MVAPICH 0.9.5 User and Tuning Guide (Version 2.0)

MVAPICH Team

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

http://nowlab.cis.ohio-state.edu/projects/mpi-iba/

Contents

1 Overview of the Open-Source MVAPICH Project

InfiniBand is emerging as a high-performance interconnect delivering low latency and high bandwidth. It is also getting widespread acceptance due to its *open standard*.

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

Currently, there are two versions of this MPI: MVAPICH with MPI-1 semantics and MVAPICH2 with MPI-2 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 and InfiniBand community. As of the release date of MVAPICH 0.9.5, more than 200 organizations (National Labs, Universities, and Industry) in 26 countries have downloaded this software from OSU's web site directly. In addition, many IBA vendors, server vendors, and systems integrators have been incorporating MVAPICH/MVAPICH2 into their software stacks and distributing it. Several InfiniBand systems using MVAPICH have obtained positions in the TOP 500 ranking. Both MVAPICH and MVAPICH2 distributions are available under BSD licensing.

More details on MVAPICH/MVAPICH2 software, users list, sample performance numbers on a wide range of platforms and interconnect, a set of OSU benchmarks, related publications, and other InfiniBand-related projects (parallel file systems, storage, data centers) can be obtained from the following URL:

http://nowlab.cis.ohio-state.edu/projects/mpi-iba/

This document contains necessary information for MVAPICH users to download, install, test, use, and tune MVAPICH 0.9.5. A text version of this document is also included in the MVAPICH 0.9.5 software distribution. As we get feedbacks from users and take care of bug-fixes, we introduce new patches against our released distribution and also continuously update this document. Thus, we strongly request you to refer to our web page for updates.

2 MVAPICH 0.9.5 Features

MVAPICH (MPI-1 over VAPI for InfiniBand) is an MPI-1 implementation. Currently, the implementation is based on Verbs Level Interface (VAPI), developed by Mellanox Technologies. *An implementation on the OpenIB Gen2 layer is being developed and will be released in the near future.*

MVAPICH implementation is based on MPICH and MVICH. MVAPICH 0.9.5 is available as a single integrated package (with the latest MPICH 1.2.6 and MVICH). MVAPICH 0.9.5 is currently available and optimized for the following architectures. Support for EM64T

platform has been optimized for PCI-Express. Support for the new generation mem-free cards has also been tested. MVAPICH 0.9.5 supports the following platforms:

- EM64T
- G5
- IA-32
- IA-64
- Opteron

MVAPICH 0.9.5 supports several new features (compared to 0.9.4) including: multi-rail (multiple ports per adapter and multiple adapters), optimized intra-node shared memory support (both for bus-based and NUMA-based systems), enhanced MPI broadcast support with IBA hardware multicast, flexible mechanisms for minimizing memory resource usage on large scale systems, and support for TotalView debugger.

A complete set of features of MVAPICH 0.9.5 are:

- Optimized RDMA Write-based scheme for Eager protocol (short message transfer)
- Optimized implementation of Rendezvous protocol (large message transfer) with suitable flow control
- Efficient memory registration/de-registration schemes for RDMA-based communication
- Optimized intra-node communication support by taking advantage of shared-memory communication
  - Bus-based SMP systems
  - NUMA-based SMP systems
- Multi-rail support with different message scheduling policies
  - Multiple Adapters per node
  - Multiple Ports per Adapter
- High performance and scalable collective communication support Broadcast support using IBA hardware multicast mechanism
  - RDMA-based Barrier support
  - RDMA-based All-to-all support
- Schemes for minimizing memory resource usage on large scale systems

  - Default tuning for small, medium, and large clusters

- Flexibility to run with different job startup schemes
  - rsh/ssh based startup
  - MPD support for scalable startup

- Single codebase for different architectures/platforms with single-rail or multi-rail support
  - EM64T
  - IA-32
  - IA-64
  - G5
  - Opteron

- Tuned thresholds and associated optimizations for
  - different architectures/platforms mentioned above
  - different memory/system bus characteristics
  - different network interfaces (PCI-X and PCI-Express)

- Incorporates a set of runtime tunable parameters and a set of compile time tunable parameters for convenient tuning on
  - large scale systems
  - future platforms

- Shared library support for existing binary MPI application programs to run

- Support for TotalView debugger

3 Installation Instructions

3.1 Download MVAPICH source code

Since 0.9.4 version, the MVAPICH source code package includes the latest MPICH 1.2.6 version and also the required MVICH files from LBNL. Thus, there is no need to download any other files except mvapich 0.9.5 source code.

You can go to the MVAPICH website to obtain the source code.

3.2 Prepare MVAPICH source code

Untar the archive you have downloaded from the web page. Given 0.9.5 version, the following command can run on most Unix machines:

```
$ tar xzf mvapich-0.9.5.tgz
```

You will have a directory named `mvapich-0.9.5`.

3.3 Apply Patches to source tree

As we are enhancing and improving MVAPICH, we are also making patches available for customers to use the enhanced support. All patches along with their descriptions are provided in our software download website. To apply these patch files, place a downloaded copy in your mvapich directory, and execute the following command.

```
$ patch -p1 < patch_filename
```

Note: For convenience, we also make an integrated tarball with latest patches available on our software download website.

Since MVAPICH 0.9.5 provides support for Single-Rail and Multi-Rail as two seperate devices, they need to be appropriately built and configured. In the following section, we indicate how to build MVAPICH with Single-Rail configuration. In section 3.5, we discuss about building MVAPICH with Multi-Rail configuration.

3.4 Build MVAPICH with Single-Rail Configuration

3.4.1 Configure MVAPICH

There are several options to configure MVAPICH. Please select the following option based on your need.

- *Default configuration*

  Go to the `mvapich-0.9.5` directory. For your reference, we have included several scripts about how to configure MVAPICH. Please see `mvapich.make.gcc, mvapich.make.icc, mvapich.make.ecc`, and `mvapich.make.macosx` in the `mvapich-0.9.5` directory for details. You can customize your configuration according to the comments in these files and then run the appropriate script directly.

- *Manual configuration*

  If you would like to configure MVAPICH manually, do so using `./configure` script in the `mvapich-0.9.5` directory. But please look at either `mvapich.make.gcc`, `mvapich.make.icc, mvapich.make.ecc,` and `mvapich.make.macosx` for appropriate configuration options. The following option is mandatory.

  `--with-device=vapi`

  Other options are reflected in the environmental variables such as `CFLAGS, FFLAGS, CXXFLAGS` and `F90FLAGS`. For MacOS platform, `MAC_OSX=yes` should be exported. We strongly recommend users to read `mvapich.make.macosx` before their manual configuration on the MacOS platform. Some configuration examples on IA32 machines and MacOS can be found in the sections 7.1.1 and 7.1.2.

- *Configure MVAPICH using MPD*

  By default, MVAPICH is configured using `mpirun_rsh` to launch applications. If you want to use MPD and want to use our provided `mvapich.make.gcc/mvapich.make.icc/ mvapich.make.ecc/mvapich.make.macosx files`, you should customize `MPD_SUPPORT`. If you use your manual configuration, `-DUSE_MPD_BASIC` or `-DUSE_MPD_RING` should be added into the above `CFLAGS`. In addition, `make install` is mandatory. An example is shown in section 7.1.3, "Configuration Example with MPD Support".

- *Configure MVAPICH using InfiniBand hardware Multicast-based MPI Broadcast*

  If you want to use our InfiniBand hardware Multicast-based MPI Broadcast implementation, special configuration should be taken into account. Details can be found in section 7.1.4, "Configuration Example with Multicast-based Broadcast".

3.4.2 make

After configuration, type `make` in the `mvapich-0.9.5` directory. If you use `mvapich.make.gcc, mvapich.make.icc`, or `mvapich.make.ecc`, this step can be skipped because it is included in the scripts.

3.4.3 make install

After make, type `make install` in the `mvapich-0.9.5` directory to install MVAPICH in the directory as specified by `--prefix`. Note that if you want to have MPD support, this step is mandatory. Note that this step is also included in our provided scripts, if you want to use our scripts, please customize the `INSTALL_PATH`.

3.5 Build MVAPICH with Multi-Rail Configuration

3.5.1 Configure MVAPICH

There are several options to configure MVAPICH. Please select the following option based on your need.

- *Default configuration*

  Go to the `mvapich-0.9.5 directory`. For your reference we have included several scripts about how to configure MVAPICH for your reference. Please see configuration scripts `multirail.make.gcc, multirail.make.icc, multirail.make.ecc`, and `multirail.make.macosx` in the `mvapich-0.9.5` directory for details. You can customize your configuration according to the comments in these files and then run the appropriate script directly.

- *Manual configuration*

  If you would like to configure MVAPICH manually, you can do that by typing `./configure` script in the `mvapich-0.9.5` directory. But please look at either `multirail.make.gcc, multirail.make.icc, multirail.make.ecc`, and `multirail.make.macosx` for appropriate configuration options. The following option is mandatory.

  `--with-device=vapi_multirail`

  Other options are reflected in the environmental variables such as `CFLAGS, FFLAGS, CXXFLAGS` and `F90FLAGS`. For MacOS platform, `MAC_OSX=yes` should be exported. We strongly recommend users to read `mvapich.make.macosx` before their manual configuration on the MacOS platform. Some configuration examples on IA32 machines and MacOS can be found in sections 7.2.1 and 7.2.2.

- *Configure MVAPICH using MPD*

  By default, MVAPICH is configured using `mpirun_rsh` to launch applications. If you want to use MPD and want to use our provided `multirail.make.gcc multirail.make.icc /multirail.make.ecc/ multirail.make.macosx` files, you should customize `MPD_SUPPORT`. If you use your manual configuration, `-DUSE_MPD_BASIC` or `-DUSE_MPD_RING` should be added into the above `CFLAGS`. In addition, `make install` is mandatory. An example is shown in section 7.2.3, "Configuration Example with MPD support".

- *Configure MVAPICH using InfiniBand hardware Multicast-based MPI Broadcast*

  If you want to use our InfiniBand hardware Multicast-based MPI Broadcast implementation, special configuration should be taken into account. Details can be found in section 7.2.4, "Configuration Example with Multicast-based Broadcast".

3.5.2 make

After configuration, type `make` in the `mvapich-0.9.5` directory. If you use `multirail.make.gcc`, `multirail.make.icc`, or `multirail.make.ecc`, this step can be skipped because it is included in the scripts.

3.5.3 make install

After make, type `make install` in the `mvapich-0.9.5` directory to install MVAPICH in the directory as specified by `--prefix`. Note that if you want to have MPD support, this step is mandatory. Note that this step is also included in our provided scripts, if you want to use our scripts, please customize the `INSTALL_PATH`.

4 Usage Instructions

4.1 Compile MPI applications

Use `mvapich-0.9.5/bin/mpicc`, `mvapich-0.9.5/bin/mpif77`, `mvapich-0.9.5/bin/mpiCC`, or `mvapich-0.9.5/bin/mpif90` to compile applications.

There are several options to run MPI applications. Please select one of the following options based on your need.

4.2 Run MPI applications using mpirun_rsh

Prerequisites:

- Either `ssh` or `rsh` should be enabled between the front nodes and the computing nodes.
- Configuring and installing MVAPICH without MPD support.

`mpirun_rsh` examples:

```
$ mpirun_rsh -np 4 n0 n1 n2 n3 ./cpi
```

`cpi` runs on n0, n1, n2 and n3 nodes, one process per each node. The default `ssh` is used.

```
$ mpirun_rsh -rsh -np 4 n0 n1 n2 n3 ./cpi
```

`cpi` runs on n0, n1, n2 and n3 nodes, one process per each node. `rsh` is used regardless of whether `ssh` or `rsh` is built in the installation time.

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

A list of nodes are in hosts, one per line.

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

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

Other options of `mpirun_rsh` can be obtained using

```
$ mpirun_rsh --help
```

4.3 Run MPI applications using MPD

MVAPICH is provided with MPD support for fast process startup. To configure and build MVAPICH with MPD support, see section 3.4.1 or section 3.5.1. Be sure to do `make install` to have MPD system installed into the correct directory. This is a general requirement for using extended features, such as MPD and TotalView, with MPICH.

To know more about MPD, please refer to the MPD documents provided along Argonne MPICH release. This should be available as `mvapich-0.9.5/doc/mpichman-chp4mpd.pdf`, section 4.9. An online document is also available from MPICH website.

Step by step instructions to setup MPD environment manually:

For an instant setup of MPD environment over two nodes node00 and node01.

- First log into node00 and then proceed with the following steps.
  - Be sure you have .mpd.conf and .mpdpasswd in your home directory. They can be a single line file like the following:
    `password=56rtG9`
  - Include MPD path into your path
    ```
    $ export MPD_BIN=$MVAPICH_HOME/bin
    $ export PATH=$MVAPICH_HOME/bin:$PATH
    ```
    `$MVAPICH_HOME` is the installation path of your MVAPICH, as specified by `--prefix` when you configure MVAPICH.
  - run `mpd` on node00
  - Find out the port number this daemon is exposing, i.e. typically a number from the following trace command. In the following lines this number is assumed to be 33333.
    ```
    $ mpdtrace &
    ```
  - Launch another daemon on node01
    ```
    $ ssh -n node01 ${MPD_BIN}/mpd -h node00 -p 33333 &
    ```
  - Simple Testing
    ```
    $ mpirun_mpd -np 4 cpi
    ```
  - Cleanup
    ```
    $ mpdallexit
    ```
- We provide a script in the `mvapich-0.9.5` directory. You can use this script to expedite MPD setup.

  - Be sure you have .mpd.conf and .mpdpasswd in your home directory. They can be a single line file like the following:
    password=56rtG9
  - Make a sample machine file, hostfile, which reads
    node00
    node01
    node02
    node03
  - Startup daemons
    `$ mvapich.mpd.sh start hostfile $MPD_BIN/mpd`
  - Stop daemons `$ mvapich.mpd.sh stop hostfile $MPD_BIN/mpd`
  - Cleanup daemons if you have trouble
    `$ mvapich.mpd.sh cleanup hostfile $MPD_BIN/mpd`

- Environmental variables setup using mpirun_mpd in MVAPICH

  `mpirun_mpd -np $np $prog <args> -MPDENV- ENV1=value1 ENV2=value2`

  Details can be referred from MPICH Website.

4.4 Run MPI applications using InfiniBand hardware Multicast based MPI Broadcast support

In mvapich-0.9.5, we provide a hardware multicast-based MPI Broadcast. Prerequisites for using this support are:

- Configuring and building MVAPICH with hardware multicast-based MPI Broadcast support. Please refer to section 3.4.1 or section 3.5.1 for related configuration and installation.
- Subnet Management (SM) Support: We have developed and tested this feature using OpenSM. Thus, we provide instructions w.r.t. OpenSM. If you are using any other SM, please make appropriate adjustment to the following paths and steps.
- OpenSM has to be run continuously on one node for hardware multicast to work. We recommend to use a non-compute node as the node running Opensm. If OpenSM is not already running on one of the nodes connected to the subnet then follow the procedure below to run OpenSM.
- Run opensm with a GUID choice. The GUID choice is essentially the port number. You can choose 1 or 2, which is the port you want to use. But make sure that whichever port you choose, it must be connected to the IB subnet.

4.4.1 Usage examples:

When MVAPICH is configured and installed with hardware multicast- based MPI Broadcast support, MPI_Bcast takes advantage of hardware multicast for broadcasting messages reliably.

This feature can be disabled by using an environment variable, `DISABLE_HARDWARE_MCST`, as shown below:

```
$ mpirun_rsh -np 4 n0 n1 n2 n3 DISABLE_HARDWARE_MCST=1 ./cpi
```

MPI_Bcast will use the original point-to-point based implementation in MPICH-1.2.6 when `DISABLE_HARDWARE_MCST` is set. Note that, `DISABLE_HARDWARE_MCST=1` should be put immediately before the executable file.

Important notes:

- If the multicast group create/join fails, restarting Opensm helps.
- If you are still facing the problem above, please create the multicast group manually by executing the `ibmcgrp` command found in the `$MVAPICH_HOME/bin` directory.

  example:

  ```
  $ ibmcgrp -c -g 0xff12a01cfe800000:HHHHHHHHHHHHHHHH --port_num 1
  ```

- In the current implementation we support a single multicast group which includes all the nodes. Thus MPI_COMM_WORLD and any communicator which includes all the nodes can take advantage of the hardware multicast based MPI_Bcast. We are working on extending this feature to support arbitrary communicators.

4.5 Run MPI applications using RDMA-based MPI Alltoall support

In MVAPICH 0.9.5, we provide a direct-RDMA based MPI_Alltoall implementation by default. When you want to disable this implementation and use the original implementation based on MPI point-to-point communication, you can put `DISABLE_RDMA_ALLTOALL=1` in the command line. For example,

```
$ mpirun_rsh -np 4 n0 n1 n2 n3 DISABLE_RDMA_ALLTOALL=1 ./cpi
```

MPI_Alltoall will use point-to-point based implementation, when `DISABLE_RDMA_ALLTOALL` is set. Note that, `DISABLE_RDMA_ALLTOALL=1` should be put immediately before the executable.

4.6 Run MPI applications using shared library support

MVAPICH provides shared library support. This feature allows you to build your application on top of shared library support. If you chose this option, you still will be able to compile applications with static libraries. An example of configuring and building MVAPICH with shared library support is provided in section 7.1.5. With MVAPICH shared library support enabled, your applications will be built on top of shared libraries by default. The following commands provide some examples of how to build and run your application with shared library support.

- To compile your application with shared library support. Run the following command.

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

- To execute an application compiled with shared library support, you need to specify the path to the shared library by putting
  `LD_LIBRARY_PATH=<path-to-shared-libraries>` in the command line.

  For example,

  ```
  $ mpirun_rsh -np 2 n0 n1 LD_LIBRARY_PATH=$MVAPICH_BUILD/lib/shared ./cpi
  ```

  Again, note that "LD_LIBRARY_PATH=path-to-shared-libraries" should be put immediately before the executable file.

- To disable MVAPICH shared library support even if you have installed MVAPICH. Run the following command.

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

4.7 Run MPI applications using TotalView Debugger support

MVAPICH 0.9.5 provides TotalView support for the single rail device: `mpid/vapi`. You need to use `mpirun_rsh` when running TotalView. An example of configuring and building MVAPICH with TotalView support is provided in section 7.1.6. The following commands also provide an example of how to build and run your application with TotalView support. *Note: running TotalView demands correct setup in your environment, if you encounter any problem with your setup, please check with your system adminstrator for help.*

- Define ssh as a `TVDSVRLAUNCHCMD` variable in your default shell. For example, with bashrc, you can do
  `$ echo "export TVDSVRLAUNCHCMD=ssh" >> /.bashrc`

- Configure mvapich with the configure options `--enable-debug --enable-sharedlib` in addition to the default options and then build mvapich.

- Compile your program with a flag -g
  `$ mpicc -g -o prog prog.c`
- Define the correct path to TotalView as the TOTALVIEW variable. For example, under bash shell:
  `$ export TOTALVIEW=<path_to_TotalView>`
- Run your program:
  `$ mpirun_rsh -tv -np 2 n0 n1`
  `LD_LIBRARY_PATH=$MVAPICH_BUILD/lib/shared:$MVAPICH_BUILD/lib`
  `prog`
- Trouble shooting:
  - X authentication errors: check if you have enabled X Forwarding
    `$ cat ‘‘ForwardX11 yes’’ >> $HOME/.ssh/config`
  - rsh connection time out: check if you have defined `TVDSVRLAUNCHCMD` as ssh in your default shell file, .bashrc, .cshrc, or the like.
  - ssh authentication error: ssh to the computer node with its long form hostname, for example, ssh i0.domain.osu.edu

5 Using OSU Benchmarks

If you have arrived at this point, you have successfully installed MVAPICH. Congratulations!! In the `mvapich-0.9.5/osu_benchmarks` directory, we provide four basic performance tests: a one-way latency test, a uni-directional bandwidth test, a bi-directional bandwidth test and a MPI-level broadcast latency test. You can compile and run these tests on your machines to evaluate the basic performance of MVAPICH.

These benchmarks as well as other benchmarks (such as for one-sided operations in MPI-2) are available on our projects' web page. Sample performance numbers for these benchmarks on representative platforms and IBA gears 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 at mvapich-help@cse.ohio-state.edu.

6 Troubleshooting with MVAPICH

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-help@cse.ohio-state.edu.

6.1 Cannot pass MPI_Init

If your MPI application cannot pass `MPI_Init`, please make sure of the following things:

- If you have enabled `ssh` based startup, make sure that you have set up ssh keys for logging into all the nodes without any password prompt.
- If you have enabled `rsh` based startup, make sure that `rsh, rlogin` etc. are active on all the nodes and you can log in without any password prompts.
- Please make sure you can run some InfiniBand level program on the nodes you are trying to run MPI programs. Usually running `perf_main` (distributed with IBGD) is a good choice of such a program.

6.2 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

```
% locate libvapi
```

which gives the path at which drivers are setup.

6.3 Cannot include vapi.h

This error is generated during compilation, if the correct path to the InfiniBand library installation is not given.

For IB Gold-0.5.0, the installation is present at:

`/usr/local/ib_hpc/ib/infinihost`

Please setup the environment variable MTHOME as

```
% export MTHOME=/usr/local/ib_hpc/ib/infinihost
```

For IB Gold-1.6.* and 1.7.0, the installation is present at:

`/usr/local/ibgd/driver/infinihost`

Please setup the environment variable MTHOME as

```
% export MTHOME=/usr/local/ibgd/driver/infinihost
```

If the problem persists, please contact your system administrator or send an email to mvapich-help@cse.ohio-state.edu.

6.4 VAPI_RETRY_EXEC_ERROR

This error usually indicates that all InfiniBand links the MPI application is trying to use are not in the `PORT_ACTIVE` state. Please make sure that all ports show `PORT_ACTIVE` with the VAPI utility `vstat`. If you are using Multi-Rail support, please keep in mind that all ports of all adapters you are using need to show `PORT_ACTIVE`.

6.5 Building mvapich hangs with hardware multicast enabled

We have found out that on some of our machines, when we build MVAPICH with hardware multicast-based Broadcast support, the system may hang in the *make* step. If this happens, please provide `"--disable-cxx"` in your configure command or add this option in the configure command in our scripts.

6.6 ld:multiple definitions of symbol _calloc error on MacOS

Please make sure that the environmental variable `"MAC_OSX"` is set before your configuration. If you use manual configuration and not `mvapich.make.macosx`, you must configure MVAPICH in the following way:

```
   export MAC_OSX=yes
./configure ..with option..
make
make install
```

If you encounter this problem compiling your own applications,

```
   "ld:  multiple definitions of symbol _calloc
/usr/lib/libm.dylib(malloc.So) definition of _calloc
/tmp/mvapich-0.9.5/mvapich/lib/libmpich.a(dreg-g5.o)
definition of _calloc in section (__TEXT,__text)
ld:  multiple definitions of symbol _free
/usr/lib/libm.dylib(malloc.So) definition of _free
```

```
/tmp/mvapich-0.9.5/mvapich/lib/libmpich.a(dreg-g5.o)
definition of _free in section (__TEXT,__text) "
```

it is likely that you have explicitly included `"-lm"`. You should remove that.

6.7 No Fortran interface on the MacOS platform.

To enable Fortran support, you would need to install the IBM compiler located at (there is a 60-day free trial version) available from IBM.

Once you unpack the tarball, you can customize and use `mvapich.make.macosx` to compile and install the package or manually configure, compile and install the package.

6.8 Other MPICH problems

Several well-known MPICH related problems on different platforms and environments have already been identified by Argonne. They are available on the MPICH patch webpage.

7 Configuration Examples

In this section, we provide a set of sample configuration examples for easy reference.

7.1 Configuration Examples for Single-Rail MVAPICH

7.1.1 A Typical Configuration without MPD and Multicast-Based Broadcast on IA32

```
#!/bin/bash

   export CFLAGS="-D_SMP_ -D_SMP_RNDV_

   -DUSE_INLINE -DEARLY_SEND_COMPLETION

   -DVIADEV_RPUT_SUPPORT -DLAZY_MEM_UNREGISTER

   -DRDMA_FAST_PATH

   -D_IA32_ -O3

   -I/usr/local/ib_hpc/ib/infinihost/include"

   export FFLAGS="-L/usr/local/ib_hpc/ib/infinihost/lib"

   export CXXFLAGS=$CFLAGS

   ./configure --with-device=vapi --with-arch=LINUX

   --prefix="/usr/local/mvapich"

   -lib="-L/usr/local/ib_hpc/ib/infinihost/lib -lmtl_common

   -lvapi -lmosal -lmpga -lpthread"
```

In this example, the system architecture is IA32, indicated by `"-D_IA32_"`; InfiniBand installation is `"/usr/local/ib_hpc/ib/infinihost"`; mpirun_rsh is configured in which `"ssh"` is used; the MVAPICH installation path is `"/usr/local/mvapich"`, indicated by `"--prefix"`; GNU compilers are used.

For using other compilers, please refer to mvapich.make.icc and mvapich.make.ecc files in the mvapich-0.9.5 directory for details.

7.1.2 A Typical Configuration without MPD and Multicast-Based Broadcast on MacOS

```
#!/bin/bash
```

```
export MAC_OSX=yes
export CC=gcc
export CXX=g++
export FC=f77
export FFLAGS="-O5"
export CFLAGS="-D_SMP_ -D_SMP_RNDV_
-DUSE_INLINE -DEARLY_SEND_COMPLETION
-DVIADEV_RPUT_SUPPORT -DLAZY_MEM_UNREGISTER
-DRDMA_FAST_PATH
-D_IA32_ -O3
-I/usr/local/ib_hpc/ib/infinihost/include"
export FFLAGS="-L/usr/local/ib_hpc/ib/infinihost/lib"
export CXXFLAGS=$CFLAGS
./configure --with-device=vapi
--prefix="/usr/local/mvapich"
-lib="-L/usr/local/ib_hpc/ib/infinihost/lib -lmtl_common
-lvapi -lmosal -lmpga -lpthread
-lxlf90 -lxlfmath -L/opt/ibmcmp/xlf/8.1/lib
-multiply_defined suppress"
```

InfiniBand installation is `"/usr/local/ib_hpc/ib/infinihost"`; mpirun_rsh is configured in which `"ssh"` is used; the MVAPICH installation path is `"/usr/local/mvapich"`, indicated by `"--prefix"`.

7.1.3 Configuration Example with MPD Support

```
#!/bin/bash
   export CFLAGS="-D_SMP_ -D_SMP_RNDV_
   -DUSE_INLINE -DEARLY_SEND_COMPLETION
   -DVIADEV_RPUT_SUPPORT -DLAZY_MEM_UNREGISTER
   -DRDMA_FAST_PATH
```

```
-D_IA32_ -O3
-DUSE_MPD_BASIC
-I/usr/local/ib_hpc/ib/infinihost/include"
export FFLAGS="-L/usr/local/ib_hpc/ib/infinihost/lib"
export CXXFLAGS=$CFLAGS
./configure --with-device=vapi --with-arch=LINUX
--prefix="/usr/local/mvapich"
-lib="-L/usr/local/ib_hpc/ib/infinihost/lib -lmtl_common
-lvapi -lmosal -lmpga -lpthread"
```

You can add either "-DUSE_MPD_BASIC" or "-DUSE_MPD_RING" in the "CFLAGS" to configure MVAPICH using MPD.

7.1.4 Configuration Example with Multicast-Based Broadcast Support

Our current implementation requires OpenSM and has been tested with OpenSM. For other subnet managers, appropriate modifications are needed. OpenSM should be installed on your system for multicast to work. In the following example, we assume your OpenSM is installed at "/usr/local/ib_hpc/ib/apps/osm/". The following things have to be done next.

In the first step, -DOSM_VENDOR_INTF_TS -DMCST_SUPPORT should be added into CFLAGS. Second, an appropriate include path to the OpenSM header files should be included as well. Third, a lib path for the OpenSM library files should be added into the "-lib" option in the configure command.

```
#!/bin/bash
export CFLAGS="-D_SMP_ -D_SMP_RNDV_
-DUSE_INLINE -DEARLY_SEND_COMPLETION
-DVIADEV_RPUT_SUPPORT -DLAZY_MEM_UNREGISTER
-DRDMA_FAST_PATH
-DOSM_VENDOR_INTF_TS -DMCST_SUPPORT
-D_IA32_ -O3
-DUSE_MPD_BASIC
-I/usr/local/ib_hpc/ib/infinihost/include
-I/usr/local/ib_hpc/ib/apps/osm/include"
```

```
export FFLAGS="-L/usr/local/ib_hpc/ib/infinihost/lib"
export CXXFLAGS=$CFLAGS
./configure --with-device=vapi --with-arch=LINUX
--prefix="/usr/local/mvapich"
-lib="-L/usr/local/ib_hpc/ib/infinihost/lib -lmtl_common
-lvapi -lmosal -lmpga -lpthread
-L/usr/local/ib_hpc/ib/apps/osm/lib"
```

Note that if you use our provided scripts, you should customize "MCST_SUPPORT" in mvapich.make.gcc/mvapich.make.icc/mvapich.make.ecc.

7.1.5 Configuration Example with Shared Library Support

To use shared library support, you need to introduce an extra option '--enable-sharedlib' into any of the above sample configuration scripts for the corresponding setup. The following provides an example of how to build shared library support for IA32 platform.

```
#!/bin/bash
export CFLAGS="-D_SMP_ -D_SMP_RNDV_
-DUSE_INLINE -DEARLY_SEND_COMPLETION
-DVIADEV_RPUT_SUPPORT -DLAZY_MEM_UNREGISTER
-DRDMA_FAST_PATH
-D_IA32_ -O3
-I/usr/local/ib_hpc/ib/infinihost/include"
export FFLAGS="-L/usr/local/ib_hpc/ib/infinihost/lib"
export CXXFLAGS=$CFLAGS
./configure --with-device=vapi --with-arch=LINUX
--prefix="/usr/local/mvapich" --enable-sharedlib
-lib="-L/usr/local/ib_hpc/ib/infinihost/lib -lmtl_common
-lvapi -lmosal -lmpga -lpthread"
```

7.1.6 Configuration Example with TotalView Support

To prepare TotalView support, you need to introduce extra options "--enable-debug" and "--enable-sharedlib" into your normal configuration scripts. The following provides an example of how to build TotalView support for IA32 platform. Note:

- TotalView support is provided for the single rail device: mpid/vapi.
- mpirun_rsh is needed to run TotalView.
- TotalView support is available to IA32, IA64, EM64T and Opteron platforms.

```
#!/bin/bash
export CFLAGS="-D_SMP_ -D_SMP_RNDV_
-DUSE_INLINE -DEARLY_SEND_COMPLETION
-DVIADEV_RPUT_SUPPORT -DLAZY_MEM_UNREGISTER
-DRDMA_FAST_PATH
-D_IA32_ -O3
-I/usr/local/ib_hpc/ib/infinihost/include"
export FFLAGS="-L/usr/local/ib_hpc/ib/infinihost/lib"
export CXXFLAGS=$CFLAGS
./configure --with-device=vapi --with-arch=LINUX
--prefix="/usr/local/mvapich" --enable-debug --enable-sharedlib
-lib="-L/usr/local/ib_hpc/ib/infinihost/lib -lmtl_common
-lvapi -lmosal -lmpga -lpthread"
```

7.2 Configuration Examples for Multi-Rail MVAPICH

7.2.1 A Typical Configuration without MPD and Multicast-Based Broadcast on IA32

```
#!/bin/bash
export CFLAGS="-D_SMP_ -D_SMP_RNDV_
-DUSE_INLINE -DEARLY_SEND_COMPLETION
-DVIADEV_RPUT_SUPPORT -DLAZY_MEM_UNREGISTER
```

```
-DRDMA_FAST_PATH
-D_IA32_ -O3
-I/usr/local/ib_hpc/ib/infinihost/include"
export FFLAGS="-L/usr/local/ib_hpc/ib/infinihost/lib"
export CXXFLAGS=$CFLAGS
./configure --with-device=vapi_multirail --with-arch=LINUX
--prefix="/usr/local/mvapich"
-lib="-L/usr/local/ib_hpc/ib/infinihost/lib -lmtl_common
-lvapi -lmosal -lmpga -lpthread
```

In this example, the system architecture is IA32, indicated by "-D_IA32_"; InfiniBand installation is "/usr/local/ib_hpc/ib/infinihost"; mpirun_rsh is configured in which "ssh" is used; the MVAPICH installation path is "/usr/local/mvapich", indicated by "--prefix"; GNU compilers are used.

For using other compilers, please refer to multirail.make.icc and multirail.make.ecc files in the mvapich-0.9.5 directory for details.

7.2.2 A Typical Configuration without MPD and Multicast-Based Broadcast on MacOS

```
#!/bin/bash
   export MAC_OSX=yes
   export CC=gcc
   export CXX=g++
   export FC=f77
   export FFLAGS="-O5"
   export CFLAGS="-D_SMP_ -D_SMP_RNDV_
   -DUSE_INLINE -DEARLY_SEND_COMPLETION
   -DVIADEV_RPUT_SUPPORT -DLAZY_MEM_UNREGISTER
   -DRDMA_FAST_PATH
   -D_IA32_ -O3
   -I/usr/local/ib_hpc/ib/infinihost/include"
```

```
export FFLAGS="-L/usr/local/ib_hpc/ib/infinihost/lib"
export CXXFLAGS=$CFLAGS
./configure --with-device=vapi_multirail
--prefix="/usr/local/mvapich
-lib="-L/usr/local/ib_hpc/ib/infinihost/lib -lmtl_common
-lvapi -lmosal -lmpga -lpthread
-lxlf90 -lxlfmath -L/opt/ibmcmp/xlf/8.1/lib
-multiply_defined suppress"
```

InfiniBand installation is "/usr/local/ib_hpc/ib/infinihost"; mpirun_rsh is configured in which "ssh is used; the MVAPICH installation path is "/usr/local/mvapich", indicated by "--prefix".

7.2.3 Configuration Example with MPD Support

```
#!/bin/bash
export CFLAGS="-D_SMP_ -D_SMP_RNDV_
-DUSE_INLINE -DEARLY_SEND_COMPLETION
-DVIADEV_RPUT_SUPPORT -DLAZY_MEM_UNREGISTER
-DRDMA_FAST_PATH
-D_IA32_ -O3
-DUSE_MPD_BASIC
-I/usr/local/ib_hpc/ib/infinihost/include"
export FFLAGS="-L/usr/local/ib_hpc/ib/infinihost/lib"
export CXXFLAGS=$CFLAGS
./configure --with-device=vapi_multirail --with-arch=LINUX
--prefix=‘‘/usr/local/mvapich"
-lib="-L/usr/local/ib_hpc/ib/infinihost/lib -lmtl_common
-lvapi -lmosal -lmpga -lpthread"
```

You can add either "-DUSE_MPD_BASIC" or "-DUSE_MPD_RING" in the "CFLAGS" to configure MVAPICH using MPD.

7.2.4 Configuration Example with Multicast-Based Broadcast Support

Our current implementation requires OpenSM and has been tested with OpenSM. For other subnet managers, appropriate modifications are needed. "OpenSM" should be installed on your system for multicast to work. In the following example, we assume your OpenSM is installed at /usr/local/ib_hpc/ib/apps/osm/. The following things have to be done next.

In the first step, "-DOSM_VENDOR_INTF_TS -DMCST_SUPPORT" should be added into CFLAGS. Second, an appropriate include path to the OpenSM header files should be included as well. Third, a lib path for the OpenSM library files should be added into the "-lib" option in the "configure" command.

```
#!/bin/bash

export CFLAGS="-D_SMP_ -D_SMP_RNDV_

-DUSE_INLINE -DEARLY_SEND_COMPLETION

-DVIADEV_RPUT_SUPPORT -DLAZY_MEM_UNREGISTER

-DRDMA_FAST_PATH

-DOSM_VENDOR_INTF_TS -DMCST_SUPPORT

-D_IA32_ -O3

-DUSE_MPD_BASIC

-I/usr/local/ib_hpc/ib/infinihost/include

-I/usr/local/ib_hpc/ib/apps/osm/include"

export FFLAGS="-L/usr/local/ib_hpc/ib/infinihost/lib"

export CXXFLAGS=$CFLAGS

./configure --with-device=vapi_multirail --with-arch=LINUX

--prefix="/usr/local/mvapich"

-lib="-L/usr/local/ib_hpc/ib/infinihost/lib -lmtl_common

-lvapi -lmosal -lmpga -lpthread

-L/usr/local/ib_hpc/ib/apps/osm/lib"
```

Note that if you use our provided scripts, you should customize "MCST_SUPPORT" in multirail.make.gcc/multirail.make.icc /multirail.make.ecc.

7.2.5 Configuration Example with Shared Library Support

To use shared library support, you need to introduce an extra option '--enable-sharedlib' into any of the above sample configuration scripts for the corresponding setup. The following provides an example of how to build shared library support for IA32 platform.

```
#!/bin/bash

export CFLAGS="-D_SMP_ -D_SMP_RNDV_

-DUSE_INLINE -DEARLY_SEND_COMPLETION

-DVIADEV_RPUT_SUPPORT -DLAZY_MEM_UNREGISTER

-DRDMA_FAST_PATH

-D_IA32_ -O3

-I/usr/local/ib_hpc/ib/infinihost/include"

export FFLAGS="-L/usr/local/ib_hpc/ib/infinihost/lib"

export CXXFLAGS=$CFLAGS

./configure --with-device=vapi_multirail --with-arch=LINUX

--prefix="/usr/local/mvapich" --enable-sharedlib

-lib=‘"-L/usr/local/ib_hpc/ib/infinihost/lib -lmtl_common

-lvapi -lmosal -lmpga -lpthread"
```

8 Performance Tuning

MVAPICH supports many different parameters for tuning performance for a wide variety of applications. These parameters can be either compile time parameters or run time parameters. Please refer to section 9 for a complete description of all the 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 Point-to-Point Tuning

Point-to-point latency, bandwidth can be tuned very simply by using the parameters VIADEV_RENDEZVOUS_THRESHOLD (9.22) and VIADEV_NUM_RDMA_BUFFER (9.17).

Messages larger than VIADEV_RENDEZVOUS_THRESHOLD will go over the Rendezvous protocol using zero copy. While this can reduce the number of copies, it can be costly for small messages.

VIADEV_NUM_RDMA_BUFFER indicates the number of RDMA buffers per connection. If this parameter is increased, more outstanding messages can be transferred by using the fast path. However, increasing this parameter also leads to increased memory usage.

8.2 Tuning Memory Usage

Memory usage often plays a significant role in application performance, and especially more so for large scale clusters. The main parameters which decide the memory usage are :

- VBUF_TOTAL_SIZE (9.12)
- VIADEV_NUM_RDMA_BUFFER (9.17)
- VIADEV_VBUF_POOL_SIZE (9.23)

VIADEV_VBUF_POOL_SIZE is a fixed number of pool of `vbufs`. These vbufs can be shared among all different connections depending on the communication needs of each connection.

On the other hand, the product of VBUF_TOTAL_SIZE and VIADEV_NUM_RDMA_BUFFER generally is a measure of the amount of memory to be pinned down for eager message passing. These buffers are not shared across connections.

In the earlier MVAPICH versions, VBUF_TOTAL_SIZE was equal to the VIADEV_RENDEZVOUS_THRESHOLD. From 0.9.5, this restriction has been removed. For a given platform, these two parameters can be independently tuned.

To provide the best performance (latency/bandwidth) to memory ratio, we have decided on a set of default values for these parameters. These parameters are often dependent on the execution platform. To use preset values for small, medium and large clusters (1-64, 64-256, 256-...), please use -D_SMALL_CLUSTER, -D_MEDIUM_CLUSTER and -D_LARGE_CLUSTER respectively.

8.3 Tuning VAPI Parameters

In addition to the above MPI point-to-point parameters, there are some VAPI parameters which have some impact on performance. They are:

- VIADEV_SQ_SIZE (9.15)
- VIADEV_DEFAULT_MAX_SG_LIST (9.21)

VIADEV_SQ_SIZE is the number of outstanding sends for each connection. If this is higher, more outstanding sends can be supported. However, increasing this value leads to bigger InfiniBand Queue Pair memory regions and overall higher memory usage.

VIADEV_DEFAULT_MAX_SG_LIST is the number of scatter-gather entries for each InfiniBand Queue Pair. Usually, only one entry is needed. However, this parameter has an impact on increasing the inline size supported by the Queue Pair. This can reduce latency for small messages. However, it should not be increased too much, since it consumes more resources per Queue Pair.

8.4 Shared Memory Tuning

MVAPICH 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 are SMPI_LENGTH_QUEUE (9.7) and SMP_EAGERSIZE (9.8).

SMPI_LENGTH_QUEUE is the size of the shared memory buffer which is used to store outstanding messages. Increasing this value leads to more buffer space being available to store outstanding messages, at the cost of more memory usage.

From 0.9.5, we support pipelining of shared memory message passing based on packets. The compile time flag _SMP_RNDV_ is used to enable that. We suggest that while using shared memory support, this flag should be turned on.

SMP_EAGERSIZE is the size of a message which will be copied to the shared memory buffer as one packet. Decreasing this will lead to more pipelining, but if it is decreased too much then the throughput might suffer.

Also from 0.9.5 we provide support for NUMA platforms. By default, MVAPICH will adjust the location of shared memory buffers to provide best performance for NUMA platforms.

8.5 InfiniBand Hardware Multicast Tuning

The following is the set of parameters which can be tuned to get better MPI_Bcast performance using Hardware Multicast of InfiniBand.

8.5.1 MCST_THRESHOLD

MCST_THRESHOLD (9.2) can be tuned depending on the size of the cluster. Currently, the parameter is set to 8KB based on an experimentation on 32 node systems. For large scale systems, this threshold may be increased to get the benefit of hardware-based multicast for larger messages. For example, this threshold can be increased to 16 KB for 64-node systems, 32 KB for 128-node systems, and so on.

8.5.2 VIADEV_UD_PREPOST_DEPTH

VIADEV_UD_PREPOST_DEPTH (9.29) can affect the performance of an application which issues multiple back-to-back multicasts. Increasing this parameter to a higher value e.g. 64 can help in such conditions.

8.5.3 VIADEV_UD_PREPOST_THRESHOLD

VIADEV_UD_PREPOST_THRESHOLD(9.30) should be less than VIADEV_UD_PREPOST_DEPTH(9.29). If there are many back-to-back multicasts, increasing this threshold can improve performance.

8.5.4 SENDER_WINDOW

Increasing SENDER_WINDOW (9.10) implies buffering more messages at the root till the Acks arrive from the receiver. On a larger cluster, where Acks can potentially arrive late, increasing this window to larger value improves MPI_Bcast thruput.

8.5.5 BCAST_TIME_OUT

BCAST_TIME_OUT (9.1) is dependent on the cluster size. On a larger sizes, this time can be increased.

8.6 Multi-Rail Tuning

Multi-Rail provides the tuning parameters associated with Single-Rail configuration of MVAPICH. In addition, it provides following parameter.

8.6.1 STRIPING_THRESHOLD

Multi-Rail provides an option to enable/disable message striping on available paths by tuning this parameter. For messages of size greater than VIADEV_RENDEZVOUS_THRESHOLD and less than STRIPING_THRESHOLD, only one available path will be used.

For clusters with high number of NUM_HCAS 9.5 and/or NUM_PORTS 9.4, a user may decide not to stripe the data and still go through rendezvous protocol. Please note that STRIPING_THRESHOLD value should at least be equal to VIADEV_RENDEZVOUS_THRESHOLD.

9 MVAPICH Parameters

9.1 BCAST_TIME_OUT

- Class: Compile time
- Location: `mpid/vapi/bcast_info.h`(single_rail config.)
  `mpid/vapi_multirail/bcast_info.h`(multi_rail config.)
- Default: 1 second

This parameter indicates the time duration the root waits for the Ack before retransmitting the message.

9.2 MCST_THRESHOLD

- Class: Compile time
- Location: `src/coll/intra_fns_new.c`
- Default: 8KB

This threshold indicates that MPI_Bcast uses hardware multicast up to MCST_THRESHOLD bytes. This parameter is currently set to 8KB based on experimentation on 32 node systems.

9.3 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 LAZY_MEM_UNREGISTER is not defined. A larger value will lead to more infrequent lazy de-registration. However, the underlying IB layer may have some limit on the total amount of memory a process can register. If you are experiencing memory registration failure, please try decreasing this value.

9.4 NUM_PORTS

- Class: Run time
- Default: 2

This parameter indicates number of ports to be used for communication per adapter on an end node. This parameter has no effect if Multi-Rail configuration is not enabled.

9.5 NUM_HCAS

- Class: Run time
- Default: 1

This parameter indicates number of adapters to be used for communication on an end node. This parameter has no effect if Multi-Rail configuration is not enabled.

9.6 SMPI_MAX_NUMLOCALNODES

- Class: Compile time
- Location: mpid_smpi.h
- Default: 4

This macro has no effect if macro _SMP_ is not defined. It specifies the upper limit of the number of processes MPI supports on a single node. Usually it can be set to be the maximum number of physical processors on an SMP node (if you are not running more than one processes on a single processor).

9.7 SMPI_LENGTH_QUEUE

- Class: Run time
- Default: 4

This has no effect if macro _SMP_ is not defined. It defines the size of shared buffer between every two processes on the same node. A larger value may allow more communication without waiting for flow control. However, a smaller value can save more resources. Note that this variable should be set in MBytes.

9.8 SMP_EAGERSIZE

- Class: Run time
- Default: Architecture dependent (1MB for IA-32)

This has no effect if macro _SMP_ is not defined. It defines the switch point from Eager protocol to Rendezvous protocol for intra-node communication. If macro _SMP_RNDV_ is defined, then for messages larger than SMP_EAGERSIZE, SMP Rendezvous protocol is used, where a message is split into smaller packets, and sent out through shared memory in a pipelining manner. The packet size is the same as the value of SMP_EAGERSIZE. If macro _SMP_RNDV_ is not defined, then IB is used for intra-node Rendezvous protocol. In the latter case the value of this variable should be determined by shared memory communication performance (memory speed, cache size, ...) and IB performance. Note that this variable should be set in KBytes.

9.9 _SMP_RNDV_

- Class: Compile time
- Default: equal to SMP_EAGERSIZE

This has no effect if macro _SMP_ is not defined. If macro _SMP_RNDV_ is defined, then for messages larger than SMP_EAGERSIZE, SMP Rendezvous protocol is used, where a message is split into smaller packets, and sent out through shared memory in a pipelining manner. The packet size is the same as the value of SMP_EAGERSIZE. If macro _SMP_RNDV_ is not defined, then IB is used for intra-node Rendezvous protocol. In the latter case the value of this variable should be determined by shared memory communication performance (memory speed, cache size, ...) and IB performance. Note that this variable should be set in KBytes.

9.10 SENDER_WINDOW

- Class: Compile time
- Location: `mpid/vapi/bcast_info.h`(single_rail config.)
  `mpid/vapi_multirail/bcast_info.h`(multi_rail config.)
- Default: 512

This parameter indicates the maximum number of outstanding MPI_Bcasts allowed at any root. After issuing these many broadcasts, the root blocks if it has not received acks for any of these MPI_Bcasts.

9.11 STRIPING_THRESHOLD

- Class: Run time
- Default: VIADEV_RENDEZVOUS_THRESHOLD

For a class of messages, a user may want to use Rendezvous protocol and not stripe the data across multiple ports/adapters. For messages of size equal and above this value, the data is striped across multiple paths. This value should atleast be equal to the VIADEV_RENDEZVOUS_THRESHOLD. The value of STRIPING THREHOSLD is currently equal to VIADEV_RENDEZVOUS_THRESHOLD. For Optimal performance, this value may need a change depending upon the number of ports and number of adapters in the system.

9.12 VBUF_TOTAL_SIZE

- Class: Compile time
- Location: `vbuf.h`
- Default: Architecture dependent (12 KB for IA-32)

This macro defines the size of each `vbuf`. Basically, vbufs store descriptors and packets used in the underlying communication (send, receive and RDMA). In our current implementation, each eager data packet must fit into one vbuf. Therefore, it also puts an upper limit on the size of the eager data packet. (Please note that eager data payload is even smaller due to the size of the packet header and the descriptor.) However, a large value may lead to more wasted memory.

In the earlier MVAPICH versions, VBUF_TOTAL_SIZE was equal to the VIADEV_RENDEZVOUS_THRESHOLD. From 0.9.5, this restriction has been removed. For a given platform, these two parameters can be independently tuned.

Different presets for this value are available for different sizes of clusters. Use -D_SMALL_CLUSTER, -D_MEDIUM_CLUSTER and -D_LARGE_CLUSTER for cluster sizes 1-64, 64-256, 256 and beyond, respectively.

9.13 VIADEV_DEVICE

- Class: Run time
- Default: First IB device found on the system

Name of the InfiniBand device. e.g. `InfiniHost0`, `InfiniHost1`, `InfiniHost_III_Ex0`.

9.14 VIADEV_RDMA_LIMIT

- Class: Run time
- Default: 2

Upper Limit of the number of outstanding RDMA operations at the InfiniBand level. Effective only when macro VIADEV_HAVE_RDMA_LIMIT is defined. However, it should be set according to the capability of HCAs. For most commonly used HCAs, this option is not required.

9.15 VIADEV_SQ_SIZE

- Class: Run time
- Default: 200

Upper Limit of the number of Send Queue entries at the InfiniBand level. Note that the number of Receive Queue entries are calculated automatically in MVAPICH. This value should be large enough to hold all outstanding send/rdma requests.

9.16 VIADEV_CQ_SIZE

- Class: Run time
- Default: 40000

Upper Limit of the number of Completion Queue entries at the InfiniBand Level. This must be large enough to hold all the outstanding signaled communication operations from all connections.

9.17 VIADEV_NUM_RDMA_BUFFER

- Class: Run time
- Default: Architecture dependent (32 for IA-32)

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 is effective only when macro RDMA_FAST_PATH is defined. The default value is architecture dependent.

Different presets for this value are available for different sizes of clusters. Please use -D_SMALL_CLUSTER, -D_MEDIUM_CLUSTER and -D_LARGE_CLUSTER for cluster sizes 1-64, 64-256, 256 and beyond, respectively.

9.18 VIADEV_MAX_RDMA_SIZE

- Class: Run time
- Default: 1 MB

The upper limit of message size when IB RDMA is used in MVAPICH. Messages such as Rendezvous data will be divided into smaller chunks if their sizes exceed this limit.

9.19 VIADEV_DEFAULT_MTU

- Class: Run time
- Default: MTU1024

The internal MTU used for IB. This parameter should be a string instead of an integer. Valid values are: `MTU256`, `MTU512`, `MTU1024`, `MTU2048`, `MTU4096`.

9.20 VIADEV_MAX_FAST_EAGER_SIZE

- Class: Fixed
- Default: 255

This is used to specify the maximum size of the messages which are sent using `header caching`. Please note that this value cannot exceed 255 in the current implementation.

9.21 VIADEV_DEFAULT_MAX_SG_LIST

- Class: Run time
- Default: 20

This specifies the maximum number of gather/scatter entries support for each queue pair. Currently, InfiniBand communication uses only one gather/scatter entry. However, this parameter also affects the maximum size of data that can be sent using "inline". Larger messages can be sent through inline with larger VIADEV_DEFAULT_MAX_SG_LIST value.

9.22 VIADEV_RENDEZVOUS_THRESHOLD

- Class: Run time
- Default: Architecture dependent (12KB for IA-32)

This specifies the switch point between eager and rendezvous protocol in MVAPICH. If this value is increased more than the VBUF_TOTAL_SIZE, then multiple packets will be sent over the `eager` path using copy-based scheme.

In the earlier MVAPICH versions, VBUF_TOTAL_SIZE was equal to the VIADEV_RENDEZVOUS_THRESHOLD. From 0.9.5, this restriction has been removed. For a given platform, these two parameters can be independently tuned.

9.23 VIADEV_VBUF_POOL_SIZE

- Class: Run time
- Default: 5000

The number of vbufs in the initial pool. This pool is shared among all the connections. A large value will lead to more initial memory usage. However, a small value may lead to memory allocation at run time and degrade performance.

9.24 VIADEV_VBUF_SECONDARY_POOL_SIZE

- Class: Run time
- Default: 500

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. A large value may lead to waste of memory. But if the value is too small, memory allocation may be frequent during run time and degrade performance.

9.25 VIADEV_INITIAL_PREPOST_DEPTH

- Class: Run time
- Default: 5

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 VIADEV_PREPOST_DEPTH.

9.26 VIADEV_PREPOST_DEPTH

- Class: Run time
- Default: 64

This defines the number of buffers pre-posted for each connection to handle send/receive operations. If RDMA_FAST_PATH is enabled, this macro can be set to a small value (such as 32 or 64). This number should not be set to a very large value for large systems. Otherwise the memory consumption will be large.

9.27 VIADEV_CREDIT_NOTIFY_THRESHOLD

- Class: Run time
- Default: 5

Flow control information is usually sent via piggybacking with other messages. These two parameters are used to determine when to send explicit flow control update messages.

9.28 VIADEV_DYNAMIC_CREDIT_THRESHOLD

- Class: Run time
- Default: 10

Flow control information is usually sent via piggybacking with other messages. These two parameters are used to determine when to send explicit flow control update messages.

9.29 VIADEV_UD_PREPOST_DEPTH

- Class: Compile time
- Location: `mpid/vapi/bcast_info.h` (single_rail config.)
  `mpid/vapi_multirail/bcast_info.h` (multi_rail config.)

- Default: 32

This parameter indicates the total number of buffers preposted for UD messages.

9.30 VIADEV_UD_PREPOST_THRESHOLD

- Class: Compile time
- Location: `mpid/vapi/bcast_info.h`(single_rail config.)
  `mpid/vapi_multirail/bcast_info.h`(multi_rail config.)
- Default: 16

This parameter defines a low water mark for the number of buffers posted for UD messages. If this water mark is reached, the posting of buffers will begin until the number of buffers reaches the value defined in VIADEV_UD_PREPOST_DEPTH.