable-hwloc" option. The value of MV2_HYBRID_BINDING_POLICY can be "linear", "compact", "bunch", "scatter", or "spread". When this parameter takes effect and its value isn't set, "linear" will be used as the default policy.

11.16 MV2_CPU_BINDING_LEVEL

- Class: Run Time
- Default: Core
- Applicable interface(s): OFA-IB-CH3 and OFA-iWARP-CH3

This allows users to specify process to CPU (core) mapping at different binding level. The detailed usage of this parameter is described in Section 6.5.1. This parameter will not take effect: if *MV2_ENABLE_AFFINITY* or *MV2_USE_SHARED_MEM* run-time parameters are set to 0; or `MV2_ENABLE_AFFINITY` is set to 1 and `MV2_CPU_MAPPING` is set, or if the library was configured with the "–disable-hwloc" option. The value of MV2_CPU_BINDING_LEVEL can be "core", "socket", or "numanode". When this parameter takes effect and its value isn't set, "core" will be used as the default binding level.

11.17 MV2_SHOW_HCA_BINDING

- Class: Run time
- Default: 0 (disabled)

- Applicable interface(s): OFA-IB-CH3, OFA-iWARP-CH3
- Possible values: 1 (Show only on node containing rank 0), 2 (Show on all nodes)

If set to 1, it shows the HCA binding of all processes on node where rank 0 exists. If set to 2, it shows the HCA binding of all processes on all nodes.

11.18 MV2_DEFAULT_MAX_SEND_WQE

- Class: Run time
- Default: 64
- Applicable interface(s): OFA-IB-CH3 and OFA-iWARP-CH3

This specifies the maximum number of send WQEs on each QP. Please note that for OFA-IB-CH3 and OFA-iWARP-CH3, the default value of this parameter will be 16 if the number of processes is larger than 256 for better memory scalability.

11.19 MV2_DEFAULT_MAX_RECV_WQE

- Class: Run time
- Default: 128
- Applicable interface(s): OFA-IB-CH3 and OFA-iWARP-CH3

This specifies the maximum number of receive WQEs on each QP (maximum number of receives that can be posted on a single QP).

11.20 MV2_DEFAULT_MTU

- Class: Run time
- Default: OFA-IB-CH3: IBV_MTU_1024 for IB SDR cards and IBV_MTU_2048 for IB DDR and QDR cards.
- Applicable interface(s): OFA-IB-CH3

The internal MTU size. For OFA-IB-CH3, this parameter should be a string instead of an integer. Valid values are: `IBV_MTU_256`, `IBV_MTU_512`, `IBV_MTU_1024`, `IBV_MTU_2048`, `IBV_MTU_4096`.

11.21 MV2_DEFAULT_PKEY

- Class: Run Time
- Applicable Interface(s): OFA-IB-CH3

Select the partition to be used for the job.

11.22 MV2_ENABLE_AFFINITY

- Class: Run time
- Default: 1
- Applicable interface(s): OFA-IB-CH3 and OFA-iWARP-CH3

Enable CPU affinity by setting MV2_ENABLE_AFFINITY to 1 or disable it by setting MV2_ENABLE_AFFINITY to 0. MV2_ENABLE_AFFINITY is currently not supported on Solaris. CPU affinity is also not supported if MV2_USE_SHARED_MEM is set to 0.

11.23 MV2_GET_FALLBACK_THRESHOLD

- Class: Run time
- This threshold value needs to be set in bytes.
- This option is effective if we define ONE_SIDED flag.
- Applicable interface(s): OFA-IB-CH3 and OFA-iWARP-CH3

This defines the threshold beyond which the MPI_Get implementation is based on direct one sided RDMA operations.

11.24 MV2_IBA_EAGER_THRESHOLD

- Class: Run time
- Default: Host Channel Adapter (HCA) dependent (12 KB for ConnectX HCA's)
- Applicable interface(s): OFA-IB-CH3 and OFA-iWARP-CH3

This specifies the switch point between eager and rendezvous protocol in MVAPICH2. For better performance, the value of MV2_IBA_EAGER_THRESHOLD should be set the same as MV2_VBUF_TOTAL_SIZE.

11.25 MV2_IBA_HCA

- Class: Run time
- Default: Unset
- Applicable interface(s): OFA-IB-CH3 and OFA-iWARP-CH3

This specifies the HCA's to be used for performing network operations.

11.26 MV2_INITIAL_PREPOST_DEPTH

- Class: Run time
- Default: 10
- Applicable interface(s): OFA-IB-CH3 and OFA-iWARP-CH3

This defines the initial number of pre-posted receive buffers for each connection. If communication happen for a particular connection, the number of buffers will be increased to RDMA_PREPOST_DEPTH.

11.27 MV2_IWARP_MULTIPLE_CQ_THRESHOLD

- Class: Run time
- Default: 32
- Applicable interface(s): OFA-iWARP-CH3

This defines the process size beyond which we use multiple completion queues for iWARP interface.

11.28 MV2_KNOMIAL_INTRA_NODE_FACTOR

- Class: Run time
- Default: 4
- Applicable interface(s): OFA-IB-CH3 and OFA-iWARP-CH3

This defines the degree of the knomial operation during the intra-node knomial broadcast phase.

11.29 MV2_KNOMIAL_INTER_NODE_FACTOR

- Class: Run time
- Default: 4
- Applicable interface(s): OFA-IB-CH3 and OFA-iWARP-CH3

This defines the degree of the knomial operation during the inter-node knomial broadcast phase.

11.30 MV2_MAX_INLINE_SIZE

- Class: Run time
- Default: Network card dependent (128 for most networks including InfiniBand)
- Applicable interface(s): OFA-IB-CH3, OFA-iWARP-CH3

This defines the maximum inline size for data transfer. Please note that the default value of this parameter will be 0 when the number of processes is larger than 256 to improve memory usage scalability.

11.31 MV2_MAX_NUM_WIN

- Class: Run time
- Default: 64
- Applicable interface(s): OFA-IB-CH3

Maximum number of RMA windows that can be created and active concurrently. Typically this value is sufficient for most applications. Increase this value to the number of windows your application uses

11.32 MV2_NDREG_ENTRIES

- Class: Run time
- Default: 1000
- Applicable interface(s): OFA-IB-CH3 and OFA-iWARP-CH3

This defines the total number of buffers that can be stored in the registration cache. It has no effect if MV2_USE_LAZY_MEM_UNREGISTER is not set. A larger value will lead to less frequent lazy de-registration.

11.33 MV2_NUM_HCAS

- Class: Run time
- Default: 1
- Applicable interface(s): OFA-IB-CH3, OFA-iWARP-CH3

This parameter indicates number of InfiniBand adapters to be used for communication on an end node.

11.34 MV2_NUM_PORTS

- Class: Run time
- Default: 1
- Applicable interface(s): OFA-IB-CH3, OFA-iWARP-CH3

This parameter indicates number of ports per InfiniBand adapter to be used for communication per adapter on an end node.

11.35 MV2_DEFAULT_PORT

- Class: Run time
- Default: none
- Applicable interface(s): OFA-IB-CH3, OFA-iWARP-CH3

This parameter is to select the specific HCA port on a active multi port InfiniBand adapter

11.36 MV2_NUM_SA_QUERY_RETRIES

- Class: Run time
- Default: 20
- Applicable Interface(s): OFA-IB-CH3, OFA-iWARP-CH3

Number of times the MPI library will attempt to query the subnet to obtain the path record information before giving up.

11.37 MV2_NUM_QP_PER_PORT

- Class: Run time
- Default: 1
- Applicable interface(s): OFA-IB-CH3, OFA-iWARP-CH3

This parameter indicates number of queue pairs per port to be used for communication on an end node. This is useful in the presence of multiple send/recv engines available per port for data transfer.

11.38 MV2_RAIL_SHARING_POLICY

- Class: Run time
- Default: Rail Binding in round-robin
- Value Domain: USE_FIRST, ROUND_ROBIN, FIXED_MAPPING
- Applicable interface(s): OFA-IB-CH3, OFA-iWARP-CH3

This specifies the policy that will be used to assign HCAs to each of the processes. In the previous versions of MVAPICH2 it was known as MV2_SM_SCHEDULING.

11.39 MV2_RAIL_SHARING_LARGE_MSG_THRESHOLD

- Class: Run time
- Default: 16K
- Applicable interface(s): OFA-IB-CH3, OFA-iWARP-CH3

This specifies the threshold for the message size beyond which striping will take place. In the previous versions of MVAPICH2 it was known as MV2_STRIPING_THRESHOLD

11.40 MV2_PROCESS_TO_RAIL_MAPPING

- Class: Run time
- Default: NONE
- Value Domain: BUNCH, SCATTER, <CUSTOM LIST>
- Applicable interface(s): OFA-IB-CH3, OFA-iWARP-CH3

When MV2_RAIL_SHARING_POLICY is set to the value "FIXED_MAPPING" this variable decides the manner in which the HCAs will be mapped to the rails. The <CUSTOM LIST> is colon(:) separated list with the HCA ranks specified. e.g. 0:1:1:0. This list must map equally to the number of local processes on the nodes failing which, the default policy will be used. Similarly the number of processes on each node must be the same. The detailed usage of this parameter is described in Section 6.12.

11.41 MV2_RDMA_FAST_PATH_BUF_SIZE

- Class: Run time
- Default: Architecture dependent
- Applicable interface(s): OFA-IB-CH3 and OFA-iWARP-CH3

The size of the buffer used in RDMA fast path communication. This value will be ineffective if MV2_USE_RDMA_FAST_PATH is not set

11.42 MV2_NUM_RDMA_BUFFER

- Class: Run time
- Default: Architecture dependent (32 for EM64T)
- Applicable interface(s): OFA-IB-CH3 and OFA-iWARP-CH3

The number of RDMA buffers used for the RDMA fast path. This *fast path* is used to reduce latency and overhead of small data and control messages. This value will be ineffective if MV2_USE_RDMA_FAST_PATH is not set.

11.43 MV2_ON_DEMAND_THRESHOLD

- Class: Run Time
- Default: 64 (OFA-IB-CH3), 16 (OFA-iWARP-CH3)
- Applicable interface(s): OFA-IB-CH3 and OFA-iWARP-CH3

This defines threshold for enabling on-demand connection management scheme. When the size of the job is larger than the threshold value, on-demand connection management will be used.

11.44 MV2_HOMOGENEOUS_CLUSTER

- Class: Run Time
- Default: 0
- Applicable interface(s): OFA-IB-CH3 and OFA-iWARP-CH3

Set this parameter to 1 on homogeneous clusters to optimize the job start-up

11.45 MV2_PREPOST_DEPTH

- Class: Run time
- Default: 64
- Applicable interface(s): OFA-IB-CH3 and OFA-iWARP-CH3

This defines the number of buffers pre-posted for each connection to handle send/receive operations.

11.46 MV2_PSM_DEBUG

- Class: Run time (Debug)
- Default: 0
- Applicable interface: PSM

This parameter enables the dumping of run-time debug counters from the MVAPICH2-PSM progress engine. Counters are dumped every PSM_DUMP_FREQUENCY seconds.

11.47 MV2_PSM_DUMP_FREQUENCY

- Class: Run time (Debug)
- Default: 10 seconds
- Applicable interface: PSM

This parameters sets the frequency for dumping MVAPICH2-PSM debug counters. Value takes effect only in PSM_DEBUG is enabled.

11.48 MV2_PUT_FALLBACK_THRESHOLD

- Class: Run time
- This threshold value needs to be set in bytes.
- This option is effective if we define ONE_SIDED flag.
- Applicable interface(s): OFA-IB-CH3 and OFA-iWARP-CH3

This defines the threshold beyond which the MPI_Put implementation is based on direct one sided RDMA operations.

11.49 MV2_RAIL_SHARING_LARGE_MSG_THRESHOLD

- Class: Run Time
- Default: 16 KB
- Applicable interface(s): OFA-IB-CH3, OFA-iWARP-CH3

This parameter specifies the message size above which we begin the stripe the message across multiple rails (if present).

11.50 MV2_RAIL_SHARING_POLICY

- Class: Run Time
- Default: PROCESS_BINDING (Options: USE_FIRST, ROUND_ROBIN, FIXED_MAPPING)
- Applicable interface(s): OFA-IB-CH3, OFA-iWARP-CH3

This parameter specifies the rail sharing policy for all message sizes if multiple rails are present.

11.51 MV2_RDMA_CM_ARP_TIMEOUT

- Class: Run Time
- Default: 2000 ms
- Applicable interface(s): OFA-IB-CH3, OFA-iWARP-CH3, OFA-RoCE-CH3

This parameter specifies the ARP timeout to be used by RDMA CM module.

11.52 MV2_RDMA_CM_MAX_PORT

- Class: Run Time
- Default: Unset
- Applicable interface(s): OFA-IB-CH3, OFA-iWARP-CH3, OFA-RoCE-CH3

This parameter specifies the upper limit of the port range to be used by the RDMA CM module when choosing the port on which it listens for connections.

11.53 MV2_RDMA_CM_MIN_PORT

- Class: Run Time
- Default: Unset
- Applicable interface(s): OFA-IB-CH3, OFA-iWARP-CH3, OFA-RoCE-CH3

This parameter specifies the lower limit of the port range to be used by the RDMA CM module when choosing the port on which it listens for connections.

11.54 MV2_REDUCE_2LEVEL_MSG

- Class: Run Time
- Default: 32K Bytes.
- Applicable interface(s): OFA-IB-CH3 and OFA-iWARP-CH3

This parameter can be used to determine the threshold for the 2-level reduce algorithm. We now use the shared-memory-based algorithm for messages smaller than the MV2_SHMEM_REDUCE_MSG (11.64), the 2-level algorithm for medium sized messages up to the threshold defined by this parameter. We use the default point-to-point algorithms messages larger than this threshold.

11.55 MV2_RNDV_PROTOCOL

- Class: Run time
- Default: RPUT
- Applicable interface(s): OFA-IB-CH3, OFA-iWARP-CH3

The value of this variable can be set to choose different Rendezvous protocols. RPUT (default RDMA-Write) RGET (RDMA Read based), R3 (send/recv based).

11.56 MV2_R3_THRESHOLD

- Class: Run time
- Default: MV2_IBA_EAGER_THRESHOLD
- Applicable interface(s): OFA-IB-CH3, OFA-iWARP-CH3

The value of this variable controls what message sizes go over the R3 rendezvous protocol. Messages above this message size use MV2_RNDV_PROTOCOL.

11.57 MV2_R3_NOCACHE_THRESHOLD

- Class: Run time
- Default: 32768
- Applicable interface(s): OFA-IB-CH3, OFA-iWARP-CH3

The value of this variable controls what message sizes go over the R3 rendezvous protocol when the registration cache is disabled (MV2_USE_LAZY_MEM_UNREGISTER=0). Messages above this message size use MV2_RNDV_PROTOCOL.

11.58 MV2_SHMEM_ALLREDUCE_MSG

- Class: Run Time
- Default: $1 \ll 15$
- Applicable interface(s): OFA-IB-CH3, OFA-iWARP-CH3

The SHMEM AllReduce is used for messages less than this threshold.

11.59 MV2_SHMEM_BCAST_LEADERS

- Class: Run time
- Default: 4096

The number of leader processes that will take part in the SHMEM broadcast operation. Must be greater than the number of nodes in the job.

11.60 MV2_SHMEM_BCAST_MSG

- Class: Run Time
- Default: $1 \ll 20$
- Applicable interface(s): OFA-IB-CH3, OFA-iWARP-CH3

The SHMEM bcast is used for messages less than this threshold.

11.61 MV2_SHMEM_COLL_MAX_MSG_SIZE

- Class: Run Time
- Applicable interface(s): OFA-IB-CH3, OFA-iWARP-CH3

This parameter can be used to select the max buffer size of message for shared memory collectives.

11.62 MV2_SHMEM_COLL_NUM_COMM

- Class: Run Time
- Applicable interface(s): OFA-IB-CH3, OFA-iWARP-CH3

This parameter can be used to select the number of communicators using shared memory collectives.

11.63 MV2_SHMEM_DIR

- Class: Run Time
- Applicable interface(s): OFA-IB-CH3 and OFA-iWARP-CH3
- Default: /dev/shm for Linux and /tmp for Solaris

This parameter can be used to specify the path to the shared memory files for intra-node communication.

11.64 MV2_SHMEM_REDUCE_MSG

- Class: Run Time
- Default: $1 \ll 13$
- Applicable interface(s): OFA-IB-CH3, OFA-iWARP-CH3

The SHMEM reduce is used for messages less than this threshold.

11.65 MV2_SM_SCHEDULING

- Class: Run Time
- Default: USE_FIRST (Options: ROUND_ROBIN)
- Applicable interface(s): OFA-IB-CH3, OFA-iWARP-CH3

11.66 MV2_SMP_USE_LIMIC2

- Class: Run Time
- Default: On if configured with –with-limic2
- Applicable interface(s): OFA-IB-CH3 and OFA-iWARP-CH3

This parameter enables/disables LiMIC2 at run time. It does not take effect if MVAPICH2 is not configured with –with-limic2.

11.67 MV2_SMP_USE_CMA

- Class: Run Time
- Default: On unless configured with –without-cma
- Applicable interface(s): OFA-IB-CH3 and OFA-iWARP-CH3

This parameter enables/disables CMA based intra-node communication at run time. It does not take effect if MVAPICH2 is configured with –without-cma. When –with-limic2 is included in the configure flags, LiMIC2 is used in preference over CMA. Please set MV2_SMP_USE_LIMIC2 to 0 in order to choose CMA if MVAPICH2 is configured with –with-limic2.

11.68 MV2_SRQ_LIMIT

- Class: Run Time
- Default: 30
- Applicable interface(s): OFA-IB-CH3, OFA-iWARP-CH3

This is the low water-mark limit for the Shared Receive Queue. If the number of available work entries on the SRQ drops below this limit, the flow control will be activated.

11.69 MV2_SRQ_MAX_SIZE

- Class: Run Time
- Default: 4096
- Applicable interface(s): OFA-IB-CH3, OFA-iWARP-CH3

This is the maximum number of work requests allowed on the Shared Receive Queue. Upon receiving a SRQ limit event, the current value of MV2_SRQ_SIZE will be doubled or moved to the maximum of MV2_SRQ_MAX_SIZE, whichever is smaller

11.70 MV2_SRQ_SIZE

- Class: Run Time
- Default: 256
- Applicable interface(s): OFA-IB-CH3, OFA-iWARP-CH3

This is the initial number of work requests posted to the Shared Receive Queue.

11.71 MV2_STRIPING_THRESHOLD

- Class: Run Time
- Default: 8192
- Applicable interface(s): OFA-IB-CH3, OFA-iWARP-CH3

This parameter specifies the message size above which we begin the stripe the message across multiple rails (if present).

11.72 MV2_SUPPORT_DPM

- Class: Run time
- Default: 0 (disabled)
- Applicable interface: OFA-IB-CH3

This option enables the dynamic process management interface and on-demand connection management.

11.73 MV2_USE_APM

- Class: Run Time
- Applicable interface(s): OFA-IB-CH3

This parameter is used for recovery from network faults using Automatic Path Migration. This functionality is beneficial in the presence of multiple paths in the network, which can be enabled by using lmc mechanism.

11.74 MV2_USE_APM_TEST

- Class: Run Time
- Applicable interface(s): OFA-IB-CH3

This parameter is used for testing the Automatic Path Migration functionality. It periodically moves the alternate path as the primary path of communication and re-loads another alternate path.

11.75 MV2_USE_BLOCKING

- Class: Run time
- Default: 0
- Applicable interface(s): OFA-IB-CH3

Setting this parameter enables MVAPICH2 to use blocking mode progress. MPI applications do not take up any CPU when they are waiting for incoming messages.

11.76 MV2_USE_COALESCE

- Class: Run time
- Default: unset
- Applicable interface(s): OFA-IB-CH3, OFA-iWARP-CH3

Setting this parameter enables message coalescing to increase small message throughput

11.77 MV2_USE_DIRECT_GATHER

- Class: Run time
- Default: set
- Applicable interface(s): OFA-IB-CH3 and OFA-iWARP-CH3

Use the "Direct" algorithm for the MPI_Gather operation. If this parameter is set to 0 at run-time, the "Direct" algorithm will not be invoked.

11.78 MV2_USE_DIRECT_SCATTER

- Class: Run time
- Default: set
- Applicable interface(s): OFA-IB-CH3 and OFA-iWARP-CH3

Use the "Direct" algorithm for the MPI_Scatter operation. If this parameter is set to 0 at run-time, the "Direct" algorithm will not be invoked.

11.79 MV2_USE_HSAM

- Class: Run Time
- Applicable interface(s): OFA-IB-CH3

This parameter is used for utilizing hot-spot avoidance with InfiniBand clusters. To leverage this functionality, the subnet should be configured with lmc greater than zero. Please refer to section 6.21 for detailed information.

11.80 MV2_USE_IWARP_MODE

- Class: Run Time
- Default: unset
- Applicable interface(s): OFA-IB-CH3, OFA-iWARP-CH3

This parameter enables the library to run in iWARP mode.

11.81 MV2_USE_LAZY_MEM_UNREGISTER

- Class: Run time
- Default: set
- Applicable interface(s): OFA-IB-CH3 and OFA-iWARP-CH3

Setting this parameter enables MVAPICH2 to use memory registration cache.

11.82 MV2_USE_RoCE

- Class: Run Time
- Default: Un Set
- Applicable interface(s): OFA-IB-CH3

This parameter enables the use of RDMA over Ethernet for MPI communication. The underlying HCA and network must support this feature.

11.83 MV2_DEFAULT_GID_INDEX

- Class: Run Time
- Default: 0
- Applicable interface(s): OFA-IB-CH3

In RoCE mode, this parameter allows to choose non-default GID index in loss-less ethernet setup using VLANs

11.84 MV2_USE_RDMA_CM

- Class: Run Time
- Default: Network Dependent (set for OFA-iWARP-CH3 and unset for OFA-IB-CH3/OFA-RoCE-CH3)
- Applicable interface(s): OFA-IB-CH3, OFA-iWARP-CH3, OFA-RoCE-CH3

This parameter enables the use of RDMA CM for establishing the connections.

11.85 MV2_RDMA_CM_MULTI_SUBNET_SUPPORT

- Class: Run Time
- Default: Unset
- Applicable interface(s): OFA-IB-CH3, OFA-iWARP-CH3

This parameter allows MPI jobs to be run across multiple subnets interconnected by InfiniBand routers. Note that this requires RDMA_CM support to be enabled at configure time and runtime. Note that, RDMA_CM support is enabled by default at configure time. At runtime, the MV2_USE_RDMA_CM environment variable described in Section 11.84 must be set to 1.

11.86 MV2_RDMA_CM_CONF_FILE_PATH

- Class: Run Time
- Default: Network Dependent (set for OFA-iWARP-CH3 and unset for OFA-IB-CH3/OFA-RoCE-CH3)
- Applicable interface(s): OFA-IB-CH3, OFA-iWARP-CH3, OFA-RoCE-CH3

This parameter is to specify the path to mv2.conf file. If this is not given, then it searches in the default location /etc/mv2.conf

11.87 MV2_USE_RDMA_FAST_PATH

- Class: Run time
- Default: set
- Applicable interface(s): OFA-IB-CH3 and OFA-iWARP-CH3

Setting this parameter enables MVAPICH2 to use adaptive RDMA fast path features for OFA-IB-CH3 interface.

11.88 MV2_USE_RDMA_ONE_SIDED

- Class: Run time
- Default: set
- Applicable interface(s): OFA-IB-CH3 and OFA-iWARP-CH3

Setting this parameter allows MVAPICH2 to use optimized one sided implementation based RDMA operations.

11.89 MV2_USE_RING_STARTUP

- Class: Run time
- Default: set
- Applicable interface(s): OFA-IB-CH3

Setting this parameter enables MVAPICH2 to use ring based start up.

11.90 MV2_USE_SHARED_MEM

- Class: Run time
- Default: set
- Applicable interface(s): OFA-IB-CH3 and OFA-iWARP-CH3

Use shared memory for intra-node communication.

11.91 MV2_USE_SHMEM_ALLREDUCE

- Class: Run Time
- Applicable interface(s): OFA-IB-CH3 and OFA-iWARP-CH3

This parameter can be used to turn off shared memory based MPI_Allreduce for OFA-IB-CH3 over IBA by setting this to 0.

11.92 MV2_USE_SHMEM_BARRIER

- Class: Run Time
- Applicable interface(s): OFA-IB-CH3 and OFA-iWARP-CH3

This parameter can be used to turn off shared memory based MPI_Barrier for OFA-IB-CH3 over IBA by setting this to 0.

11.93 MV2_USE_SHMEM_BCAST

- Class: Run Time
- Applicable interface(s): OFA-IB-CH3 and OFA-iWARP-CH3

This parameter can be used to turn off shared memory based MPI_Bcast for OFA-IB-CH3 over IBA by setting this to 0.

11.94 MV2_USE_SHMEM_COLL

- Class: Run time
- Default: set
- Applicable interface(s): OFA-IB-CH3 and OFA-iWARP-CH3

Use shared memory for collective communication. Set this to 0 for disabling shared memory collectives.

11.95 MV2_USE_SHMEM_REDUCE

- Class: Run Time
- Applicable interface(s): OFA-IB-CH3 and OFA-iWARP-CH3

This parameter can be used to turn off shared memory based MPI_Reduce for OFA-IB-CH3 over IBA by setting this to 0.

11.96 MV2_USE_SRQ

- Class: Run time
- Default: set
- Applicable interface(s): OFA-IB-CH3, OFA-iWARP-CH3

Setting this parameter enables MVAPICH2 to use shared receive queue.

11.97 MV2_GATHER_SWITCH_PT

- Class: Run time
- Default: set
- Applicable interface(s): OFA-IB-CH3 and OFA-iWARP-CH3

We use different algorithms depending on the system size. For small system sizes (up to 386 cores), we use the "2-level" algorithm following by the "Direct" algorithm. For medium system sizes (up to 1k), we use "Binomial" algorithm following by the "Direct" algorithm. Users can set the switching point between algorithms using the run-time parameter MV2_GATHER_SWITCH_PT.

11.98 MV2_SCATTER_SMALL_MSG

- Class: Run time
- Default: set
- Applicable interface(s): OFA-IB-CH3 and OFA-iWARP-CH3

When the system size is lower than 512 cores, we use the "Binomial" algorithm for small message sizes. MV2_SCATTER_SMALL_MSG allows the users to set the threshold for small messages.

11.99 MV2_SCATTER_MEDIUM_MSG

- Class: Run time
- Default: set
- Applicable interface(s): OFA-IB-CH3 and OFA-iWARP-CH3

When the system size is lower than 512 cores, we use the "2-level" algorithm for medium message sizes. MV2_SCATTER_MEDIUM_MSG allows the users to set the threshold for medium messages.

11.100 MV2_USE_TWO_LEVEL_GATHER

- Class: Run time
- Default: set
- Applicable interface(s): OFA-IB-CH3 and OFA-iWARP-CH3

Use the two-level multi-core-aware algorithm for the MPI_Gather operation. If this parameter is set to 0 at run-time, the two-level algorithm will not be invoked.

11.101 MV2_USE_TWO_LEVEL_SCATTER

- Class: Run time
- Default: set
- Applicable interface(s): OFA-IB-CH3 and OFA-iWARP-CH3

Use the two-level multi-core-aware algorithm for the MPI_Scatter operation. If this parameter is set to 0 at run-time, the two-level algorithm will not be invoked.

11.102 MV2_USE_XRC

- Class: Run time
- Default: 0
- Applicable Interface(s): OFA-IB-CH3

Use the XRC InfiniBand transport available since Mellanox ConnectX adapters. This features requires OFED version later than 1.3. It also automatically enables SRQ and ON-DEMAND connection management. Note that the MVAPICH2 library needs to have been configured with –enable-xrc=yes to use this feature.

11.103 MV2_VBUF_POOL_SIZE

- Class: Run time
- Default: 512
- Applicable interface(s): OFA-IB-CH3, OFA-iWARP-CH3

The number of vbufs in the initial pool. This pool is shared among all the connections.

11.104 MV2_VBUF_SECONDARY_POOL_SIZE

- Class: Run time
- Default: 256
- Applicable interface(s): OFA-IB-CH3, OFA-iWARP-CH3

The number of vbufs allocated each time when the global pool is running out in the initial pool. This is also shared among all the connections.

11.105 MV2_VBUF_TOTAL_SIZE

- Class: Run time
- Default: Host Channel Adapter (HCA) dependent (12 KB for ConnectX HCA's)
- Applicable interface(s): OFA-IB-CH3, OFA-iWARP-CH3

The size of each `vbuf`, the basic communication buffer of MVAPICH2. For better performance, the value of MV2_IBA_EAGER_THRESHOLD should be set the same as MV2_VBUF_TOTAL_SIZE.

11.106 MV2_SMP_EAGERSIZE

- Class: Run time
- Default: Architecture dependent
- Applicable interface(s): OFA-IB-CH3 and OFA-iWARP-CH3

This parameter defines the switch point from Eager protocol to Rendezvous protocol for intra-node communication. Note that this variable can be set with suffixes such as 'K'/'k', 'M'/'m' or 'G'/'g' to denote Kilobyte, Megabyte or Gigabyte respectively.

11.107 MV2_SMPI_LENGTH_QUEUE

- Class: Run time
- Default: Architecture dependent
- Applicable interface(s): OFA-IB-CH3 and OFA-iWARP-CH3

This parameter defines the size of shared buffer between every two processes on the same node for transferring messages smaller than or equal to MV2_SMP_EAGERSIZE. Note that this variable can be set with suffixes such as 'K'/'k', 'M'/'m' or 'G'/'g' to denote Kilobyte, Megabyte or Gigabyte respectively.

11.108 MV2_SMP_NUM_SEND_BUFFER

- Class: Run time
- Default: Architecture dependent
- Applicable interface(s): OFA-IB-CH3 and OFA-iWARP-CH3

This parameter defines the number of internal send buffers for sending intra-node messages larger than MV2_SMP_EAGERSIZE.

11.109 MV2_SMP_SEND_BUF_SIZE

- Class: Run time
- Default: Architecture dependent
- Applicable interface(s): OFA-IB-CH3 and OFA-iWARP-CH3

This parameter defines the packet size when sending intra-node messages larger than MV2_SMP_EAGERSIZE.

11.110 MV2_USE_HUGEPAGES

- Class: Run time
- Default: 1
- Applicable interface(s): OFA-IB-CH3

Set this to 0, to not use any HugePages.

11.111 MV2_HYBRID_ENABLE_THRESHOLD

- Class: Run time
- Default: 1024
- Applicable interface(s): OFA-IB-CH3

This defines the threshold for enabling Hybrid communication using UD and RC/XRC. When the size of the job is greater than or equal to the threshold value, Hybrid mode will be enabled. Otherwise, it uses default RC/XRC connections for communication.

11.112 MV2_HYBRID_MAX_RC_CONN

- Class: Run time
- Default: 64
- Applicable interface(s): OFA-IB-CH3

Maximum number of RC or XRC connections created per process. This limits the amount of connection memory and prevents HCA QP cache thrashing.

11.113 MV2_UD_PROGRESS_TIMEOUT

- Class: Run time
- Default: System size dependent.
- Applicable interface(s): OFA-IB-CH3

Time (usec) until ACK status is checked (and ACKs are sent if needed). To avoid unnecessary retries, set this value less than MV2_UD_RETRY_TIMEOUT. It is recommended to set this to 1/10 of
MV2_UD_RETRY_TIMEOUT.

11.114 MV2_UD_RETRY_TIMEOUT

- Class: Run time
- Default: System size dependent.
- Applicable interface(s): OFA-IB-CH3

Time (usec) after which an unacknowledged message will be retried

11.115 MV2_UD_RETRY_COUNT

- Class: Run time
- Default: System size dependent.
- Applicable interface(s): OFA-IB-CH3

Number of retries of a message before the job is aborted. This is needed in case of HCA fails.

11.116 MV2_USE_UD_HYBRID

- Class: Run time
- Default: 1
- Applicable interface(s): OFA-IB-CH3

Set this to Zero, to disable UD transport in hybrid configuration mode.

11.117 MV2_USE_ONLY_UD

- Class: Run time
- Default: 0
- Applicable interface(s): OFA-IB-CH3

Set this to 1, to enable only UD transport in hybrid configuration mode. It will not use any RC/XRC connections in this mode.

11.118 MV2_USE_UD_ZCOPY

- Class: Run time
- Default: 1
- Applicable interface(s): OFA-IB-CH3

Whether or not to use the zero-copy transfer mechanism to transfer large messages on UD transport.

11.119 MV2_USE_LIMIC_GATHER

- Class: Run time
- Default: 0
- Applicable interface(s): OFA-IB-CH3, PSM

If this flag is set to 1, we will use intra-node Zero-Copy MPI_Gather designs, when the library has been configured to use LiMIC2.

11.120 MV2_USE_MCAST

- Class: Run time
- Default: 0
- Applicable interface(s): OFA-IB-CH3

Set this to 1, to enable hardware multicast support in collective communication

11.121 MV2_MCAST_NUM_NODES_THRESHOLD

- Class: Run time
- Default: 8
- Applicable interface(s): OFA-IB-CH3

This defines the threshold for enabling multicast support in collective communication. When MV2_USE_MCAST is set to 1 and the number of nodes in the job is greater than or equal to the threshold value, it uses multicast support in collective communication

11.122 MV2_USE_CUDA

- Class: Run time
- Default: 0
- Applicable interface(s): OFA-IB-CH3

set this to One. to enable support for communication with GPU device buffers.

11.123 MV2_CUDA_BLOCK_SIZE

- Class: Run time
- Default: 262144
- Applicable interface(s): OFA-IB-CH3

The chunk size used in large message transfer from device memory to host memory. The other suggested values for this parameter are 131072 and 524288.

11.124 MV2_CUDA_KERNEL_VECTOR_TIDBLK_SIZE

- Class: Run time
- Default: 1024
- Applicable interface(s): OFA-IB-CH3

This controls the number of CUDA threads per block in pack/unpack kernels for MPI vector datatype in communication involving GPU device buffers.

11.125 MV2_CUDA_KERNEL_VECTOR_YSIZE

- Class: Run time
- Default: tuned based on dimensions of the vector
- Applicable interface(s): OFA-IB-CH3

This controls the y-dimension of a thread block in pack/unpack kernels for MPI vector datatype in communication involving GPU device buffers. It controls the number of threads operating on each block of data in a vector.

11.126 MV2_CUDA_NONBLOCKING_STREAMS

- Class: Run time
- Default: 1 (Enabled)
- Applicable interface(s): OFA-IB-CH3

This controls the use of non-blocking streams for asynchronous CUDA memory copies in communication involving GPU memory.

11.127 MV2_CUDA_IPC

- Class: Run time
- Default: 1
- Applicable interface(s): OFA-IB-CH3

This enables intra-node GPU-GPU communication using IPC feature available from CUDA 4.1

11.128 MV2_CUDA_SMP_IPC

- Class: Run time
- Default: 0
- Applicable interface(s): OFA-IB-CH3

This enables an optimization for short message GPU device-to-device communication using IPC feature available from CUDA 4.1

11.129 MV2_ENABLE_SHARP

- Class: Run time
- Default: 0
- Applicable interface(s): OFA-IB-CH3

Set this to 1, to enable hardware SHArP support in collective communication

11.130 MV2_SHARP_HCA_NAME

- Class: Run time
- Default: unset
- Applicable interface(s): OFA-IB-CH3

By default, this is set by the MVAPICH2 library. However, you can explicitly set the HCA name which is realized by the SHArP library.

11.131 MV2_SHARP_PORT

- Class: Run time
- Default: 1
- Applicable interface(s): OFA-IB-CH3

By default, this is set by the MVAPICH2 library. However, you can explicitly set the HCA port which is realized by the SHArP library.

12 MVAPICH2 Parameters (OFA-IB-Nemesis Interface)

12.1 MV2_DEFAULT_MAX_SEND_WQE

- Class: Run time
- Default: 64

This specifies the maximum number of send WQEs on each QP. Please note that for Gen2 and Gen2-iWARP, the default value of this parameter will be 16 if the number of processes is larger than 256 for better memory scalability.

12.2 MV2_DEFAULT_MAX_RECV_WQE

- Class: Run time
- Default: 128

This specifies the maximum number of receive WQEs on each QP (maximum number of receives that can be posted on a single QP).

12.3 MV2_DEFAULT_MTU

- Class: Run time
- Default: IBV_MTU_1024 for IB SDR cards and IBV_MTU_2048 for IB DDR and QDR cards.

The internal MTU size. For Gen2, this parameter should be a string instead of an integer. Valid values are: `IBV_MTU_256`, `IBV_MTU_512`, `IBV_MTU_1024`, `IBV_MTU_2048`, `IBV_MTU_4096`.

12.4 MV2_DEFAULT_PKEY

- Class: Run Time

Select the partition to be used for the job.

12.5 MV2_IBA_EAGER_THRESHOLD

- Class: Run time
- Default: Architecture dependent (12KB for IA-32)

This specifies the switch point between eager and rendezvous protocol in MVAPICH2. For better performance, the value of MV2_IBA_EAGER_THRESHOLD should be set the same as MV2_VBUF_TOTAL_SIZE.

12.6 MV2_IBA_HCA

- Class: Run time
- Default: Unset

This specifies the HCA to be used for performing network operations.

12.7 MV2_INITIAL_PREPOST_DEPTH

- Class: Run time
- Default: 10

This defines the initial number of pre-posted receive buffers for each connection. If communication happen for a particular connection, the number of buffers will be increased to RDMA_PREPOST_DEPTH.

12.8 MV2_MAX_INLINE_SIZE

- Class: Run time
- Default: Network card dependent (128 for most networks including InfiniBand)

This defines the maximum inline size for data transfer. Please note that the default value of this parameter will be 0 when the number of processes is larger than 256 to improve memory usage scalability.

12.9 MV2_NDREG_ENTRIES

- Class: Run time
- Default: 1000

This defines the total number of buffers that can be stored in the registration cache. It has no effect if MV2_USE_LAZY_MEM_UNREGISTER is not set. A larger value will lead to less frequent lazy de-registration.

12.10 MV2_NUM_RDMA_BUFFER

- Class: Run time
- Default: Architecture dependent (32 for EM64T)

The number of RDMA buffers used for the RDMA fast path. This *fast path* is used to reduce latency and overhead of small data and control messages. This value will be ineffective if MV2_USE_RDMA_FAST_PATH is not set.

12.11 MV2_NUM_SA_QUERY_RETRIES

- Class: Run time
- Default: 20
- Applicable Interface(s): OFA-IB-CH3, OFA-iWARP-CH3

Number of times the MPI library will attempt to query the subnet to obtain the path record information before giving up.

12.12 MV2_PREPOST_DEPTH

- Class: Run time
- Default: 64

This defines the number of buffers pre-posted for each connection to handle send/receive operations.

12.13 MV2_RNDV_PROTOCOL

- Class: Run time
- Default: RPUT

The value of this variable can be set to choose different Rendezvous protocols. RPUT (default RDMA-Write) RGET (RDMA Read based), R3 (send/recv based).

12.14 MV2_R3_THRESHOLD

- Class: Run time
- Default: 4096

The value of this variable controls what message sizes go over the R3 rendezvous protocol. Messages above this message size use MV2_RNDV_PROTOCOL.

12.15 MV2_R3_NOCACHE_THRESHOLD

- Class: Run time
- Default: 32768

The value of this variable controls what message sizes go over the R3 rendezvous protocol when the registration cache is disabled (MV2_USE_LAZY_MEM_UNREGISTER=0). Messages above this message size use MV2_RNDV_PROTOCOL.

12.16 MV2_SRQ_LIMIT

- Class: Run Time
- Default: 30

This is the low water-mark limit for the Shared Receive Queue. If the number of available work entries on the SRQ drops below this limit, the flow control will be activated.

12.17 MV2_SRQ_SIZE

- Class: Run Time
- Default: 512

This is the maximum number of work requests allowed on the Shared Receive Queue.

12.18 MV2_STRIPING_THRESHOLD

- Class: Run Time
- Default: 8192

This parameter specifies the message size above which we begin the stripe the message across multiple rails (if present).

12.19 MV2_USE_BLOCKING

- Class: Run time
- Default: 0

Setting this parameter enables mvapich2 to use blocking mode progress. MPI applications do not take up any CPU when they are waiting for incoming messages.

12.20 MV2_USE_LAZY_MEM_UNREGISTER

- Class: Run time
- Default: set

Setting this parameter enables mvapich2 to use memory registration cache.

12.21 MV2_USE_RDMA_FAST_PATH

- Class: Run time
- Default: set

Setting this parameter enables MVAPICH2 to use adaptive rdma fast path features for the Gen2 interface.

12.22 MV2_USE_SRQ

- Class: Run time
- Default: set

Setting this parameter enables mvapich2 to use shared receive queue.

12.23 MV2_VBUF_POOL_SIZE

- Class: Run time
- Default: 512

The number of vbufs in the initial pool. This pool is shared among all the connections.

12.24 MV2_VBUF_SECONDARY_POOL_SIZE

- Class: Run time
- Default: 128

The number of vbufs allocated each time when the global pool is running out in the initial pool. This is also shared among all the connections.

12.25 MV2_VBUF_TOTAL_SIZE

- Class: Run time
- Default: Architecture dependent (6 KB for EM64T)

The size of each `vbuf`, the basic communication buffer of MVAPICH2. For better performance, the value of MV2_IBA_EAGER_THRESHOLD should be set the same as MV2_VBUF_TOTAL_SIZE.

12.26 MV2_RUN_THROUGH_STABILIZATION

- Class: Run Time
- Default: 0

This enables run through stabilization support to handle the process failures. This is valid only with Hydra process manager with –disable-auto-cleanup flag.

13 MPIRUN_RSH Parameters

13.1 MV2_COMM_WORLD_LOCAL_RANK

- Class: Run time
- Applicable Interface(s): All

The local rank of a process on a node within its job. The local rank ranges from 0,1 ... N-1 on a node with N processes running on it.

13.2 MV2_COMM_WORLD_LOCAL_SIZE

- Class: Run time
- Applicable Interface(s): All

The number of ranks from this job that are running on this node.

13.3 MV2_COMM_WORLD_RANK

- Class: Run time
- Applicable Interface(s): All

The MPI rank of this process in current MPI job

13.4 MV2_COMM_WORLD_SIZE

- Class: Run time
- Applicable Interface(s): All

The number of processes in this MPI job's MPI_Comm_World.

13.5 MV2_FASTSSH_THRESHOLD

- Class: Run time
- Default: 256
- Applicable Interface(s): All

Number of nodes beyond which to use hierarchical ssh during start up. This parameter is only relevant for mpirun_rsh based start up. Note that unlike most other parameters described in this section, this is an environment variable that has to be set in the run time environment (for e.g. through export in the bash shell).

13.6 MV2_NPROCS_THRESHOLD

- Class: Run time
- Default: 8192
- Applicable Interface(s): All

Number of nodes beyond which to use file-based communication scheme in the hierarchical ssh during start up. This parameter is only relevant for mpirun_rsh based start up.

13.7 MV2_MPIRUN_TIMEOUT

- Class: Run time
- Default: Dynamic - based on number of nodes

The number of seconds after which mpirun_rsh aborts job launch. Note that unlike most other parameters described in this section, this is an environment variable that has to be set in the run time environment (for e.g. through export in the bash shell).

13.8 MV2_MT_DEGREE

- Class: Run time
- Default: 32

The degree of the hierarchical tree used by mpirun_rsh. Note that unlike most other parameters described in this section, this is an environment variable that has to be set in the run time environment (for e.g. through export in the bash shell).

13.9 MPIEXEC_TIMEOUT

- Class: Run time
- Default: Unset
- Unit: Seconds

Set this to limit, in seconds, of the execution time of the mpi application. This overwrites the MV2_MPIRUN_TIMEOUT parameter.

13.10 MV2_DEBUG_FORK_VERBOSE

- Class: Run time
- Type: Null or positive integer
- Default: 0 (disabled)

Set the verbosity level of the debug output for the process management operations (fork, waitpid, kill, ...) of mpirun_rsh and mpispawn processes. The value 0 disables any debug output, a value of 1 enables a basic debug output, and a value of 2 enables a more verbose debug output.

Note: All debug output is disabled when MVAPICH2 is configured with the `--enable-fast=ndebug` option.