

# Implementation of Programming models on Systems with Modern Networks

Ketaki Koppal

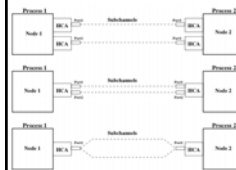
# Papers

- J. Liu, A. Vishnu and D. K. Panda, **Building Multirail InfiniBand Clusters: MPI-Level Design and Performance Evaluation.**
- A. Vishnu, M. Koop, A. Moody, A. Mamidala, S. Narravula and D. K. Panda, **Hot-Spot Avoidance With Multi-Pathing Over InfiniBand: An MPI Perspective,**
- Q. Gao, W. Yu, W. Huang and D. K. Panda, **Application-Transparent Checkpoint/Restart for MPI Programs over InfiniBand**
- S. Narravula, A. R. Mamidala, A. Vishnu, G. Santhanaraman, and D. K. Panda, **High Performance MPI over iWARP: Early Experiences**

# Introduction

- Performance Bottleneck - network bandwidth
- Multirail Networks
  - Multiplexing
    - messages sent through rails selected in round robin fashion
    - Used for send/receive operations and RDMA with small data
  - Striping
    - Messages divided into chunks and sent simultaneously on several rails
    - Used for large RDMA messages

# Multirail Network Configurations



- Virtual Subchannel Abstraction
- Multiple HCAs
  - Aggregated bandwidth of all HCAs without modifying application
  - Possible Bandwidth bottleneck in I/O buses can be avoided
- Multiple Ports in single HCA
  - Local I/O bus can be performance bottleneck - overridden by using future HCAs that support PCI-X DDR or QDR interfaces
- Single Port with LID Mask Control (LMC)
  - Virtual multirail networks
  - Each port has a local identifier (LID), multiple logical LIDs associated with a single physical port.
  - Advantage - if performance bottleneck is inside network, communication performance can be improved by utilizing multiple paths.
  - Disadvantage - if port link bandwidth or local I/O bus is the performance bottleneck this approach cannot bring performance benefit.

# Basic Architecture

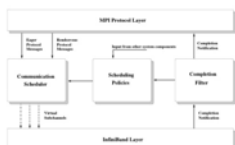


Figure 2. Basic Architecture of Multirail MPI Design

- Communication Scheduler - accepts protocol messages and stripes/multiplexes them across multiple virtual subchannels
- Scheduling Policies
  - Static/Dynamic
  - Multiplexing/Striping
- Completion Filter - gather information based on completion notifications and use it as input to adjust dynamic scheduling policies.

# Adaptive Striping

- Weighted striping scheme
- Constantly monitor the progress of different stripes in each subchannel and exploit feedback information to adjust weights
- Load balancing and minimum message delivering time



Figure 4. Feedback Loop in Adaptive Striping

## Design Issues

- Handling Multiple HCAs
  - Situations that must be handled differently for multiple HCAs -
    - Completion Queue
      - Multiple connections can be associated with a single CQ if all the connections are from a single HCA.
      - Higher overhead due to extra polling of CQs
    - Buffer Registration
      - Buffer needs to be registered with each HCA
      - Solutions -
        1. Use of registration cache
        2. Register part of buffer with each HCA

## Design Issues

- Out-of-Order Message Processing
  - No ordering guarantee for multiple subchannels of same virtual channel
  - Packet Sequence Number (PSN), Expected Sequence Number (ESN)
  - Out-of-order queue - enqueue, dequeue and search
- RDMA Completion Notification
  - Multiple completion notifications generated for a single message at sender side
  - Multiple Rendezvous finish control messages for multiple subchannels at receiver side

## Experimental Testbed

- Components -
  - Cluster of 8 SuperMicro SUPER X5DL8-GG nodes with ServerWorks GC LE chipsets.
  - Each node has
    - dual Intel Xeon 3.0 GHz processors,
    - 512 KB L2 cache
    - PCI-X 64-bit 133 MHz bus.
  - HCAs - InfiniHost MT23108 DualPort 4x HCAs from Mellanox.
- Version and Configuration -
  - To reduce the impact of I/O bus, the two HCAs are connected to PCI-X buses connected to different I/O bridges of the Server-Works GC LE chipsets.
  - All nodes are connected to a single Mellanox InfiniScale 24 port switch (MTS 2400), which supports all 24 ports running at full 4x speed.
  - kernel version we used is Linux 2.4.22smp.
  - The InfiniHost SDK version is 3.0.1 and
  - HCA firmware version is 3.0.1.
  - The Front Side Bus (FSB) of each node runs at 533MHz.
  - The physical memory is 1 GB of PC2100 DDR-SDRAM.

## Performance Evaluation - Microbenchmarks

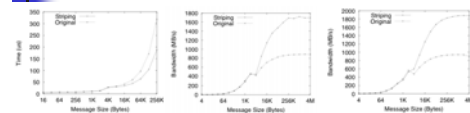


Figure 5. MPI Latency (UP mode)

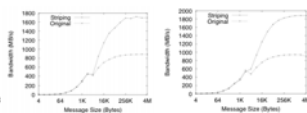


Figure 7. MPI Bandwidth (UP mode)

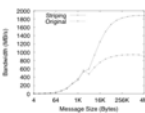


Figure 6. MPI Bidirectional Bandwidth (UP mode)

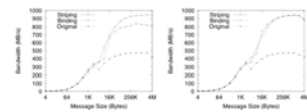


Figure 9. MPI Bandwidth (SMP mode)

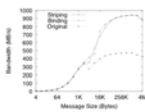


Figure 10. MPI Bidirectional Bandwidth (SMP mode)

## Performance Evaluation - Pallas Benchmarks

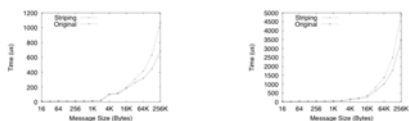


Figure 11. MPI Bcast Latency (UP mode)

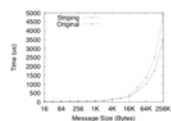


Figure 12. MPI Alltoall Latency (UP mode)

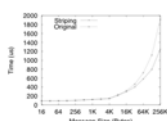


Figure 13. MPI Bcast Latency (SMP mode)

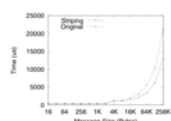


Figure 14. MPI Alltoall Latency (SMP mode)

## Papers

- J. Liu, A. Vishnu and D. K. Panda, Building Multirail InfiniBand Clusters: MPI-Level Design and Performance Evaluation.
- A. Vishnu, M. Koop, A. Moody, A. Mamidala, S. Narravula and D. K. Panda, Hot-Spot Avoidance With Multi-Pathing Over InfiniBand: An MPI Perspective,
- Q. Gao, W. Yu, W. Huang and D. K. Panda, Application-Transparent Checkpoint/Restart for MPI Programs over InfiniBand
- S. Narravula, A. R. Mamidala, A. Vishnu, G. Santhanaraman, and D. K. Panda, High Performance MPI over iWARP: Early Experiences

## Introduction

- MPI Functionality which provides hot-spot avoidance for different communication patterns, without apriori knowledge of the pattern
- Leverage LID Mask Count (LMC) to create multiple paths in the network

## Background

P= Processing Element

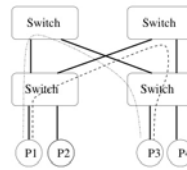


Figure 1. A Fat Tree with Four Switches

- Overview of InfiniBand
  - Each Infiniband port in a network is identified by one of more Local Identifiers (LIDs), assigned by subnet manager
  - Destination based routing - Each switch has a routing table corresponding to the LIDs of the destination.
  - Each port can be assigned multiple LIDs to exploit multiple paths in the network.
- Fat Tree Topology
  - General purpose interconnection topology used for effective utilization of h/w resource devoted to communication
  - Leaf nodes - processor
  - Internal nodes - switches
  - Edges - bidirectional links

## Motivation

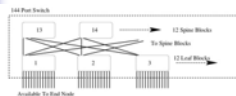


Figure 2. 144-port InfiniBand Switch Block Diagram

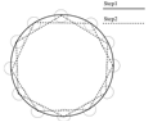


Figure 3. Communication Steps in Displaced Ring Communication

- Displaced Ring Communication
  - $rank_i$  = rank of  $i$ th process in the program
  - $step_j$  =  $j$ th step during execution
  - At  $step_j$ , an MPI process with  $rank_i$  communicates with MPI process  $rank_j$

## Motivation

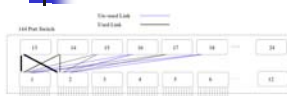


Figure 4. Link Usage with Displaced Ring Communication

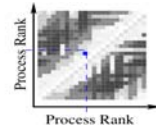


Figure 5. Displaced Ring Communication, 24 Processes

- Experimental setup -
  - Program with 24 processes
  - Schedule MPI processes with rank0 - rank11 on nodes connected to block 1 and rank12 - rank23 on block 2
  - MVAPICH used for evaluation of DRC
- Observations -
  - No contention for intra-block communication
  - Link contention increases as the step iteration increases
  - Deterministic routing nature of Infiniband does not allow us to use these links efficiently

## Hot-Spot Avoidance MPI (HSAM) over Infiniband

- Leveraging Multiple Paths Using LMC
  - Using LMC value of  $x$ ,  $2^x$  paths can be created (max value allowed for LMC = 7)
  - OpenSM used to configure these paths utilizing different spine blocks in the switch.
  - Efficient Utilization dependent upon the scheduling policy for data transfer.
- Scheduling Policies - static vs. dynamic
  - Even Striping - fixed weight distribution
  - Adaptive Striping - weights adjusted in accordance with the completion times of stripes

## Hot-Spot Avoidance MPI (HSAM)

- Selecting Number of Paths
  - Practical considerations in leveraging all paths simultaneously -
    - Sending a message stripe requires posting a descriptor - startup overhead
    - For each message stripe a completion is generated on sender side. Increasing number of paths, more completions need to be handled delaying progress of application
    - Accuracy of path bandwidth dependent upon discovery of completions
    - Memory usage increases with increasing number of paths.

## Scalability Aspects of HSAM

- Increasing number of paths increases memory utilization

- Memory utilization per path:

$$mem_{qp} = mem_{qp-context} + ne_s * mem_{sq} + ne_r * mem_{rp}$$

- $mem_{qp}$  = connection memory usage per path
- $ne_s, ne_r$  = Number of send/receive work queues
- $mem_{sq}, mem_{rp}$  = size of each send queue and receive queue elements
- $mem_{qp-context}$  = size of each qp context, corresponding to each path

As a result, memory usage in our design can be represented as:

$$mem_{qp} = mem_{qp-context} + ne_s * mem_{sq}$$

## Performance Evaluation

- Experimental Testbed

- Components -
  - 64 nodes: 32 nodes with Intel EM64T architecture and 32 nodes with AMD Opteron architecture.
  - Each node Intel with EM64T architecture is -
    - dual socket, single core with 3.6 GHz, has 2 MB L2 cache and 2 GB DDR 2533 MHz main memory.
  - Each node with AMD Opteron architecture is -
    - dual-socket, single core with 2.8 GHz, has 1 MB L2 cache and 4 GB DDR 2533 MHz main memory.
- Configuration and version -
  - I/O bus is x8 PCI-Express with Mellanox MT25208 dual-port DDR HCAs attached to 144-port DDR Flextronics switch.
  - The Mellanox firmware version is 5.1.400.
  - Open Fabrics Enterprise distribution (OFED) version 1.1 for evaluation on each of the nodes
  - OpenSM as the subnet manager, distributed with this version.

## Performance Benefits of HSAM with Collective Communication

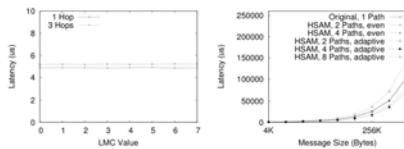


Figure 7. MPI Latency (UP mode)

Figure 8. MPI Alltoall Personalized (48x1)

- Increasing LMC does not increase latency
- Latency when processes are located on the different block (3-hops) is 0.25us higher than the latency when processes are located on same block (1-hop)

## Performance Evaluation

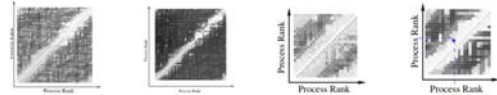


Figure 9. Displaced Ring Communication, 64x1, HSAM, 4 Paths, adaptive

Figure 10. Displaced Ring Communication, 64x1, Original, 1 Path

Figure 11. Displaced Ring Communication, 24x1, HSAM, 4 Paths, adaptive

Figure 12. Displaced Ring Communication, 24x1, Original, 1 Path

- On InfiniBand cluster with 64 processors average bandwidth improves by 23% by using HSAM and the adaptive policy

## Papers

- J. Liu, A. Vishnu and D. K. Panda, Building Multirail InfiniBand Clusters: MPI-Level Design and Performance Evaluation.
- A. Vishnu, M. Koop, A. Moody, A. Mamidala, S. Narravula and D. K. Panda, Hot-Spot Avoidance With Multi-Pathing Over InfiniBand: An MPI Perspective
- O. Gao, W. Yu, W. Huang and D. K. Panda, Application-Transparent Checkpoint/Restart for MPI Programs over InfiniBand
- S. Narravula, A. R. Mamidala, A. Vishnu, G. Santhanaraman, and D. K. Panda, High Performance MPI over iWARP: Early Experiences

## Introduction

- The Message Passing Interface (MPI) has no specification about fault tolerant support that a particular implementation must achieve. Hence MPI implementations too lack fault tolerant support.
- Checkpoint/restart design implemented in MPI-2, based on the capability of Berkeley Lab's Check-point/Restart (BCLR) to take snapshots of processes on a single node
- Takes global checkpoints of the entire MPI program while ensuring global consistency
- Low-overhead, application transparent checkpointing, with only insignificant performance impact.

## Checkpointing and Rollback Recovery

- We choose coordinated checkpointing because:
  - Message logging can impose considerable overhead
  - Uncoordinated checkpointing is susceptible to domino effect
- Checkpoint/restart techniques can be categorized as:
  - Application level checkpointing
    - Efficiency with assistance from user application
    - Source code needs to be tailored to checkpointing interface
  - System-level checkpointing
    - Application-transparent, no code modification required
    - More overhead
- Design Objectives
  - Consistency
  - Transparency
  - Responsiveness

## Checkpoint/Restart Framework and Design Issues

- Checkpoint/Restart Framework
  - A typical MPI program consists of: a front-end MPI job console, a process manager and individual MPI processes running on these nodes.
  - Multi Purpose Daemon (MPD) - default process manager. All MPDs are connected as a ring
  - Key components of the proposed structure:
    - Global C/R Coordinator
    - Control Message Manager - Interface between global C/R Coordinator and local C/R controller
    - Local C/R Controller
    - C/R Library
    - Communication Channel Manager

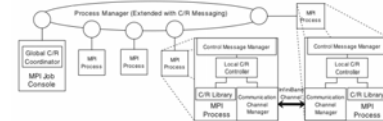


Figure 1. Checkpoint/Restart Framework

## Overall Checkpoint/Restart Procedure

- Checkpointing cycle consists of four phases:
  - Initial Synchronization Phase
    - All processes in MPI jobs synchronize with each other and prepare for checkpoint coordination
  - Pre-Checkpoint Coordination Phase
    - C/R controllers coordinate with each other to make all MPI processes individually checkpointable.
  - Local Checkpointing Phase
    - C/R controllers invoke C/R library to save the current state of local MPI process
  - Post-checkpoint Coordination Phase
    - C/R controllers cooperate with communication channel managers to reactivate communication channels
- Restarting Phase

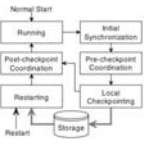


Figure 2. State Diagram for Checkpoint/Restart

## Checkpoint/Restart Framework

- Suspension/Reactivation InfiniBand Channel
  - Network connection information - user-level data structures
  - Dedicated Communication buffers - registered buffers directly accessed by HCA for sending/receiving small messages
  - Channel progress information - book-keeping and flow control
  - Registered user buffers - registered by communication channel to HCA for zero-copy transmission of large messages

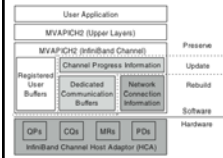


Figure 3. InfiniBand Channel for MVAPICH2

## Performance Evaluation

- Components -
  - InfiniBand cluster of 12 nodes.
  - Each node is equipped with
    - dual Intel Xeon 3.4GHz CPUs,
    - 2GB memory
    - Mellanox MT25208 PCI-Express InfiniBand HCA.
    - Operating system: Redhat Linux AS4 with kernel 2.6.11.
    - File system: ext3 on top of local SATA disk.

## Overhead Analysis for Checkpoint/Restart

- Time for checkpointing/restarting is determined by two factors:
  - Time for coordination - increases with file size
  - Time for writing/reading the checkpoint file to/from file systems - depends on checkpoint file size and performance of underlying file system.

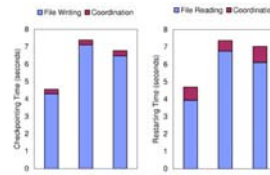


Figure 4. Overall Time for Checkpointing/Restarting NAS

# Performance Impact

## 5.2 Performance Impact for Checkpointing

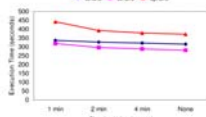


Figure 7. Performance Impact for Checkpointing NAS

- Very little extra book-keeping overhead on data communication introduced by C/R functionality so that checkpoint/restart-capable MVAPICH2 has almost the same performance as original MVAPICH2 if no checkpoint is taken.
- Total running time of LU, BT, and SP decreases as the checkpointing interval increases.

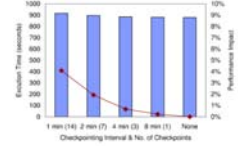


Figure 9. Performance Impact for Checkpointing GROMACS

# Papers

- J. Liu, A. Vishnu and D. K. Panda, Building Multirail InfiniBand Clusters: MPI-Level Design and Performance Evaluation.
- A. Vishnu, M. Koop, A. Moody, A. Mamidala, S. Narravula and D. K. Panda, Hot-Spot Avoidance With Multi-Pathing Over InfiniBand: An MPI Perspective,
- Q. Gao, W. Yu, W. Huang and D. K. Panda, Application-Transparent Checkpoint/Restart for MPI Programs over InfiniBand
- S. Narravula, A. R. Mamidala, A. Vishnu, G. Santhanaraman, and D. K. Panda, High Performance MPI over iWARP: Early Experiences

# High Performance MPI over iWARP: Early Experiences



Figure 1. Basic Steps for Connection Setup Using RDMA CM

- iWARP
  - Defines RDMA operations over Ethernet networks
  - Supports two types of communication semantics
    - Channel Semantics (Send-Receive communication model)
    - Memory Semantics (RDMA communication model)
  - Basic communication is achieved over connected end points known as Queue Pairs
- RDMA-CM
  - Establishes connections between QPs of a pair of processes based on IP address and port number

# Design and Implementation

- RDMA-CM based Connection Management
  - Mismatch in the connection semantics of RDMA-CM and the MPI processes
    - MPI library assumes fully connected model while RDMA CM connection is based on traditional TCP/IP style
  - Differences in the current connection setup in MVAPICH2 and the mechanism used by RDMA-CM

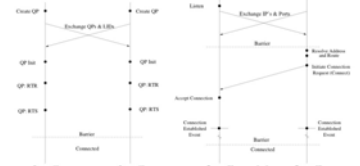


Figure 2. Connection Setup Mechanisms: (a) Existing OFA Gen2 Verbs and (b) RDMA-CM

# Design and Implementation

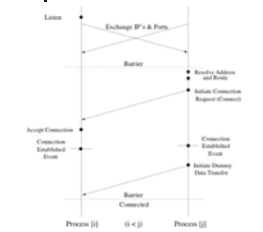


Figure 3. RDMA CM based Connection Setup for MVAPICH2

- Connection Initiation for iWARP
  - iWARP MPA requires client process to initiate first data send or first RDMA write. Thus MPI processes with lower ranks cannot initiate first data transfer.
  - This aspect is handled by having an extra dummy message sent from each of the client processes to all its server processes before marking connection as connected.
- RDMA based data transfers
  - Accelerated data transfer mechanism for small messages
  - Switch provided to enable this mechanism for adapters that guarantee in order placement of data.

# Experimental Setup

- Components
  - Four node cluster
  - Each node contains -
    - Two quad core Intel Xeon 2.33 GHz
    - Memory of 4 GB each.
    - Chelsio T3B10 GigE PCI-Express adapters (Firmware Version 4.2) plugged into an x8 PCI-Express slot.
- Version and Configuration -
  - The Chelsio adapters are connected through a 24 port Fulcrum 10 GigE evaluation switch.
  - MTU used for the path is 9000 bytes.
  - Software stack - OFED 1.2 rc4
  - Operating systems: RH4 U4.
  - MPICH2 1.0.5p3 is used for comparisons

## Experimental Results

- MPI Send/Recv: Latency and Bandwidth

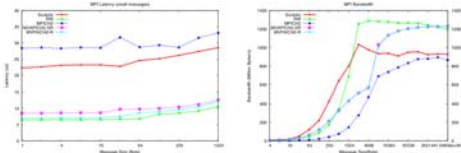


Figure 4. Basic Performance: (a) Latency and (b) Bandwidth

## Experimental Results

- MPI Get/Put Performance: Latency and Bandwidth

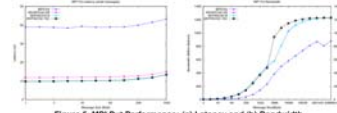


Figure 5. MPI Put Performance: (a) Latency and (b) Bandwidth

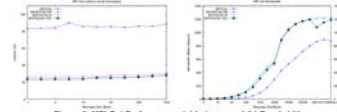


Figure 6. MPI Get Performance: (a) Latency and (b) Bandwidth