MPI: Overview, Performance Optimizations and Tuning

A Tutorial at HPC Advisory Council Conference, Lugano 2012

by

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

- Trends in Designing Petaflop and Exaflop Systems
- Overview of Programming Models and MPI
- MPI Features and How to Use These
- Overview of MVAPICH2 MPI Stack
- Basic Performance Optimization and Tuning of MVAPICH2
Expected to have an ExaFlop system in 2019-2020!
## Trends for Commodity Computing Clusters in the Top 500 List (http://www.top500.org)

<table>
<thead>
<tr>
<th>Date</th>
<th>Number of Clusters</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nov. 1996: 0/500 (0%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jun. 1997: 1/500 (0.2%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nov. 1997: 1/500 (0.2%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jun. 1998: 1/500 (0.2%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nov. 1998: 2/500 (0.4%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jun. 1999: 6/500 (1.2%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nov. 1999: 7/500 (1.4%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jun. 2000: 11/500 (2.2%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nov. 2000: 28/500 (5.6%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jun. 2001: 33/500 (6.6%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nov. 2001: 43/500 (8.6%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jun. 2002: 80/500 (16%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nov. 2002: 93/500 (18.6%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jun. 2003: 149/500 (29.8%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nov. 2003: 208/500 (41.6%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jun. 2004: 291/500 (58.2%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nov. 2004: 294/500 (58.8%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jun. 2005: 304/500 (60.8%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nov. 2005: 360/500 (72.0%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jun. 2006: 364/500 (72.8%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nov. 2006: 361/500 (72.2%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jun. 2007: 373/500 (74.6%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nov. 2007: 406/500 (81.2%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jun. 2008: 400/500 (80.0%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nov. 2008: 410/500 (82.0%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jun. 2009: 410/500 (82.0%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nov. 2009: 417/500 (83.4%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jun. 2010: 424/500 (84.8%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nov. 2010: 415/500 (83%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jun. 2011: 411/500 (82.2%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nov. 2011: 410/500 (82.0%)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Large-scale InfiniBand Installations

- 209 IB Clusters (41.8%) in the November‘11 Top500 list
  ([http://www.top500.org](http://www.top500.org))
- Installations in the Top 30 (13 systems):

<table>
<thead>
<tr>
<th>Installation</th>
<th>Cores</th>
<th>Location</th>
<th>Ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td>120,640 cores (Nebulae) in China</td>
<td>120,640</td>
<td>China</td>
<td>4th</td>
</tr>
<tr>
<td>73,278 cores (Tsubame-2.0) in Japan</td>
<td>73,278</td>
<td>Japan</td>
<td>5th</td>
</tr>
<tr>
<td>111,104 cores (Pleiades) at NASA Ames</td>
<td>111,104</td>
<td>NASA Ames</td>
<td>7th</td>
</tr>
<tr>
<td>138,368 cores (Tera-100) at France</td>
<td>138,368</td>
<td>France</td>
<td>9th</td>
</tr>
<tr>
<td>122,400 cores (RoadRunner) at LANL</td>
<td>122,400</td>
<td>LANL</td>
<td>10th</td>
</tr>
<tr>
<td>137,200 cores (Sunway Blue Light) in China</td>
<td>137,200</td>
<td>China</td>
<td>14th</td>
</tr>
<tr>
<td>46,208 cores (Zin) at LLNL</td>
<td>46,208</td>
<td>LLNL</td>
<td>15th</td>
</tr>
<tr>
<td>33,072 cores (Lomonosov) in Russia</td>
<td>33,072</td>
<td>Russia</td>
<td>18th</td>
</tr>
</tbody>
</table>

More are getting installed!
Presentation Overview

- Trends in Designing Petaflop and Exaflop Systems
- **Overview of Programming Models and MPI**
  - MPI Features and How to Use These
  - Overview of MVAPICH2 MPI Stack
  - Basic Performance Optimization and Tuning of MVAPICH2
Parallel Systems - History and Overview

- Parallel system offers greater compute and memory capacity than a serial system
  - Tackle problems that are too big to fit in one computer

- Different types of systems
  - Uniform Shared Memory (bus based)
    - Many way symmetric multi-processor machines: SGI, Sun, ...
  - Non uniform Shared Memory (NUMA)
    - CCNUMA machines: Cray CX1000, AMD Magny Cours, Intel Westmere
  - Distributed Memory Machines
    - Commodity clusters, Blue Gene, Cray XT5

- Similarly, there are different types of programming models
  - Shared memory, Distributed memory ...
Parallel Programming Models Overview

- Programming models provide abstract machine models
- Models can be mapped on different types of systems
  - e.g. Distributed Shared Memory (DSM), MPI within a node, etc.
- In this presentation series, we concentrate on MPI first and then focus on PGAS
Designing Communication Libraries for Multi-Petaflop and Exaflop Systems: Challenges

Applications/Libraries

Programming Models
Message Passing Interface (MPI), Sockets and PGAS (UPC, Global Arrays)

Library or Runtime for Programming Models

Point-to-point Communication
Collective Communication
Synchronization & Locks
I/O & File Systems
QoS
Fault Tolerance

Networking Technologies
(InfiniBand, 1/10/40GigE, RNICs & Intelligent NICs)

Commodity Computing System Architectures
(single, dual, quad, ..)
Multi/Many-core architecture and Accelerators

HPC Advisory Council, Lugano, Switzerland '12
MPI Overview and History

• Message Passing Library standardized by MPI Forum
  – C, C++ and Fortran

• Goal: portable, efficient and flexible standard for writing parallel applications

• Not IEEE or ISO standard, but widely considered “industry standard” for HPC application

• Evolution of MPI
  – MPI-1: 1994
  – MPI-2: 1996
  – MPI-3: on-going effort (2008 – current)
What does MPI do?

• Primarily intended for distributed memory machines

• P2 needs value of A
  – **MPI-1:** P1 will have to send a message to P2 with value of A using MPI_Send
  – **MPI-2:** P2 can get value of A directly using MPI_Get

• P1, P2, P3 need sum of A+B+C
  – MPI_Allreduce with SUM op
  – Multi-way communication
Presentation Overview

- Trends in Designing Petaflop and Exaflop Systems
- Overview of Programming Models and MPI
- **MPI Features and Their Usage**
  - Overview of MVAPICH2 MPI Stack
  - Basic Performance Optimization and Tuning of MVAPICH2
MPI Features and Their Usage

- Point-to-point Two-sided Communication
- Collective Communication
- One-sided Communication
- Job Startup
- Parallel I/O
Types of Point-to-Point Communication

• Synchronous (MPI_Ssend)
  – Sender process blocks on send until receiver arrives

• Blocking Send / Receive (MPI_Send, MPI_Recv)
  – Block until send buffer can be re-used
  – Block until receive buffer is ready to read

• Non-blocking Send / Receive (MPI_Isend, MPI_Irecv)
  – Start send and receive, but don’t wait until complete

• Others: buffered send, sendrecv, ready send
  – Not used very frequently
Message Matching

• How does MPI know which send is for which receive?

• Programmer (i.e. you!) need to provide this information
  – Sender side: tag, destination rank and communicator
  – Communicator is a subset of the entire set of MPI processes
  – MPI_COMM_WORLD represents all MPI processes
  – Receiver side: tag, source rank and communicator
  – The triples: tag, rank and communicator must match

• Some special, pre-defined values: MPI_ANY_TAG, MPI_ANY_RANK
Buffering

- MPI library has internal “system” buffers
  - Optimize throughput (do not wait for receiver)
  - Opaque to programmer
  - Finite Resource

- Blocking send may copy to sender system buffer and return

Courtesy: https://computing.llnl.gov/tutorials/mpi/
Blocking vs. Non-blocking

• Blocking
  – Send returns when safe to re-use buffer (maybe in system buffer)
  – Receive only returns when data is fully received

• Non-blocking
  – Returns immediately (data may or may not be buffered)
  – Simply request MPI library to transfer the data
  – Need to “wait” on handle returned by call
  – Benefit is that computation and communication can be overlapped
MPI Features and Their Usage

• Point-to-point Two-sided Communication

• Collective Communication

• One-sided Communication

• Job Startup

• Parallel I/O
Types of Collective Communication

• Synchronization
  – Processes wait until all of them have reached a certain point

• Data Movement
  – Broadcast, Scatter, All-to-all ...

• Collective Computation
  – Allreduce with min, max, multiply, sum ... on data

• Considerations
  – Blocking, no tag required, only with pre-defined datatypes
  – MPI-3 considering non-blocking versions (discussing tomorrow)
**Example Collective Operation: Scatter**

- Using Scatter, an array can be distributed to multiple processes.

---

**Table: Example Array Distribution**

<table>
<thead>
<tr>
<th>Rank 0</th>
<th>Rank 1</th>
<th>Rank 2</th>
<th>Rank 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>5.0</td>
<td>9.0</td>
<td>13.0</td>
</tr>
<tr>
<td>2.0</td>
<td>6.0</td>
<td>10.0</td>
<td>14.0</td>
</tr>
<tr>
<td>3.0</td>
<td>7.0</td>
<td>11.0</td>
<td>15.0</td>
</tr>
<tr>
<td>4.0</td>
<td>8.0</td>
<td>12.0</td>
<td>16.0</td>
</tr>
</tbody>
</table>
MPI Features and Their Usage

- Point-to-point Two-sided Communication
- Collective Communication
- One-sided Communication
- Job Startup
- Parallel I/O
MPI One-sided communication: Benefits and The Model

- Easy to express irregular pattern of communication
- Decouple data transfer with synchronization
- Potentially better computation-communication overlap

- Each process can contribute part of its memory to form a larger “window” of global memory, through a collective operation
MPI One Sided Taxonomy

- Provides active and passive synchronization modes
- Different modes suite different communication patterns
MPI Features and Their Usage

- Point-to-point Two-sided Communication
- Collective Communication
- One-sided Communication
- Job Startup
- Parallel I/O
Launching MPI jobs

- MPI process managers provide support to launch jobs
- “mpiexec” is a utility to launch jobs
- Example usage:
  - mpiexec -np 2 -machinefile mf ./a.out
- Supports SPMD (single program multiple data) model along with MPMD (multiple program multiple data)
- Different resource management systems / MPI stacks may do things slightly differently
  - SLURM, PBS, Torque
  - mpirun_rsh (fastest launcher for MVAPICH and MVAPICH2), hydra (MPICH2) and ORTE (Open MPI)
- Launch time should not increase with number of processes
MPI Features and Their Usage

- Point-to-point Two-sided Communication
- Collective Communication
- One-sided Communication
- Job Startup
- Parallel I/O
Parallel I/O

• Parallel I/O very important for scientific applications

• Parallel file systems offer high bandwidth access to large volumes of data
  – PVFS (parallel virtual file system)
  – Lustre, GPFS ...

• MPI applications can use MPI-IO layer for collective I/O
  – Using MPI-IO optimal I/O access patterns are used to read data from disks
  – Fast communication network then helps re-arrange data in order desired by end application
Collective I/O in MPI

- Critical optimization in MPI I/O
- All processes must call collective function
- Basic idea: build large blocks from small requests so requests will be large from disk point of view
  - Particularly effective when accesses by different processes are non-contiguous and interleaved

![Diagram showing small individual requests merged into large collective access]

Presentation Overview

• Trends in Designing Petaflop and Exaflop Systems
• Overview of Programming Models and MPI
• MPI Features and Their Usage
• **Overview of MVAPICH2 MPI Stack**
• Basic Performance Optimization and Tuning of MVAPICH2
Designing MPI Using InfiniBand Features

Major Components in MPI

- Protocol Mapping
- Flow Control
- Communication Progress
- Multi-rail Support
- Checkpoint Restart
- Process Migration
- Buffer Management
- Connection Management
- Collective Communication
- One-sided Active/Passive
- Reliability and Resiliency

Many different design choices

Optimal design choices
- Performance
- Scalability
- Fault-Tolerance & Resiliency
- Power-Aware

InfiniBand Features

- Send / Receive
- Reliable Connection
- eXtended Reliable Connection (XRC)
- QoS (SL and VL)
- Static Rate Control
- Shared Receive Queues (SRQ)
- RDMA Operations
- Unreliable Datagram
- Atomic Operations
- End-to-End Flow Control
- Multi-Path LMC
- Multicast
MVAPICH/MVAPICH2 Software

- High Performance open-source MPI Library for InfiniBand, 10Gig/iWARP and RDMA over Converged Enhanced Ethernet (RoCE)
  - MVAPICH (MPI-1) and MVAPICH2 (MPI-2.2), Available since 2002
  - Used by more than 1,850 organizations (HPC Centers, Industry and Universities) in 65 countries
  - More than 99,000 downloads from OSU site directly
  - Empowering many TOP500 clusters
    - 5th ranked 73,278-core cluster (Tsubame 2.0) at Tokyo Institute of Technology
    - 7th ranked 111,104-core cluster (Pleiades) at NASA
    - 25th ranked 62,976-core cluster (Ranger) at TACC
    - and many others
  - Available with software stacks of many InfiniBand, High-speed Ethernet and server vendors including Open Fabrics Enterprise Distribution (OFED) and Linux Distros (RedHat and SuSE)
    - http://mvapich.cse.ohio-state.edu
- Partner in the upcoming U.S. NSF-TACC Stampede (10-15 PFlop) System
MVAPICH2 Architecture (Latest Release 1.8a2)

Major Computing Platforms: IA-32, EM64T, Nehalem, Westmere, Sandybridge, Opteron, Magny, ..
Performance Optimization and Tuning with MVAPICH2

MVAPICH and MVAPICH2 Libraries

Major Components and Optimizations?

Performance?

Scalability?

Fault Tolerance?

Usage Guidelines & Flexibility?

Strongly encouraged to refer to the User Guide
http://mvapich.cse.ohio-state.edu/support/
Major Components and Optimizations

- Pt-to-pt Inter-node Communication
  - Eager and Rendezvous Protocols
  - RDMA Fast Path
- Pt-to-pt Intra-node Communication
- Collective Communication
- One-sided Communication
- Multi-rail Support
- Scalability Issues (SRQ, UD, XRC and Hybrid)
- QoS and 3D Torus Topology
- Fault Tolerance
Pt-Pt Internode Communication

• EAGER (buffered, used for small messages)
  – RDMA Fast Path
  – Send/Recv

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

- RDMA Fast Path has advantages for smaller message range
- Switching Eager to Rendezvous transfer
  - MV2_IBA_EAGER_THRESHOLD
- Default: Architecture dependent on common platforms
- Can be modified by users to get smooth performance across message sizes
Point-to-Point - Usage

- `mpirun_rsh -np 1024 -f hostfile a.out`
  - Fast Path channel setup with 16 buffers [Default: 16 (large cluster), 32 (small clusters)]
- `mpirun_rsh -np 1024 -f hostfile MV2_NUM_RDMA_BUFFERS=32 a.out`
  - Fast Path channel setup with 32 buffers
- `mpirun_rsh -np 1024 -f hostfile MV2_USE_RDMA_FASTPATH=0 a.out`
  - Disables RDMA Fast Path [Default enabled]
- `mpirun_rsh -np 1024 -f hostfile a.out`
  - Default large messages go over RPUT (RDMA write based) rendezvous protocol
- `mpirun_rsh -np 1024 -f hostfile MV2_RNDV_PROTOCOL=RGET a.out`
  - Large messages go over RGET (RDMA Read based) rendezvous protocol
- `mpirun_rsh -np 1024 -f hostfile MV2_RNDV_PROTOCOL=R3 a.out`
  - Large messages go over R3 (Send/Recv based) rendezvous protocol
Major Components and Optimizations

- Pt-to-pt Inter-node Communication
- Pt-to-pt Intra-node Communication
  - Basic shared memory and LiMIC2
  - CPU Core Mapping and Impact
- Collective Communication
- One-sided Communication
- Multi-rail Support
- Scalability Issues (SRQ, UD, XRC and Hybrid)
- QoS and 3D Torus Topology
- Fault Tolerance
Intra-node Communication Support in MVAPICH2

- Shared-Memory based two-copy intra-node communication
  - Copy from the sender’s user buffer to the shared buffer
  - Copy from the shared buffer to the receiver’s user buffer
  - SMP_EAGERSIZE is used to set the threshold for Eager protocol
    - $ mpirun_rsh -np 4 -hostfile hosts SMP_EAGERSIZE=16k ./a.out
    - This parameter can be adjusted to get best performance

- LiMIC2 on modern multi-core platforms
  - Motivation: Eliminate one memory copy
  - Kernel-level module for achieving efficient intra-node communication
  - LIMIC2 is used for rendezvous protocol message size
  - Use ‘--with-limic2’ option at configure step
    - $ ./configure --with-limic2
    - MV2_SMP_USE_LIMIC2 is used to enable/disable LIMIC2
      - Default: 1, if configured with the --with-limic2 flag
      - To disable: $ mpirun_rsh -np 4 -hostfile hosts MV2_SMP_USE_LIMIC2=0 ./a.out
MVAPICH2 Two-Sided Intra-Node Performance
(Shared memory and Kernel-based Zero-copy Support)

Latency

- Intra-Socket
- Inter-Socket

Latency (us)

- 0.19 us
- 0.45 us

Message Size (Bytes)

0 1 2 4 8 16 32 64 128 256 512 1K

Bandwidth

- intra-Socket-LiMIC
- intra-Socket-Shmem
- inter-Socket-LiMIC
- inter-Socket-Shmem

Bandwidth (MB/s)

0 2000 4000 6000 8000 10000 12000 14000 16000 18000 20000

Message Size (Bytes)

1 4 16 64 256 1K 4K 16K 64K 256K 1M 4M

Bi-Directional Bandwidth

- intra-Socket-LiMIC
- intra-Socket-Shmem
- inter-Socket-LiMIC
- inter-Socket-Shmem

Bandwidth (MB/s)

0 5000 10000 15000 20000

Message Size (Bytes)

1 4 16 64 256 1K 4K 16K 64K 256K 1M 4M

Latest MVAPICH2 1.8a2
Intel Westmere

MVAPICH2 Two-Sided Intra-Node Performance
(Shared memory and Kernel-based Zero-copy Support)
**Latency Impact of Core selection**

<table>
<thead>
<tr>
<th>Core Pair</th>
<th>0-byte</th>
<th>8K-byte</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-2</td>
<td>0.17us</td>
<td>1.83us</td>
<td>Same socket, shared L3, best performance</td>
</tr>
<tr>
<td>0-1</td>
<td>0.17us</td>
<td>1.87us</td>
<td>Same socket, shared L3, but core0 handles interrupts</td>
</tr>
<tr>
<td>1-5</td>
<td>0.41us</td>
<td>3.16us</td>
<td>Different sockets, does not share L3</td>
</tr>
<tr>
<td>0-4</td>
<td>0.42us</td>
<td>3.17us</td>
<td>Different sockets, does not share L3, core0 handles interrupts</td>
</tr>
</tbody>
</table>

**Scatter Mapping**

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

**Bunch Mapping**

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

Intel Westmere (8 cores/node)

More core selection schemes will be available in future MVAPICH2 releases
NUMA Interactions

- Different cores in a NUMA platform have different communication costs
Latency impact on NUMA platforms

- Cores in NUMA Node 0 (closest to network card) have lowest latency
- Cores in NUMA Node 3 (farthest from network card) have highest latency
- Cores in NUMA Node 2 (one hop from network card) have in-between latency

24-core AMD Magny Cours (dual-socket) node with a Mellanox ConnectX-2 QDR IB HCA
Major Components and Optimizations

- Pt-to-pt Inter-node Communication
- Pt-to-pt Intra-node Communication
- Collective Communication
  - Pt-to-pt and Shared-Memory-Aware Schemes
- One-sided Communication
- Multi-rail Support
- Scalability Issues (SRQ, UD, XRC and Hybrid)
- QoS and 3D Torus Topology
- Fault Tolerance
Collective Communication Overview

Run-time flag to control all shared-memory based collectives:
MV2_USE_SHMEM_COLL (Default: ON)

Run-time flags to select different collective operations

**MPI_Allreduce:**
MV2_ALLREDUCE_SHMEM_MSG (Default: 32KB )
MV2_ALLREDUCE_2LEVEL_MSG (Default: 256KB)

**MPI_Reduce:**
MV2_REDUCE_SHMEM_MSG (Default: Up to 8KB)
MV2_REDUCE_2LEVEL_MSG (Default: Up to 16KB)

**MPI_Bcast:**
MV2_USE_KNOMIAL_2LEVEL_BCAST
MV2_KNOMIAL_INTRA_NODE_FACTOR (Default: 4)
MV2_KNOMIAL_INTER_NODE_FACTOR (Default: 4)
Shared-memory Aware Collectives

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

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

HPC Advisory Council, Lugano, Switzerland '12
Major Components and Optimizations

- Pt-to-pt Inter-node Communication
- Pt-to-pt Intra-node Communication
- Collective Communication
- One-sided Communication
  - Inter-node and Intra-node
- Multi-rail Support
- Scalability Issues (SRQ, UD, XRC and Hybrid)
- QoS and 3D Torus Topology
- Fault Tolerance
MPI One-sided Communication

- Specified by the MPI-2 standard
- Data movement operations
  - MPI_Put
  - MPI_Get
  - MPI_Accumulate
- Synchronization operations
  - MPI_Lock/MPI_Unlock
  - MPI_Win_fence
  - MPI_Win_post, MPI_Win_start, MPI_Win_complete, MPI_Win_wait
- Design Challenges
  - Improved Communication Performance (two-sided vs. direct one-sided)
  - Reduced Synchronization Overhead
  - Better overlap capability
Inter-node Overlap Performance

![Graph showing Inter-node Overlap Performance](image)

- **Overlap Percentage (%)**
  - Y-axis range: 0 to 100
- **Message Size (Bytes)**
  - X-axis values: 8K, 16K, 32K, 64K, 128K, 256K, 512K, 1M

Legend:
- **2-sided**
- **1-sided over 2-sided**
- **1-sided over Direct RDMA Design**

HPC Advisory Council, Lugano, Switzerland '11

HPC Advisory Council, Lugano, Switzerland '12
Intra-node One-sided Communication

- Shared memory techniques based on two-sided communication leads to two copies
- One copy methods
  - Shared memory based windows
  - Kernel based (LiMIC2)
Experimental Results with One-Sided Design and Overlap

- Experiments on TACC Ranger cluster – 64x64x64 data grid per process – 25 iterations – 32KB messages
- On 4K processes – 11% with Async-2sided and 12% with Async-1sided (RMA)
- On 8K processes – 6% with Async-2sided and 10% with Async-1sided (RMA)

Joint work with OSU, SDSC and TACC
Gordon-Bell Finalist Paper for Supercomputing 2010
Major Components and Optimizations

- Pt-to-pt Inter-node Communication
- Pt-to-pt Intra-node Communication
- Collective Communication
- One-sided Communication
- Multi-rail Support
- Scalability Issues (SRQ, UD, XRC and Hybrid)
- QoS and 3D Torus Topology
- Fault Tolerance
MVAPICH2 Multi-Rail Design

- **RR** – Round Robin
- **FM** – Fixed Mapping
  - **B** – Bunch (0:0:1:1)
  - **S** – Scatter (0:1:0:1)
  - **U** – User Defined
Multi-Pair Performance on Multi-Rail Clusters

- System: two 24-core Magny nodes with dual Mellanox ConnectX QDR adapters
- Numbers taken for 6 communicating pairs with OSU Multi-Pair bandwidth and messaging rate benchmark
- Multi-rail design significantly improves bandwidth
- Advanced multi-rail schemes significantly improves messaging rate
Major Components and Optimizations

- Pt-to-pt Inter-node Communication
- Pt-to-pt Intra-node Communication
- Collective Communication
- One-sided Communication
- Multi-rail Support
- Scalability Issues (SRQ, UD, XRC and Hybrid)
- QoS and 3D Torus Topology
- Fault Tolerance
**Shared Receive Queue (SRQ)**

- SRQ is a hardware mechanism for a process to share receive resources (memory) across multiple connections
  - Introduced in specification v1.2

- \( 0 < Q << P \times ((M \times N) - 1) \)
• SRQ consumes only 1/10th compared to RDMA for 16,000 processes
• Send/Recv exhausts the buffer pool after 1000 processes; consumes 2X memory as SRQ for 16,000 processes

**UD vs. RC: Performance and Scalability (SMG2000)**

- **Large number of peers per process (992 at maximum)**
  - UD reduces HCA QP cache thrashing

---

**Memory Usage (MB/process)**

<table>
<thead>
<tr>
<th></th>
<th>RC (MVAPICH 0.9.8)</th>
<th>UD Design</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Conn.</td>
<td>Buffers</td>
</tr>
<tr>
<td>512</td>
<td>22.9</td>
<td>65.0</td>
</tr>
<tr>
<td>1024</td>
<td>29.5</td>
<td>65.0</td>
</tr>
<tr>
<td>2048</td>
<td>42.4</td>
<td>65.0</td>
</tr>
<tr>
<td>4096</td>
<td>66.7</td>
<td>65.0</td>
</tr>
</tbody>
</table>

---

eXtended Reliable Connection (XRC)

- Each QP takes at least one page of memory
  - Connections between all processes is very costly for RC
- **New** IB Transport added: eXtended Reliable Connection
  - Allows connections between nodes instead of processes

\[ M = \# \text{ of processes/node} \]
\[ N = \# \text{ of nodes} \]

\[ M^2 \times (N - 1) \text{ connections/node} \]

\[ M \times (N - 1) \text{ connections/node} \]
Both UD and RC/XRC have benefits

Evaluate characteristics of all of them and use two sets of transports in the same application – get the best of both

Available in MVAPICH (since 1.1), available in MVAPICH2 (since 1.7)

M. Koop, T. Jones and D. K. Panda, “MVAPICH-Aptus: Scalable High-Performance Multi-Transport MPI over InfiniBand,” IPDPS ‘08
Major Components and Optimizations

- Pt-to-pt Inter-node Communication
- Pt-to-pt Intra-node Communication
- Collective Communication
- One-sided Communication
- Multi-rail Support
- Scalability Issues (SRQ, UD, XRC and Hybrid)
- QoS and 3D Torus Topology
- Fault Tolerance
QoS in MVAPICH2

• Users can make use of multiple SLs (and in-turn VLs)

• Pre-requisite
  – Setup network with desired number of VLs using OpenSM
  – One-to-One mapping of SL to VL

• Multiple VLs also used to break deadlocks in IB Torus networks
  – New routing algorithm in OpenSM (torus-2QoS)
    • Uses multiple VLs to route messages in conflicting paths
  – MPI needs to choose correct SL for communication
    • Query OpenSM and retrieve "Path Record" for source destination LID pair
    • Use the SL specified in Path Record for communication
  – Enabled by default at configure time (--enable-3dtorus-support)

• Environment variables
  – MV2_USE_QOS – Distributes traffic across 8 VLs (default)
  – MV2_NUM_SLS – Defines number of SLs to use
  – MV2_3DTORUS_SUPPORT – Enable querying OpenSM for Path Record
MPI Performance with Multiple VLs & Inter-Job QoS

- Micro-benchmarks use 8 communicating pairs
  - One QP per pair
- Performance improvement over One VL case
  - Alltoall – 20%
  - Application – 11%
- 12% performance improvement with Inter-Job QoS

Major Components and Optimizations

- Pt-to-pt Inter-node Communication
- Pt-to-pt Intra-node Communication
- Collective Communication
- One-sided Communication
- Multi-rail Support
- Scalability Issues (SRQ, UD, XRC and Hybrid)
- QoS and 3D Torus Topology

- Fault Tolerance
  - Network-Level
  - Process-Level (Checkpoint-Restart and Migration)
Fault Tolerance and Multiple Solutions

- Component failures are common in large-scale clusters
- Imposes need for reliability and fault tolerance
- Network-Level Fault Tolerance/Resiliency in MVAPICH/MVAPICH2
  - Automatic Path Migration (APM)
  - Mem-to-Mem Reliability
  - Network Level Fault-Resilience (NFR)
- Process-Level Fault Tolerance in MVAPICH2
  - BLCR-based systems-level Checkpoint-Restart (CR)
  - Enhancing CR Performance with I/O Aggregation
  - Fault-Tolerant Backplane (FTB) over InfiniBand
  - Fault-Prediction with FTB-IPMI
  - Pro-active Process Migration with Job-Suspend and Resume
Enhancing Checkpoint-Restart Performance on Multicore Platforms

• Process-level Fault Tolerance
  – User-transparent, system-level checkpointing
  – Based on BLCR from LBNL to take coordinated checkpoints of entire program, including front end and individual processes
• Available for the last several years with MVAPICH2 and is being used by many organizations
• Checkpoint time is dominated by writing the files to storage
• Multi-core systems are emerging
  – 8/16-cores per node
  – a lot of data needs to be written
  – affects scalability
• Can we reduce checkpoint time with I/O aggregation of short messages?
Checkpoint Writing Performance w/o Write-Aggregation

( CRFS: Write-Aggregation module within MVAPICH2)

Process Migration for MVAPICH2

- Checkpoint/Restart is not scalable for very large systems
- Job-Pause and Process Migration framework can deliver pro-active fault-tolerance
  - also allows processes to be relocated on large-scale system for compaction
  - Process migration:
    - Fault-prediction generated by FTB-IPMI / User triggers migration
      (A) Process memory snapshot obtained at source node (Aggregated to Buffer Pool)
      (B) Snapshots transferred to target node in a pipelined manner using RDMA
      (C) Application processes resume execution at target node (from the in-memory snapshot)
Process Migration vs. Checkpoint/Restart

CR with Write-Agggregation / Process Migration Usage

- Define environment variables for
  - MV2_CKPT_USE_AGGREGATION (to enable CR-Aggregation)
  - MV2_CKPT_AGGREGATION_BUFPOOL_SIZE
  - MV2_CKPT_AGGREGATION_CHUNK_SIZE

- Sample usage
  - mpirun_rsh -np 64 -hostfile ./hosts
    MV2_CKPT_USE_AGGREGATION=1
    MV2_CKPT_AGGREGATION_BUFPOOL_SIZE=16m
    MV2_CKPT_AGGREGATION_CHUNK_SIZE=4m
    MV2_CKPT_FILE=/mnt/lustre/ckpt1 ./a.out

- Add ‘--sparehosts <spare_host_file>’ in case of process migration
- Trigger checkpoints with ‘<install_dir>/bin/mv2_checkpoint’ tool
- Trigger migration with ‘<install_dir>/bin/mv2_trigger’ tool
Conclusions

- MPI is a dominant programming model for HPC Systems
- MVAPICH2 is a popular MPI stack and offers many performance optimizations and tuning
- Most default settings are geared towards generic applications
  - Default values are chosen after careful study across a wide variety of applications
- Default settings may be further tuned based on
  - Application needs
  - Specific system characteristics
  - Other needs (like memory limits, etc.)
Tomorrow’s (March 14th) Presentation

- Emerging MPI features geared towards Multi-PetaFlop and ExaFlop systems
  - Support for NVIDIA GPUs and Accelerators
  - Advanced Collectives
    - Offload
    - Topology-aware
    - Power-aware
  - Designing Support for PGAS
    - UPC
    - OpenShmem
- Upcoming MPI-3 Standard and Features
Programming models like MPI have taken advantage of RDMA to design high-performance and scalable libraries. Can similar things be done for other middleware being used for processing Big Data? Present challenges and provide initial designs for:

- Memcached
- HBase and HDFS in Hadoop framework
Web Pointers

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

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

panda@cse.ohio-state.edu