

Using Infiniband's Hardware Multicast support to optimize collective performance.

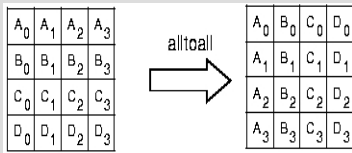
Hari Subramoni.

## Commonly used collective calls.

- MPI\_Barrier
  - Blocks the caller until all group members have called it.
  - Returns at any process only after all group members have entered the call.
- MPI\_Bcast
  - Broadcasts a message from the process with rank root to all processes of the group, itself included.
  - Called by all members of group with same arguments.
  - On return, the contents of root's communication buffer has been copied to all processes.

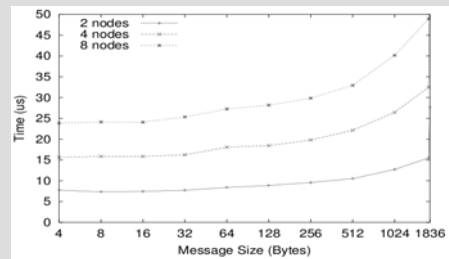
### •MPI\_Alltoall

- Each process sends distinct data to each of the receivers.
- The *j*th block sent from process *i* is received by process *j* and is placed in the *i*th block of recvbuf.



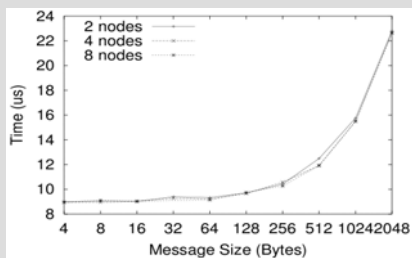
## So what?

- All collective operations use point to point communication scheme.
- Hence, as the number of nodes increases, latency increases as well.



## How can we improve this?

- As communication is the major hurdle, why not move it to hardware using Infiniband's hardware multicast.



## What are the challenges?

- Fact: Infiniband hardware multicast uses UD protocol.
- Challenge: Due to the unreliable nature of UD, higher level applications like MPI is unable to use it
- Solution: Create an abstraction layer with the following capabilities.
  - Reliability
  - In-order packet delivery.
  - Ability to handle large messages.

## Challenges in designing the abstraction layer.

- Providing reliability and other features comes with some overhead.
- The challenge lies in
  - Removing the overhead from the communication critical path to the background.
  - Balance and reduce this background overhead so that it doesn't become a performance bottleneck.

## Solution – Basic Design.

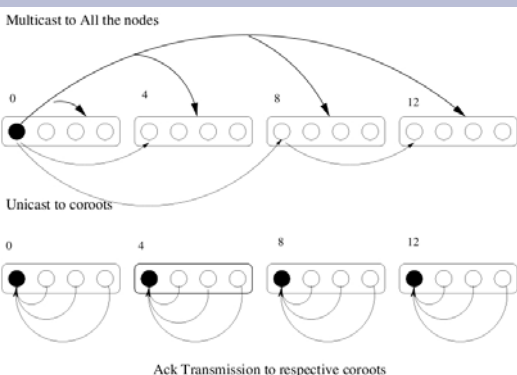
- Use an ACK based scheme to ensure reliability and in-order message delivery.
- Root sends out a broadcast message using infiniband's multicast feature and waits for ACK's from all processes.
- If ACK not received within timeout, we retransmit.
- Problems:
  - Blocking mode of operation creates a bottleneck at root.
  - Hot spot is created at root due to large number of ACK's that reach it.

## Sliding Window at Root.

- Use multiple pre-allocated buffers organized in a ring fashion at root.
- Input buffer given to MPI\_Bcast is copied to this buffer and the call immediately returns without waiting for the ACK's.
- Buffer freed when all ACK's arrive for the message.
- Advantages
  - No blocking at root.
  - ACK processing moved out of the main communication path.
- Disadvantages
  - More buffer space required.
  - Extra copy.
  - Hot spot at root due to ACK implosion.

## Avoiding ACK Implosion.

- We could use an ACK collection scheme where each node collects all the ACK's from it's children and sends one ACK up to it's parent.
- Problems: Very susceptible to process skew, resulting in false re-transmissions.
- Alternative approach: Co-root scheme.
  - Root sends reliable point to point transfer to co-roots and multicast to all other nodes.
  - The root and each co-root is responsible to collect ACK's from a pre-specified set of nodes.
  - Root / Co-root takes care of re-transmission to all nodes in it's local group.



## Reducing ACK traffic.

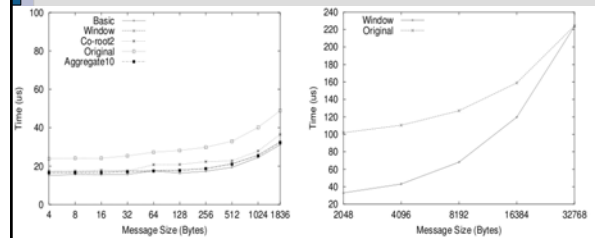
- Piggy backing – ACK is piggybacked along with another data transmission.
- Problems: Dependant on data traffic. If sparse, then we end up sending ACK's by itself. But even here, we can combine multiple ACK's into one message, thereby reducing ACK traffic.
- ACK for every Mth message – Reduces ACK's to 1/Mth of the original number.
- Problem: Might lead to ACK implosion at M'th ACK. So skew the ACK's sent by each node.

## Other Issues.

- Large messages: Fall back to Point to Point mode of operation due to high copy cost, and simplified design and implementation.
- Buffer management: If the ring buffer is full, we consider it similar to a timeout and re-transmit the oldest message in the queue.
- Duplicate messages are dropped silently.
- Out of order messages are buffered to be processed later.
- Use a credit based scheme for flow control.
  - Credit information is piggybacked on normal messages.

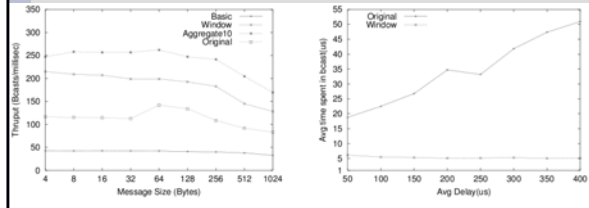
## Broadcast Latency Test.

- Multicast based designs performs better than normal schemes for both small and large messages. (Best – up to 58%).
- For very large messages, they converge due to fragmentation and re-assembly cost.



## Broadcast Thruput & Impact of process skew.

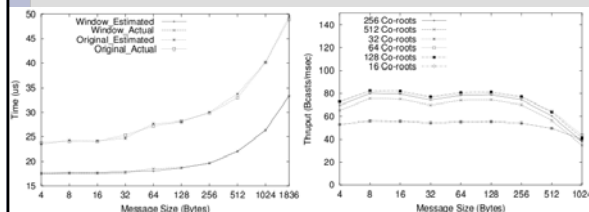
- Basic design performs worse than the original point to point scheme due to blocking mode of operation and ACK implosion at root.
- Aggregate 10 performs the best due to ACK reduction and non-blocking operation. Best performance improvement seen = 112%
- Performance drops in original scheme with increase in process skew due to dependency on intermediate nodes for communication.



## Analytical Model

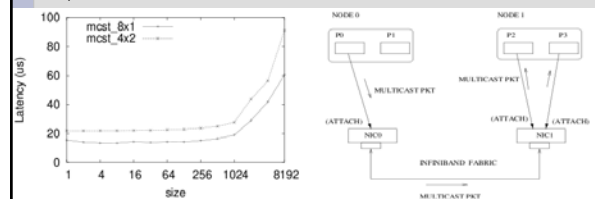
- Total Latency for original bcast operation = Average latency of one hop \* number of hops ( $\log(n)$ ).
- Latency for sliding window model ( $T_{sr}$ ) = Cost of copying at root + Cost of posting a UD descriptor + Latency of HW Multicast + Cost of copying at receiver.
- Cost of RC in co-root scheme ( $T_{rc}$ ) = Cost of copying at root + Average latency of one hop \* number of hops ( $\log(n/s)$ ).
- Cost of getting msg to co-root =  $\text{Min}(T_{rc}, T_{sr})$ .

- Estimated and actual latencies are more or less same.
- With too many co-roots, performance is bad due to large number of reliable point to point messages.
- With too few co-roots, performance is bad due to too much ACK processing at co-roots.



## What about Shared Memory Processors?

- No, This will not scale well for SMP's where shared memory communications yield much lesser latencies for intra-node communication. Why?
  - NIC looks up QP's attached to multicast group.
  - Replicates packets and DMA's them to each process.
  - As DMA is sequentialized, cost increases with increase in the number of local processes.

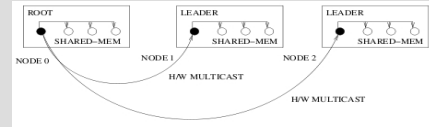


## Solution – Possible Designs.

- Use a combination of shared memory channel and hardware multicast.
  - Direct multicast to shared memory
    - Packets directly received into shared memory region. Local processes copy from here.
  - Problems.
    - Getting notification has performance overheads.
    - Requires process to be part of the multicast group. And as seen before this approach has drawbacks.
- Leader based approach
  - Similar to the co-root scheme. Each node is assigned a leader process that receives and processes the multicast message.
  - It notifies other local processes of the arrival of the message.

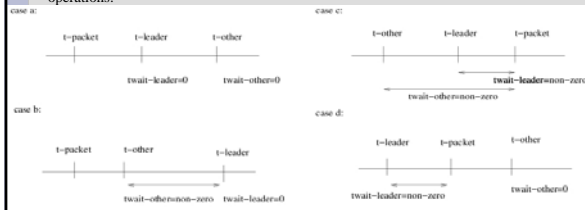
## Leader Based Approach.

- Some processes are selected as leader processes for a group of local processes, with the root being the leader for its process group (Now, process with local ID 0 is chosen as leader).
- Root multicasts the packet to all other leader processes which would've registered to the multicast group.
- On receiving the message, the leader gives it to other processes via the shared memory channel.



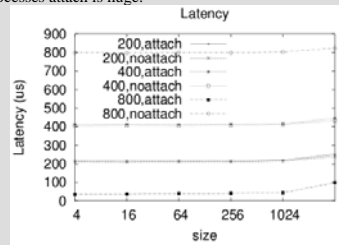
## Dynamic Attach Policy

- If the leader process consistently comes late as in 'c', the other processes would've to wait.
- If this wait time exceeds some heuristic limit, the other process attaches itself to the multicast group.
- Once attached, it detaches only slowly due to the high cost of attach/detach operations.



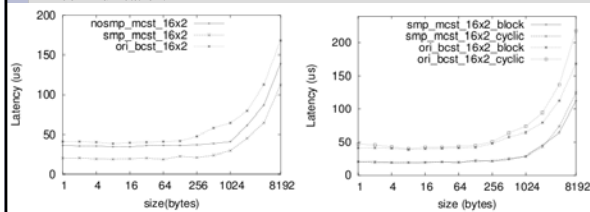
## Performance benefit of Dynamic Attach.

- We see that there is little overhead for implementing the dynamic attach scheme.
- The benefits obtained when we reach the heuristic wait of 800us when the other processes attach is huge.



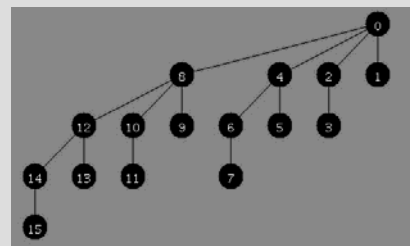
## Performance Evaluation.

- Latency: SMP\_Aware\_Mcast < SMP\_UnAware\_Mcast < Original Broadcast
- In SMP, Block mode latency < Cyclic mode latency.
- This is due to lack of overlap between inter-node and intra-node communication.



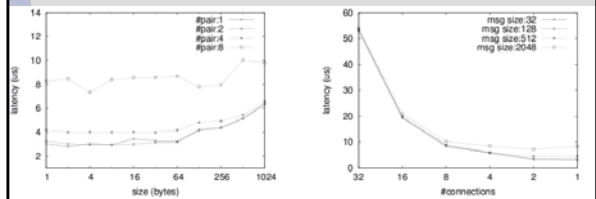
## Why does Block performs better than Cyclic.

- This has to do with the way the processes are arranged in the binomial tree as shown below.



### Performance Evaluation of collectives with small to medium size messages.

- Main bottle neck is startup cost.
- Fewer the number of connections, better is the bandwidth here.
- We again use leader based collective operations for better performance.



### MPIAlltoall

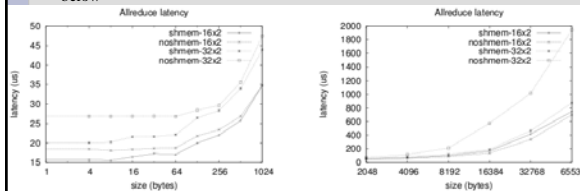
- Has been split up into 3 phases – intra\_Gather, interAlltoall and intra\_Scatter.
- intra\_Gather consists of packing data from local processes and intra\_Scatter consists of the unpacking of data.
- Data is packed according to local ranks of the source and destination node.

### Analytical Model.

- The main aim of the model is to estimate the cut off point where aggregating the message is not beneficial in MPI\_Alltoall
- Latency of Scatter = Serialization Time + Constant
- Depending on which factor dominates the total time, we can write
  - Latency = (n-1)\*Startup time+Transmit time+C or
  - Latency =(n-1)\*Transmit time+Startup time+C
- Further, Transmit time = B \* msg\_size, where B(beta) is the per byte transmission cost.
- From this we get B = 0.000935 and C = -10.
- The negative value of C is serialization overhead is not present for all transactions, due to the presence of multiple DMA engines.

### Conclusions from the model.

- No benefits seen if transmission time dominates start up time.
- This can be determined by finding the point where serialization cost exceeds start-up cost.
  - $x * B = \text{start-up time}$ .
- By substituting, we get  $x = 2243$ , which we can validate from the graphs below



### Savings in Communication Resources.

- m – is the number of intra-node processes and
- n – is the number of inter-node processes.

	Without Shmem	With Shmem	Savings
MPI_Alltoall	$O((n-1)m^2)$	$O(n-1)$	$O(m^2)$
MPI_Barrier	$O(m \log(n))$	$O(\log(n))$	$O(m)$
MPI_Allreduce	$O(m \log(n))$	$O(\log(n))$	$O(m)$

### Acknowledgments & Reference

- Many thanks to Sundeep and Rahul for their help ... :).
- J. Liu, A. Mamidala and D. K. Panda. Fast and Scalable MPI-Level Broadcast using InfiniBand's Hardware Multicast Support.
- A. Mamidala, L. Chai, H.-W. Jin and D. K. Panda, Efficient SMP-Aware MPI-Level Broadcast over InfiniBand's Hardware Multicast
- A. Mamidala, D. De, A. Vishnu, S. Narravula and Dhableswar Panda, Scalable Collective Communication for Next-Generation Multicore Clusters with InfiniBand.
- [www.mpi-forum.org](http://www.mpi-forum.org)