Designing High Performance MPI Intra-node Communication

Presented by: Pai-Wei Lai

Outline

- Background
- MPI Intra-node Communication
- (4g) Small / large messages design
- (4h) LiMIC
- (4i) Hybrid approach
- Conclusion

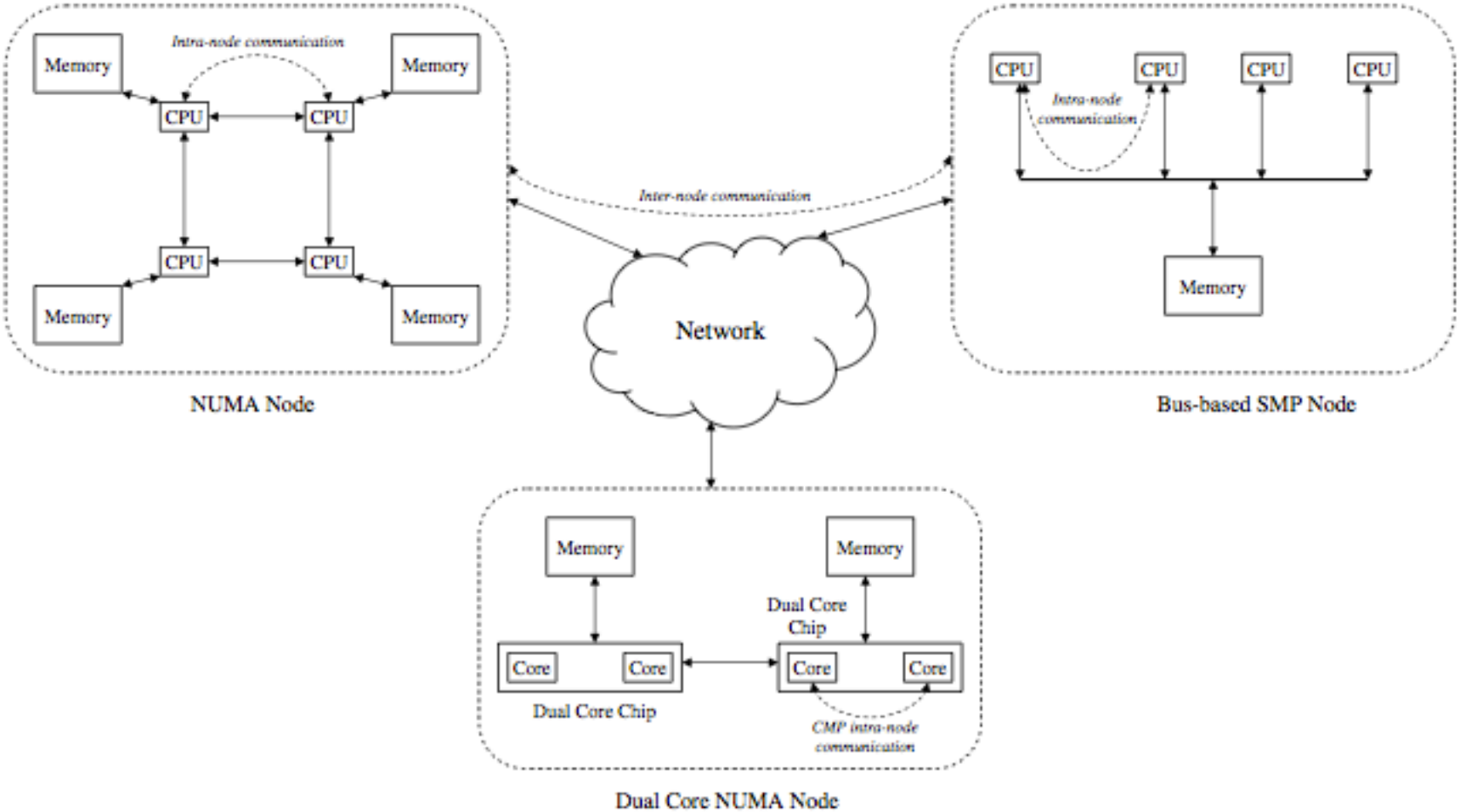
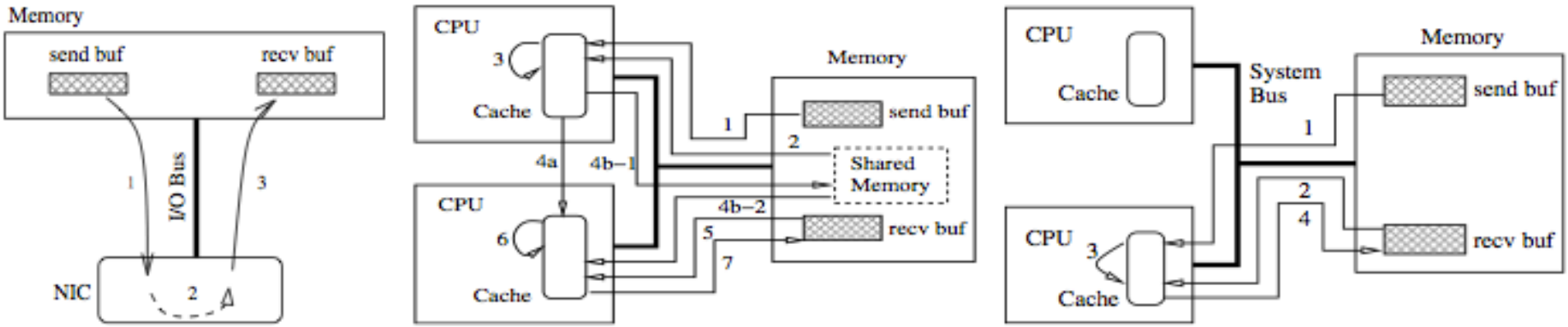


Figure 1. An Illustration of a Cluster Built From SMPs

MPI intra-node communication

- Network loopback
- User-level shared memory
- Kernel-assisted memory mapping

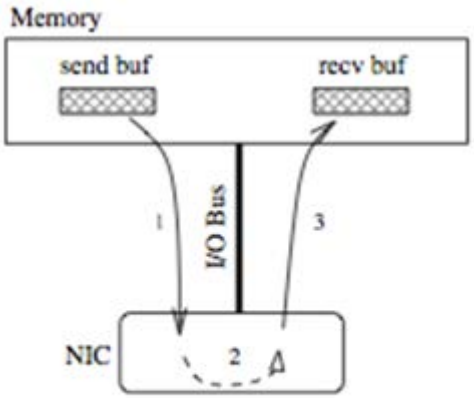


(a) NIC-Level Loopback (b) User-Space Shared Memory (c) Kernel-Based Memory Mapping

Figure 1. Memory Transactions for Different Intra-Node Communication Schemes

(1) Network loopback

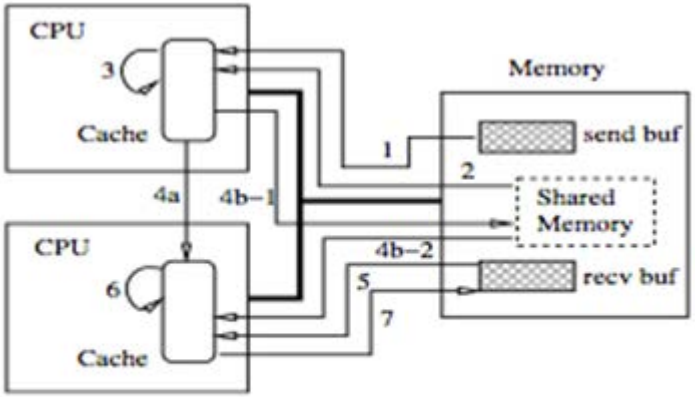
- Does not distinguish intra-node or inter-node traffic
  - Depends on NIC to detect source and destination
  - If source and destination are same node, simply loopback instead of injecting it into network
- Higher latency



(a) NIC-Level Loopback

(2) User-level shared memory

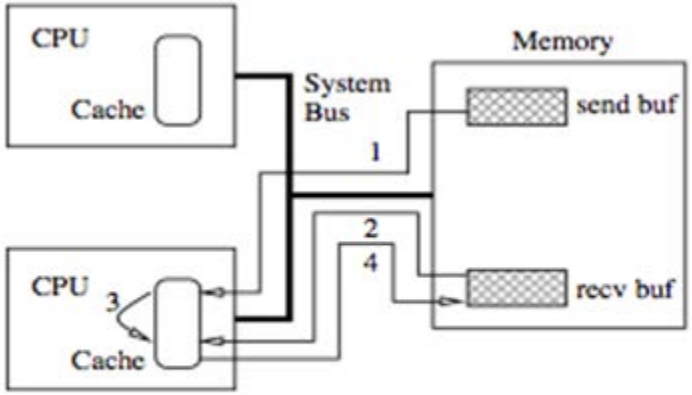
- Most popular with good performance
  - Sending process copies messages into a shared buffer
  - Receiving process copies messages out
- Portability
  - Does not need kernel help



(b) User-Space Shared Memory

(3) Kernel-assisted memory mapping

- Take help from OS kernel
  - Directly copy messages from sender's buffer to receiver's buffer
  - Deploy 'copy-on-write' to reduce number of copies



(c) Kernel-Based Memory Mapping

4g

Designing High Performance and Scalable MPI Intra-code Communication Support for Clusters

- Lei Chai, Albert Hartono, D.K. Panda

Original Design

- Improve (2) based on MVAPICH
- MVAPICH
  - MPI implementation over InfiniBand clusters
  - Each pair of processes on the same node allocate 2 shared memory buffers between them.

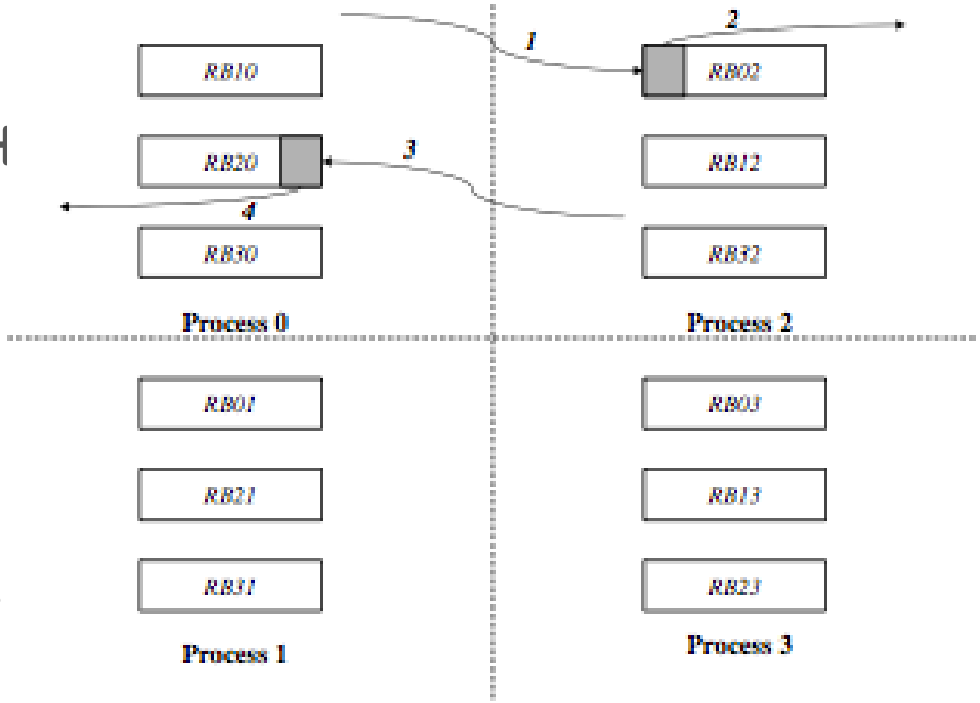


Figure 2. Original Design of MVAPICH Intra-node Communication

New Design

- Overall Architecture
  - P-1 small-sized Receive Buffers (RBs)
  - 1 Send Buffer Pool (SBP)
  - P-1 Send Queues (SQs)
- Message Transfer Schemes
  - Small message
  - Large message

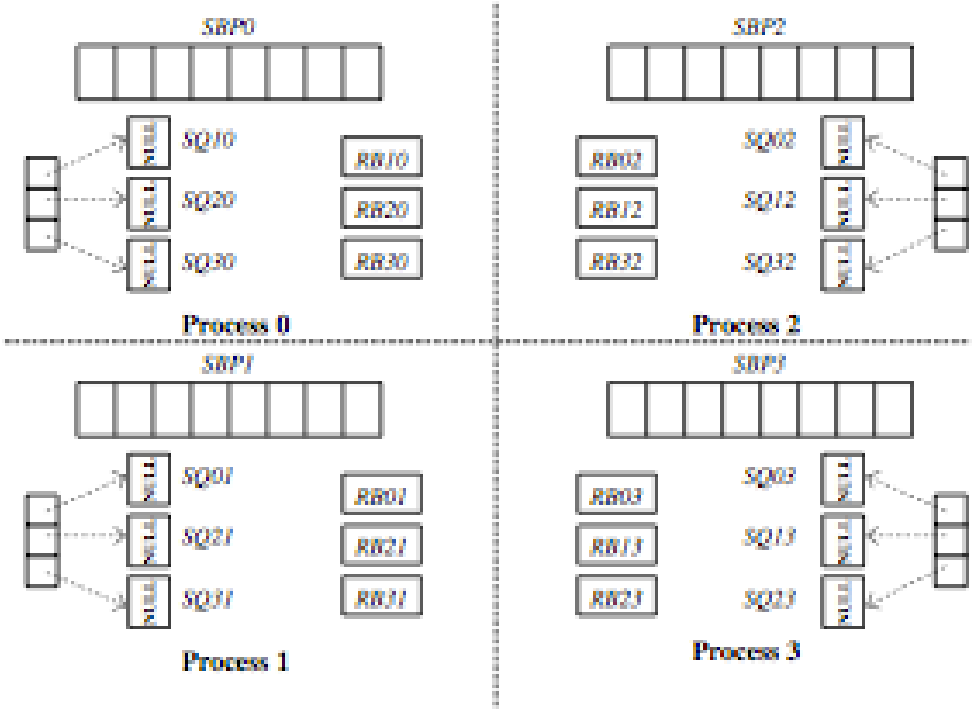


Figure 3. Overall Architecture of the Proposed Design

Small Message Transfer

- Sender directly access the receiver's RB to write the actual data to be sent (1)
- Receiver copies the data from its RB to local buffer (2)

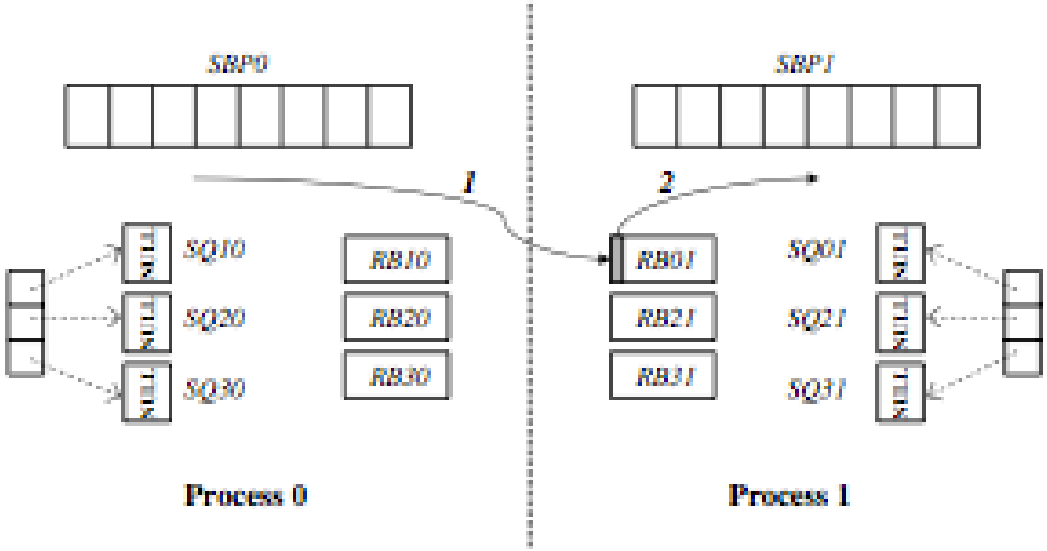


Figure 4. Send/Receive Mechanism for a Small Message

Large Message Transfer

- Sender:
  - Fetch a free cell from SBP, copies the msg into the free cell, and marks cell **BUSY** (1)
  - Enqueue the cell into SQ (2)
  - Sends a control msg including address info of the cell and write it into receiver's RB (3)
- Receiver
  - Read control msg (4)
  - Directly access the cell in SQ (5)
  - Copies the data to its local buffer, marks cell **FREE** (6)

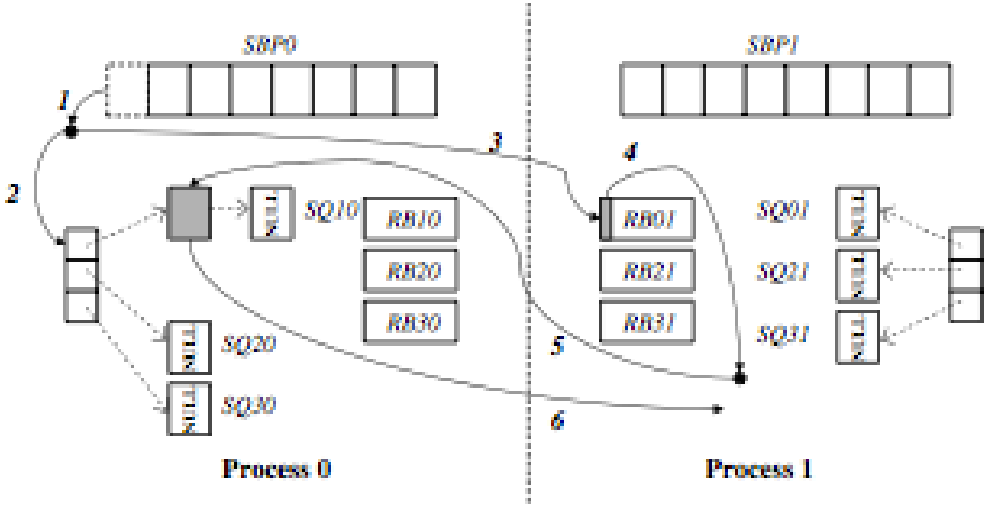


Figure 5. Send/Receive Mechanism for a Large Message

Analysis of Design

- Lock avoidance
  - Mark-and-sweep
- Effective cache utilization
  - RB are designed for small msg, the buffer size can be small that completely fit into cache
  - Cell reuse for sender
- Effective memory usage
  - Original: P*(P-1)*BufSize
  - New: 1 SBP with small P*(P-1)*RB

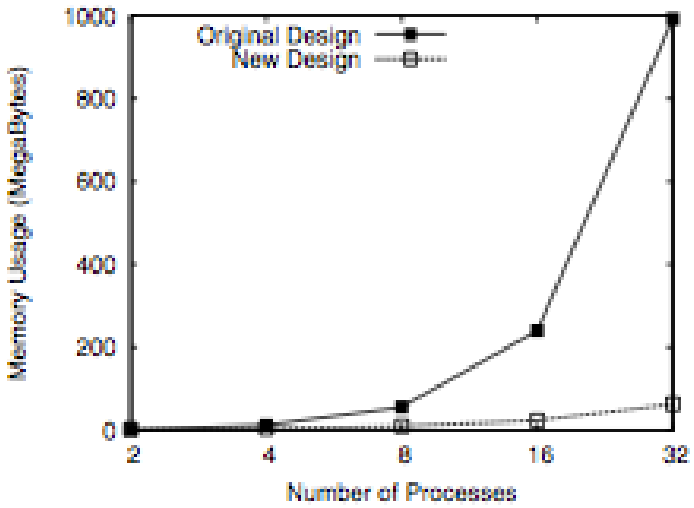
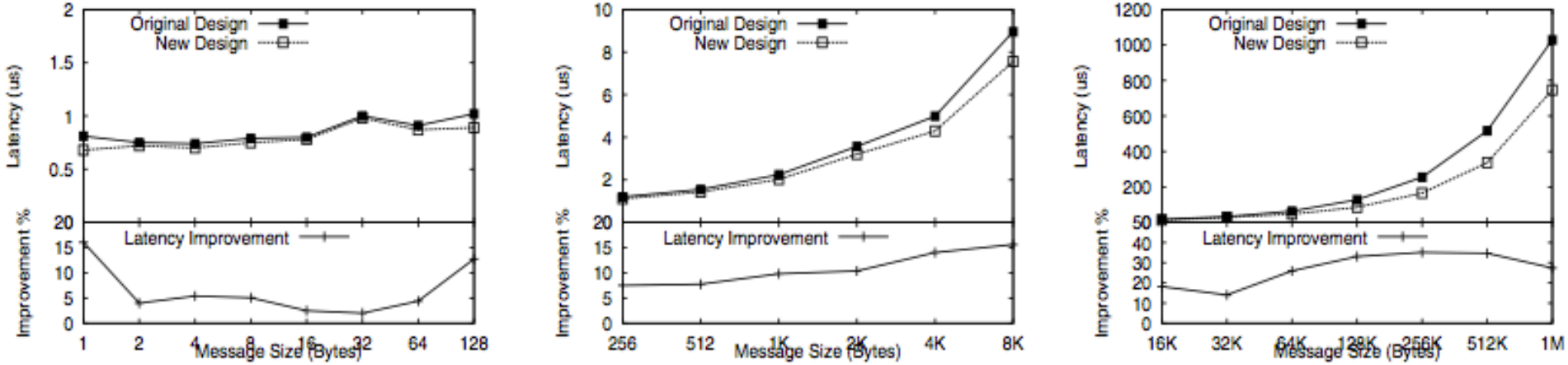


Figure 6. Memory Usage of the Proposed New Design Within a Node



(a) Small Messages (b) Medium Messages (c) Large Messages

Figure 7. Latency on NUMA Cluster

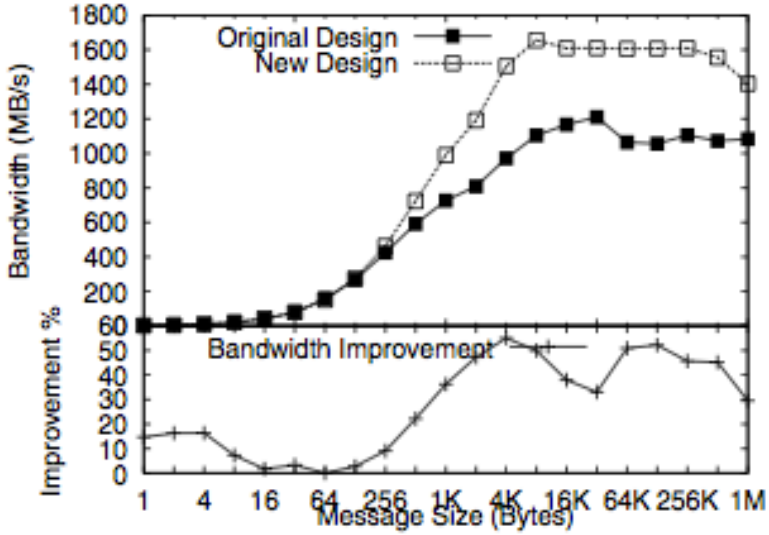


Figure 8. Bandwidth on NUMA Cluster

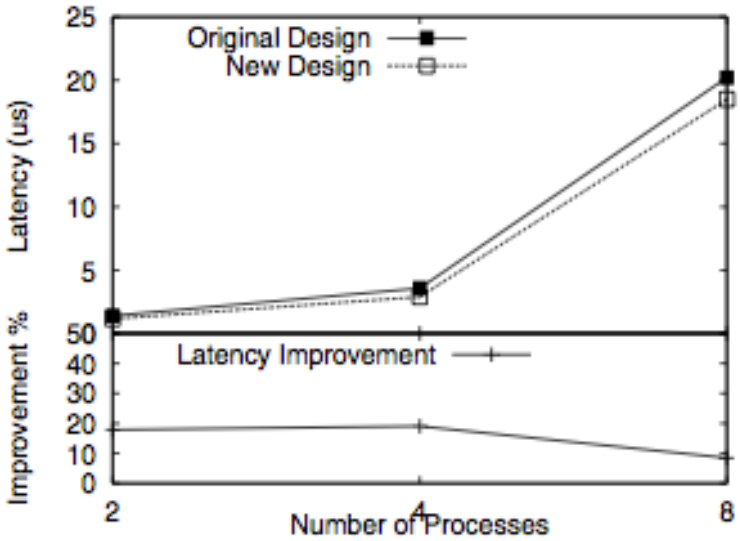


Figure 10. MPI_Barrier Performance

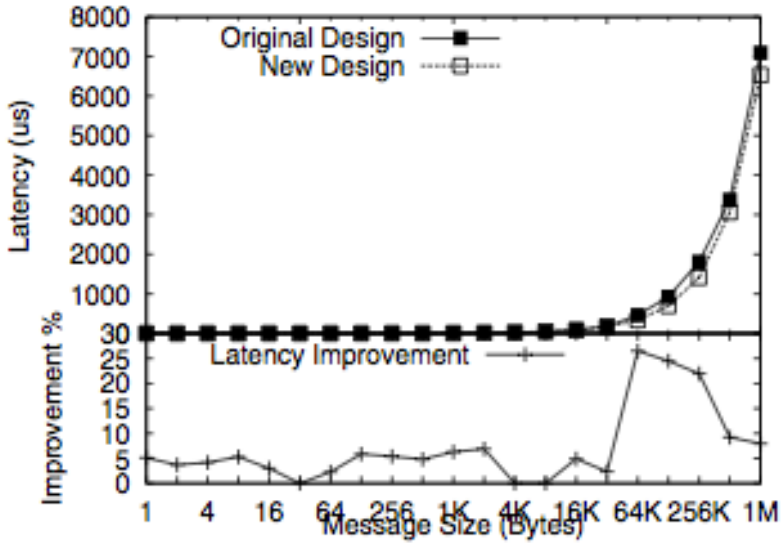
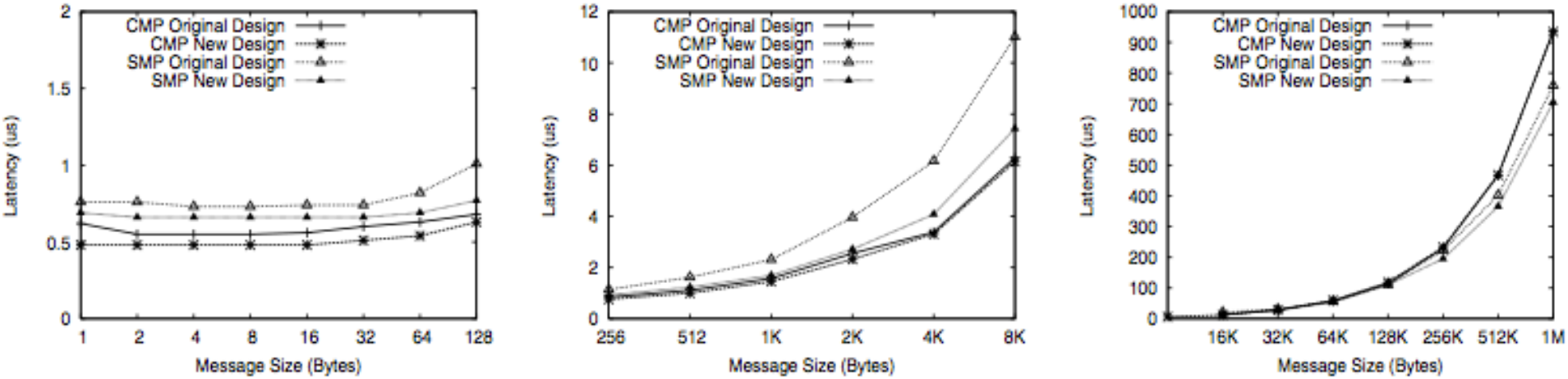


Figure 11. MPI_Alltoall Performance



(a) Small Messages (b) Medium Messages (c) Large Messages

Figure 12. Latency on Dual Core NUMA Cluster

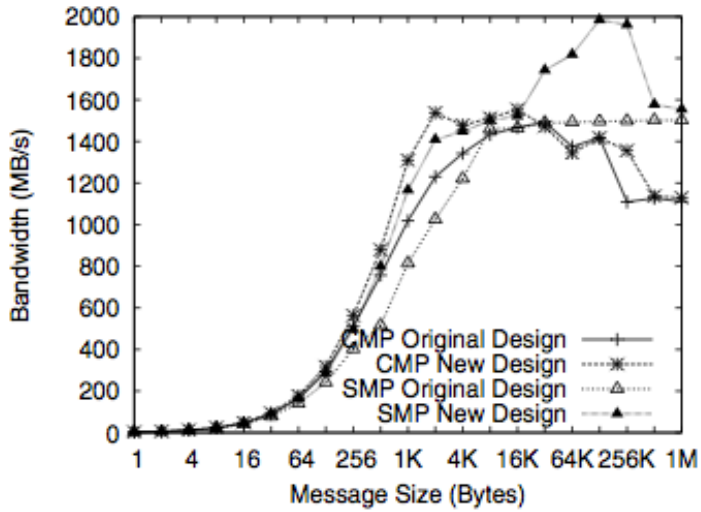


Figure 13. Bandwidth on Dual Core NUMA Cluster

Summary

- Distinguish small/large messages and handle them differently.
  - Direct copy for small / control message
  - SBP decrease the size of memory usage
- Achieved improved performance on NUMA clusters.

4h

LiMIC: Support for High-Performance MPI Intra-Node Communication on Linux Cluster

- Hyun-Wook Jin, Sayantan Sur, Lei Chai, D.K. Panda

- **Li**nux kernal module for **M**PI **I**ntra-node **C**ommunication
- Improve (3) based on integration with MVAPICH

LiMIC high level approach

- Traditionally, kernel based approaches are explored as an extension to user-level protocols.
- As a result, most of these methods have been non-portable to other user-level protocols or other MPI implementations.

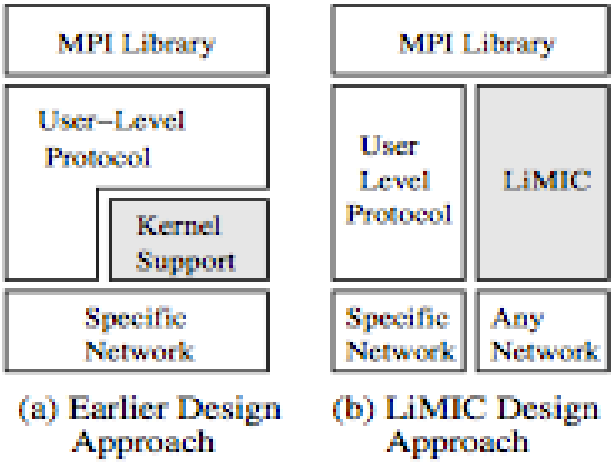


Figure 2. Kernel Support Design Approaches

LiMIC design

- Runtime loadable module with no modification to the kernel code
- Major interface functions
  - LiMIC_Isend: non-blocking send to a destination with appropriate message tags
  - LiMIC_Irecv: non-blocking receive
  - LiMIC_Wait: polls LiMIC completion queue once for incoming send/receives
- Interface does not include any specific information on user-level protocol or interconnect

Memory mapping mechanism

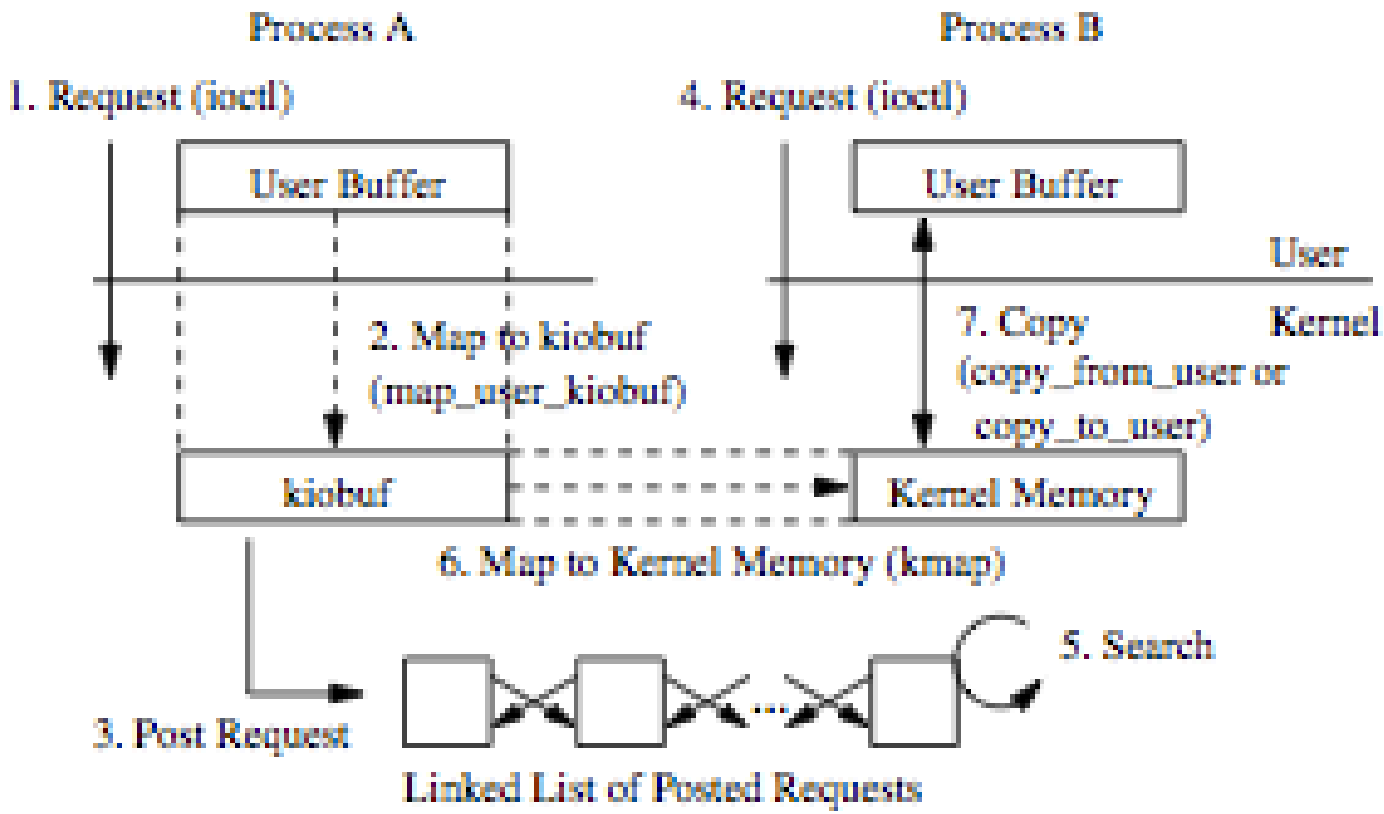
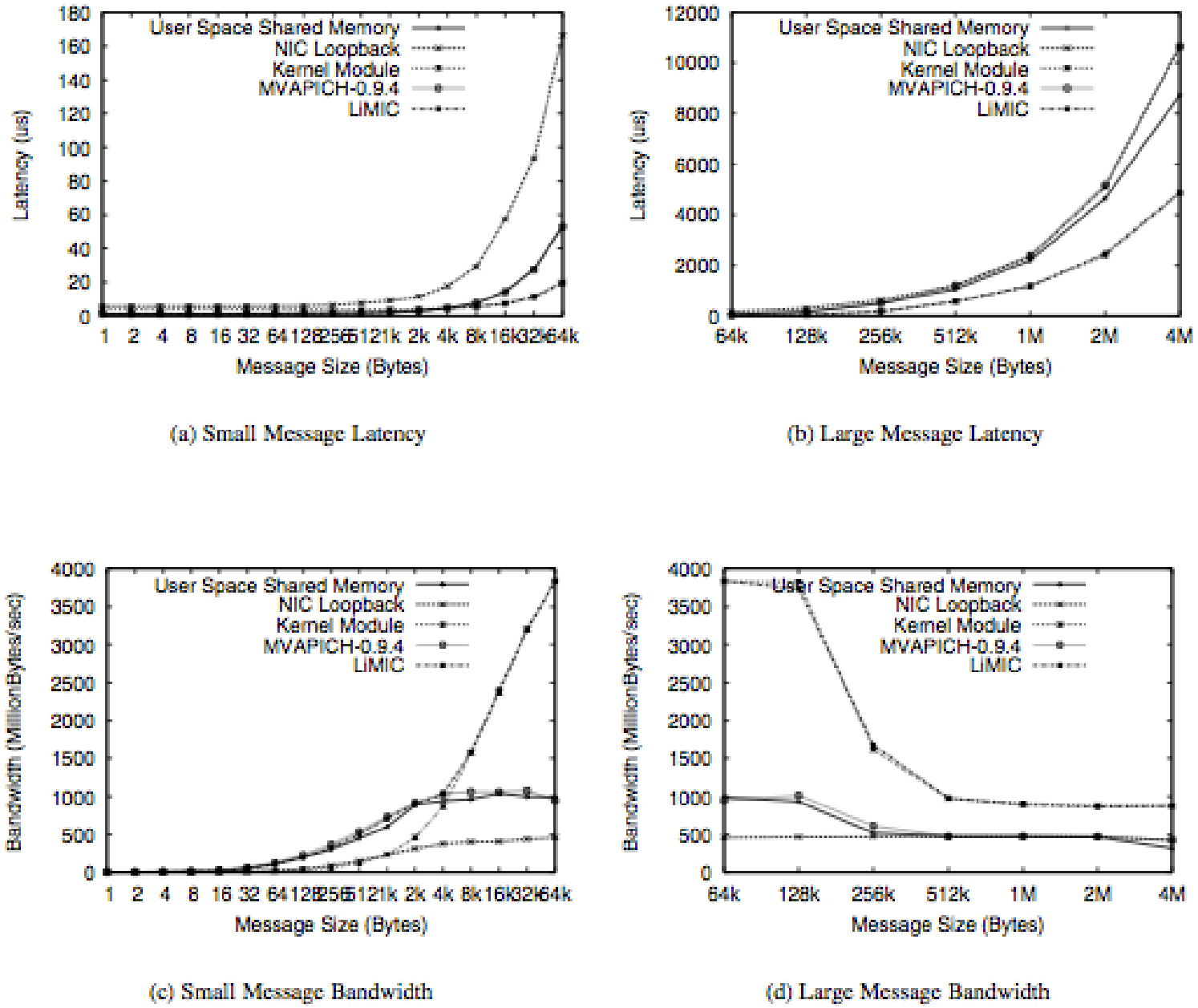


Figure 3. Memory Mapping Mechanism

MPI Message Matching

- There are separate message queue for msg sent/receive through kernel module
- Source in the same node
  - Receiver request is directly posted into the LiMIC_queue
- Source in a different node
  - Not responsible by LiMIC, posted in MPI_queue
- Source in the same node and MPI_ANY_TAGS
- MPI_ANY_SOURCE and MPI_ANY_TAGS
  - Receiver request is posted in the MPI_queue
  - Check using LiMIC_Iprobe, if same node, call LiMIC receive



(a) Small Message Latency

(b) Large Message Latency

(c) Small Message Bandwidth

(d) Large Message Bandwidth

Figure 4. MPI Level Latency and Bandwidth

Performance Evaluation

Table 1. b_eff Results Comparisons (MB/s)

Cluster	Config.	MVAPICH	LiMIC	Improv.
A	2x1	152	244	61%
	2x2	317	378	19%
	2x4	619	694	12%
	2x8	1222	1373	12%
B	2x1	139	183	31%
	2x2	282	308	9%
	2x4	545	572	5%
	2x8	1052	1108	5%
A & B	2x16	2114	2223	5%

Summary

- LiMIC is a stand-alone library that provides MPI-like interfaces that provides memory mapping and kernel assisted direct copy
- Performance of LiMIC with MVAPICH reduced latency by 71% and improved bandwidth by 405% for 64kb msg size

4i

Designing An Efficient Kernel-level and User-level Hybrid Approach for MPI Intra-node Communication on Multi-core Systems

- Lei Chai, Ping Lai, Hyun-Wook Jin, D.K. Panda

Testbed

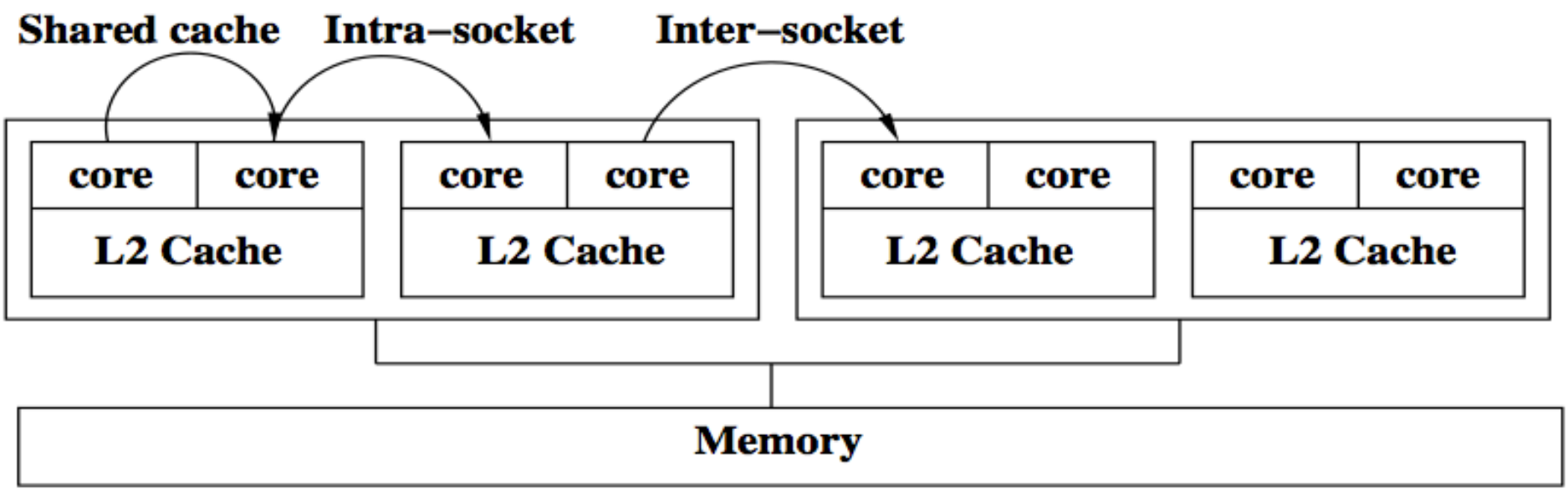
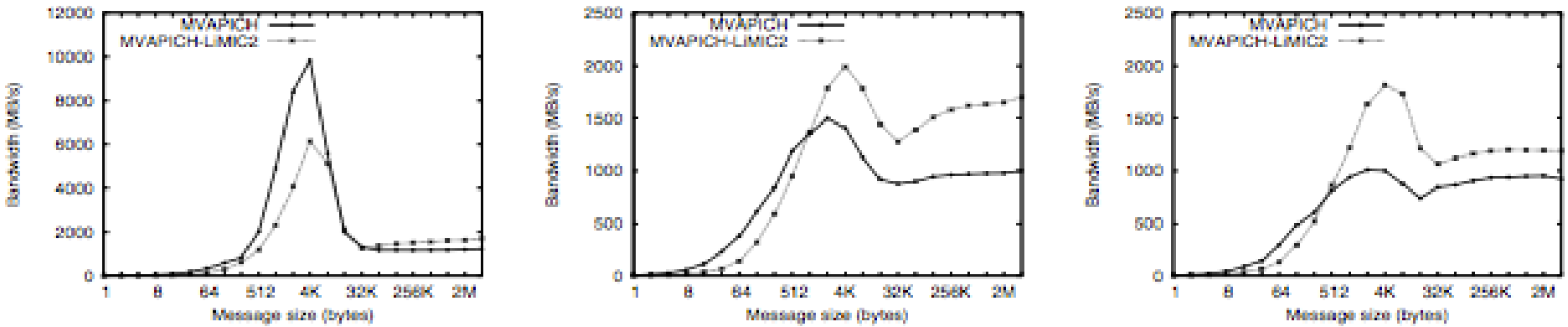


Figure 1. Illustration of Intel Clovertown Processor

Topology

- Step 1: Micro-benchmark analysis
  - Shared memory approach: MVAPICH
  - Kernel-assisted approach: MVAPICH-LiMIC2



(a) Shared Cache (b) Intra-socket (c) Inter-socket

Figure 2. Multi-pair Bandwidth

Buffer reuse and cache utilization

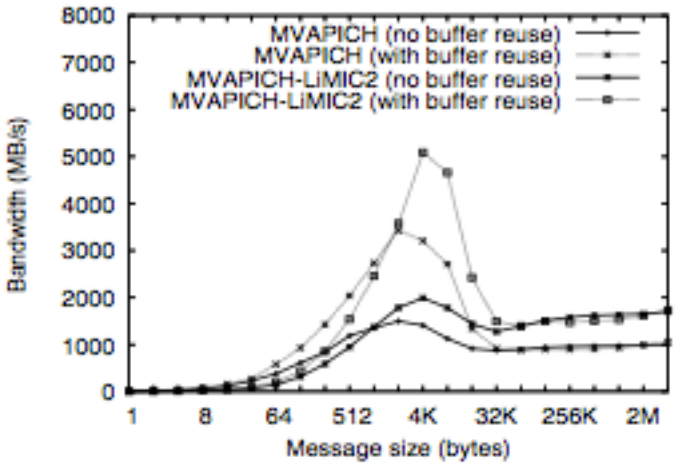


Figure 3. Impact of Buffer Reuse (Intra-socket)

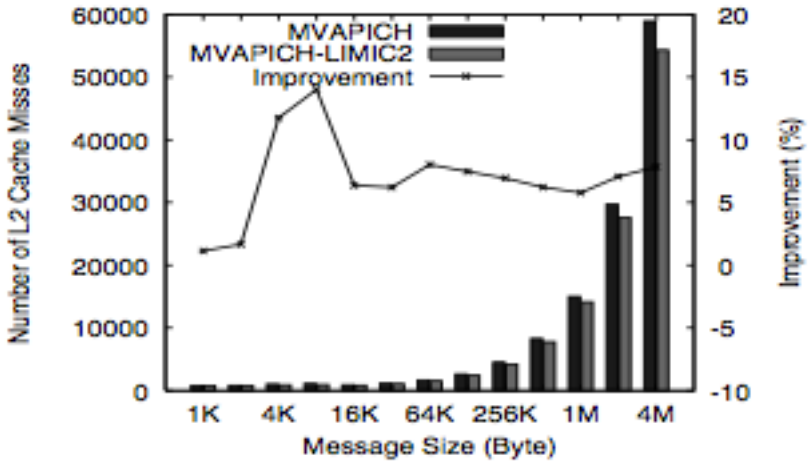


Figure 4. L2 Cache Misses

Process Skew

- Reason: a send operation cannot complete until the matching receive completes
- Comparing to LiMIC2, MVAPICH is more skew-tolerant

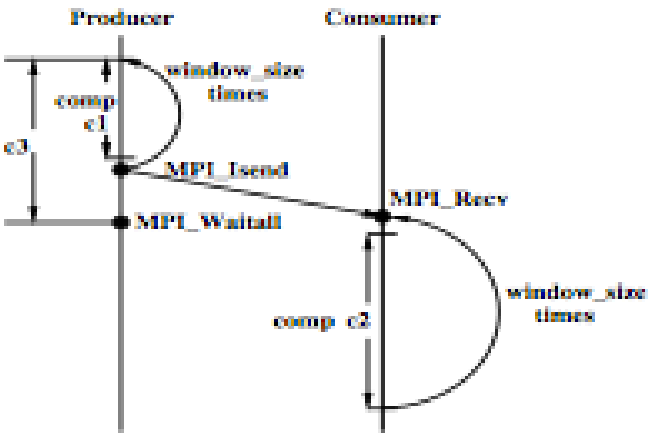


Figure 5. Process Skew Benchmark

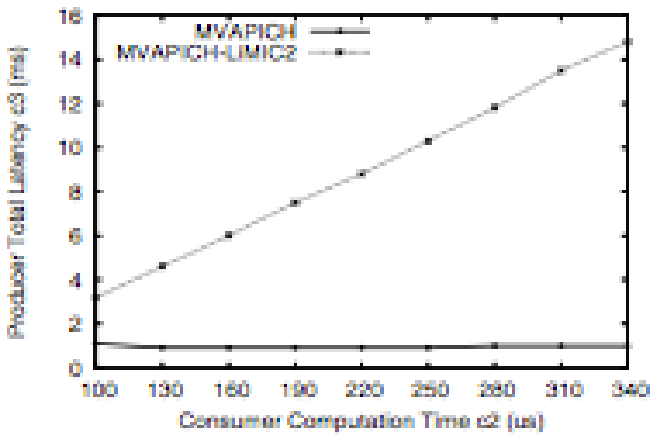
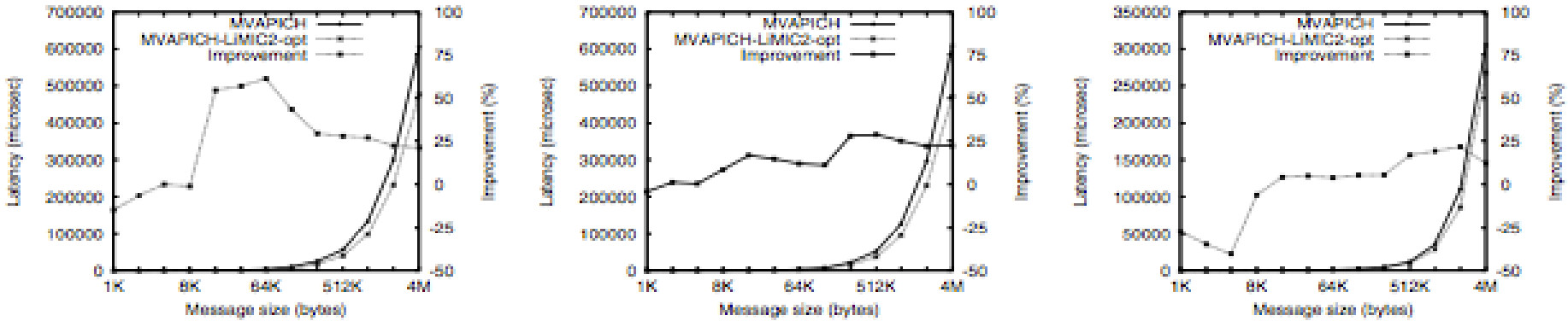


Figure 6. Impact of Process Skew

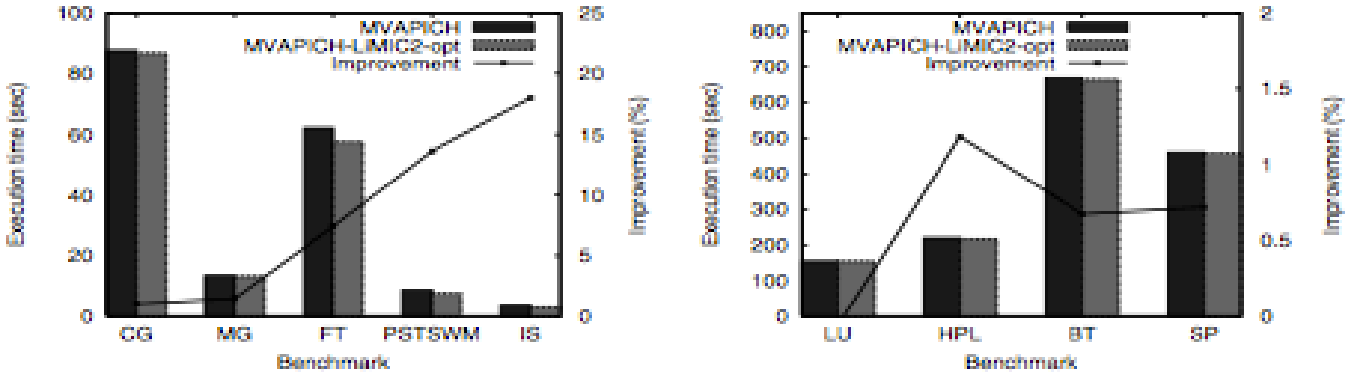
Hybrid Approach

- Topology aware threshold
  - Shared cache : 32KB
  - Intra-socket : 2KB
  - Inter-socket : 1KB
- Skew aware threshold
  - Use shared memory when skew, LiMIC2 otherwise
  - Detect skew by keeping track of length of unexpected queue at receiver side
  - If length is larger than some threshold, skew occurs, send control msg to sender to switch to shared memory



(a) MPI_Alltoall (b) MPI_Allgather (c) MPI_Allreduce

Figure 8. Collective Results (Single Node 1x8)



(a) (b)

Figure 9. Application Performance (Single Node 1x8)

Summary

- Run benchmarks to analyze the advantages and limitations of shared memory and OS kernel assist
- Proposed topology and skew aware threshold to build an optimized hybrid approach
- Evaluate the impact of this approach, performance of MPI collective operations improved by up to 60%, applications up to 17%