Argonne
NATIONAL
LABORATORY

... for a brighter future

Implementation and Shared-Memory Evaluation of MPICH2 over the Nemesis Communication Subsystem

Darius Buntinas, Guillaume Mercier and William Gropp

U.S. Department of Energy

A U.S. Department of Energy laboratory managed by The University of Chicago

Motivation

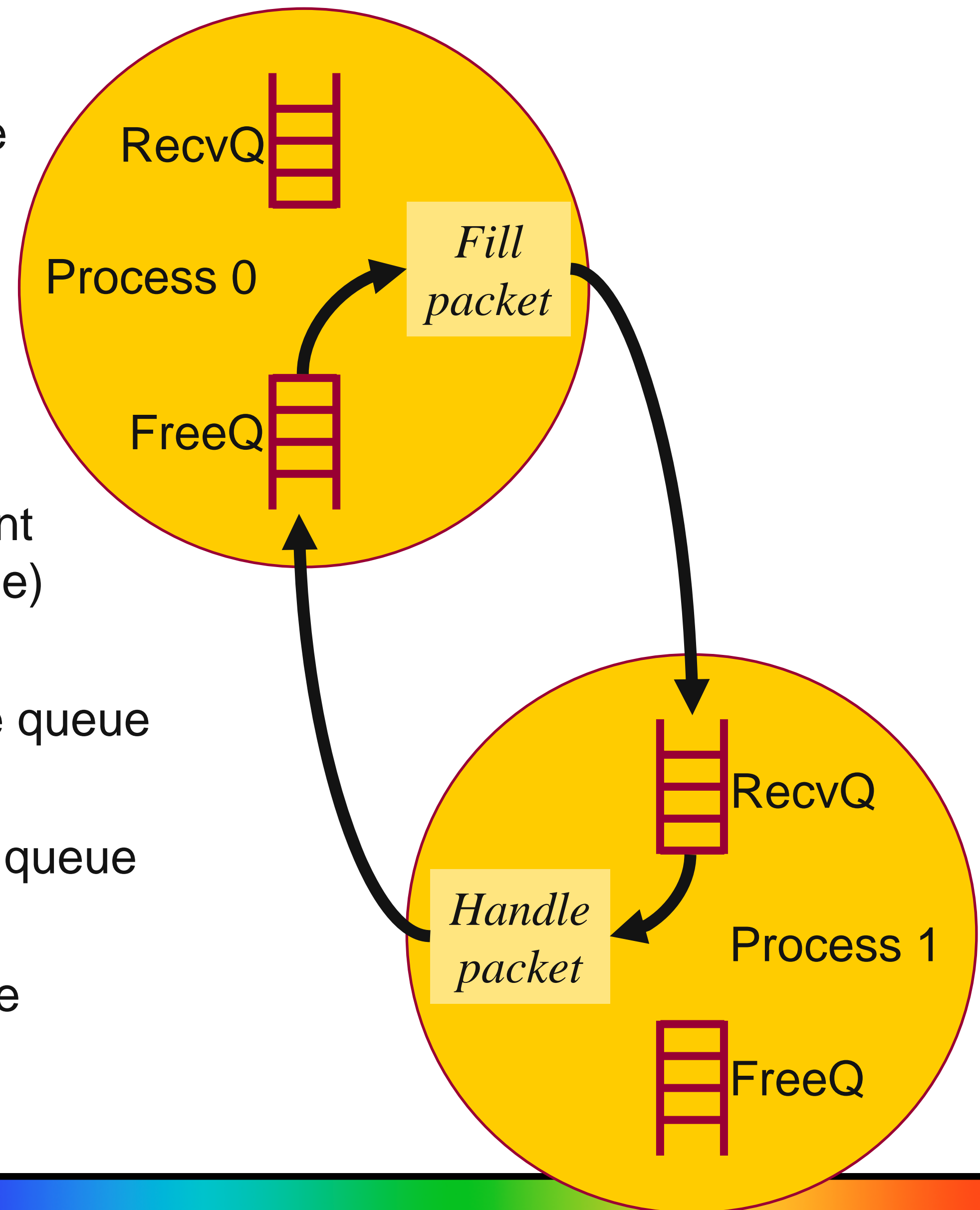
- We were surprised by the number of instructions MPICH2:shm required to perform a send-recv
  - What's the minimum number of instructions required?
  - How fast can we make intranode communication?
- Started by implementing an efficient communication subsystem: Nemesis
  - Efficient scalable intranode communication
  - Internode communication through multiple networks
  - Presented at CCGrid06
- Ported MPICH2 over Nemesis
- Incremental optimization of MPICH2 layers

Outline

- Nemesis Overview
- Porting MPICH2 over Nemesis
  - Nemesis Channel
  - Optimizations
- Performance Evaluation
- Conclusion

Nemesis Overview: Intranode Communication

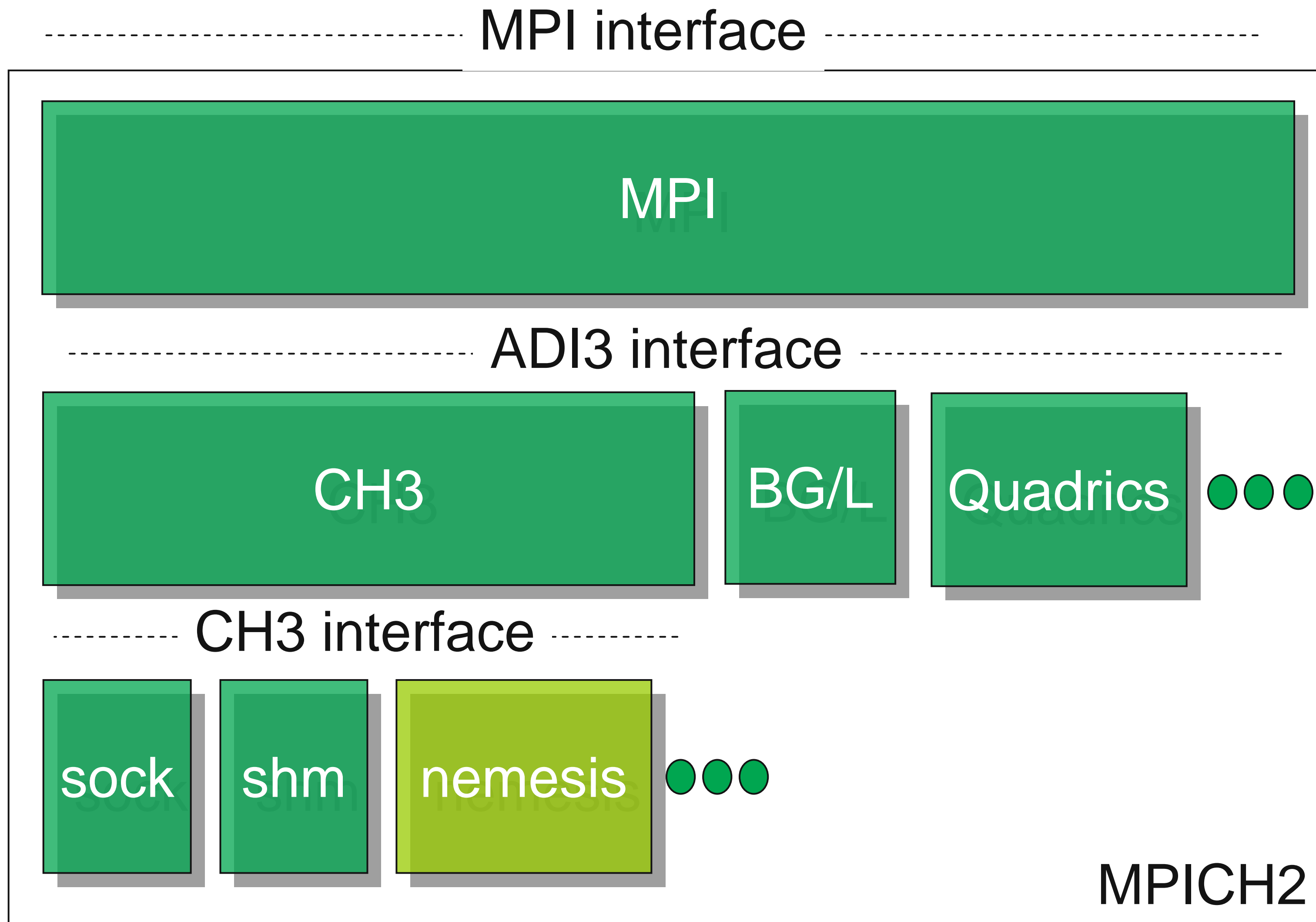
- Each process has a single lock-free *receive queue*
  - Only one queue to poll, not one per connection
  - Very scalable
- To send a message
  - Sender dequeues a free element from a *free queue* (also lock-free)
  - Fills element with message
  - Enqueues on receiver's receive queue
- To receive a message
  - Dequeue element from receive queue
  - Process message in element
  - Enqueue element on free queue



Nemesis Overview: Optimizations

- Fastboxes
  - Bypass Nemesis receive queue
  - Pair of buffers per pair of processes
  - Not scalable, but intended for SMPs with small number of processors
- Optimized placement *head* and *tail* pointers
  - Reduced L2 cache misses
- Memory copy
  - Prefetching and nontemporal store instructions

Porting MPICH2 over Nemesis



Nemesis Channel: Send

- Basic send
  - CH3 calls the channel's send function
    - *VC identifying the destination*
    - *IOV describing the data*
    - *Request object corresponding to the send*
  - Nemesis channel
    - *Dequeues a element from free queue*
    - *Copies data described by the IOV into the element*
    - *Enqueues the element on the appropriate recv queue*
    - *Once data described by IOV has been sent*
      - Upcall to reload the IOV or mark request as complete
- Messages larger than a element use multiple elements
- If no free element is available
  - IOV is saved in the request
  - Request is queued in send queue

Nemesis Channel: Receive

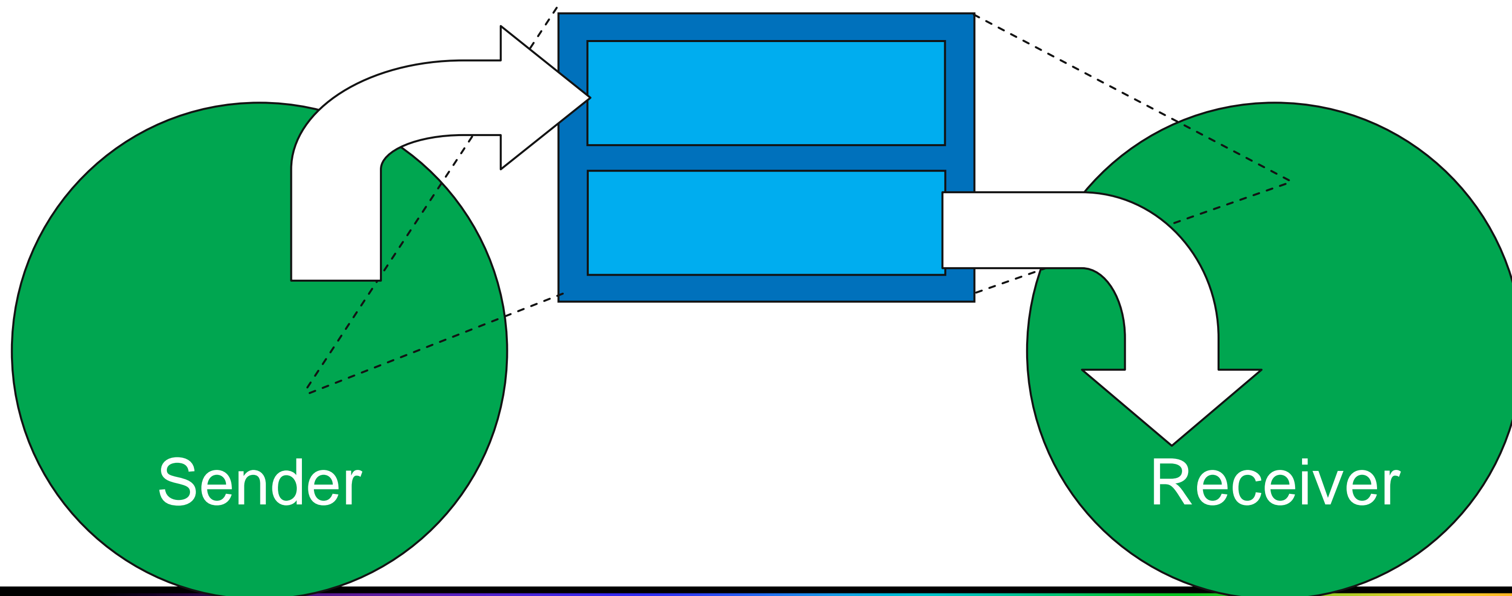
- Receive a message
  - Poll receive queue and *active* fastboxes
    - *Active fastbox: fastbox of process for which there is a posted recv*
  - Upcall to CH3 with pointer to message header
    - *Returns request and IOV describing dest buffer*
  - Channel copies data from element to dest buffer
  - Upcall to CH3 to reload IOV or mark request as complete
- If IOV describes more data than what's in the element
  - Save IOV in request and save request as pending receive
  - Continue when another element from the same source is received
- Queue elements are a limited resource and should be freed as soon as possible
  - Unexpected messages are copied out of element to a temp buffer

Optimization: Large Message Transfer Using Rendezvous

- MPICH2 uses rendezvous to transfer large messages
  - Original implementation: channel was oblivious to rendezvous
    - *CH3 sent RTS, CTS, DATA*
    - *Large messages would be sent through queue*
- Queues may not be the most efficient mechanism to transfer large data
  - E.g., network RDMA, inter-process copy mechanism, copy buffer
- Developed LMT interface to support various mechanisms
  - Sender transfers data (put)
  - Receiver transfers data (get)
  - Both sender and receiver participate in data transfer
- Modified CH3 to use LMT
  - Works with rendezvous protocol

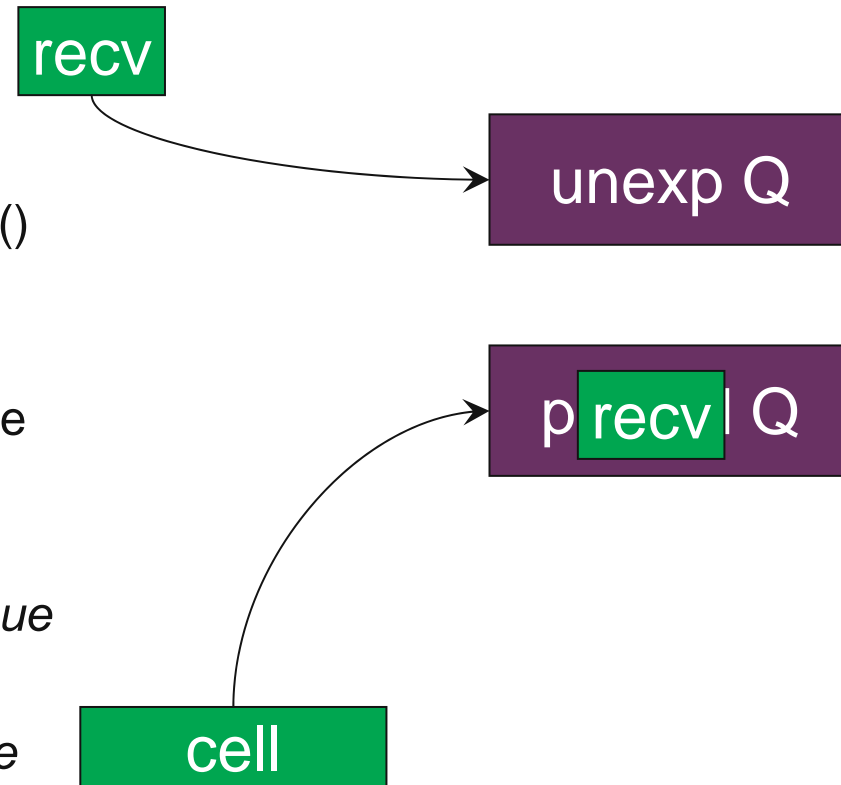
Optimization: LMT for Intranode Communication

- For intranode, LMT copies through buffer in shared memory
- Sender allocates shared memory region
  - Sends buffer ID to sender in RTS packet
- Receiver attaches to memory region
- Both sender and receiver participate in transfer
  - Use double-buffering
- ~130 MiBps improvement for large messages



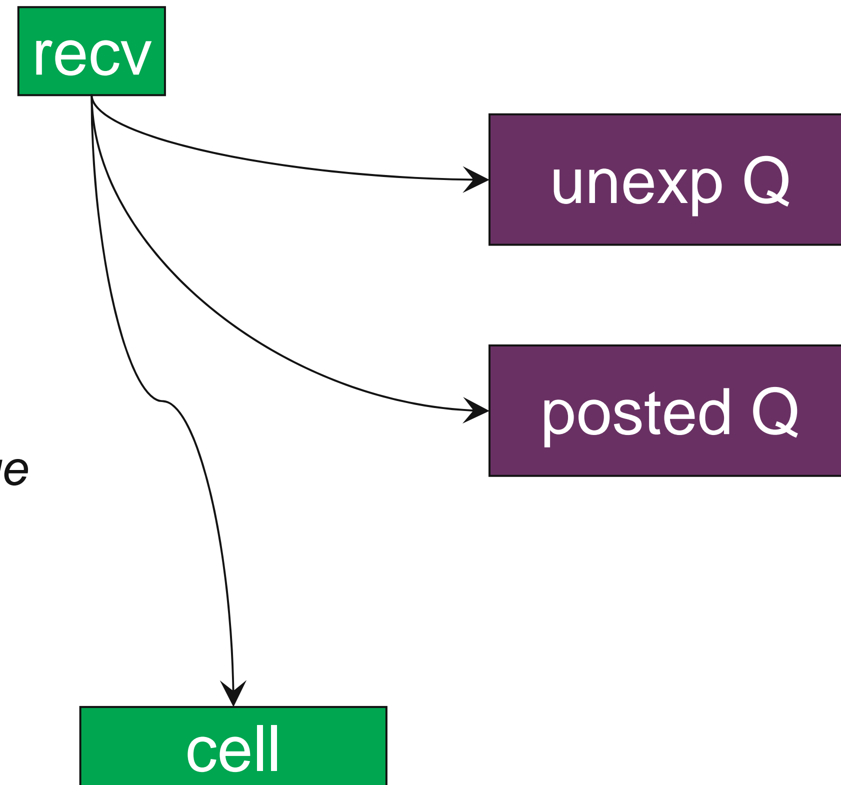
Optimization: Bypassing the Posted Receive Queue

- Problem: MPI_Recv() is called and the matching message is waiting in the Nemesis recv queue
- Original CH3 implementation of MPI_Recv()
  - Check unexpected queue for matching message
  - If not found, enqueue on posted receive queue
  - Call progress function
    - *Gets cell from Nemesis receive queue*
    - *Checks posted receive queue*
    - *Dequeue from posted receive queue*



Optimization: Bypassing the Posted Receive Queue 2

- Optimized implementation
  - Check unexpected queue for matching message
  - If not found, and there is no previously posted recv request that would match this recv request
    - *Get cell from Nemesis receive queue*
    - *Match recv with cell*
    - *Receive message*
- 18% improvement for small messages



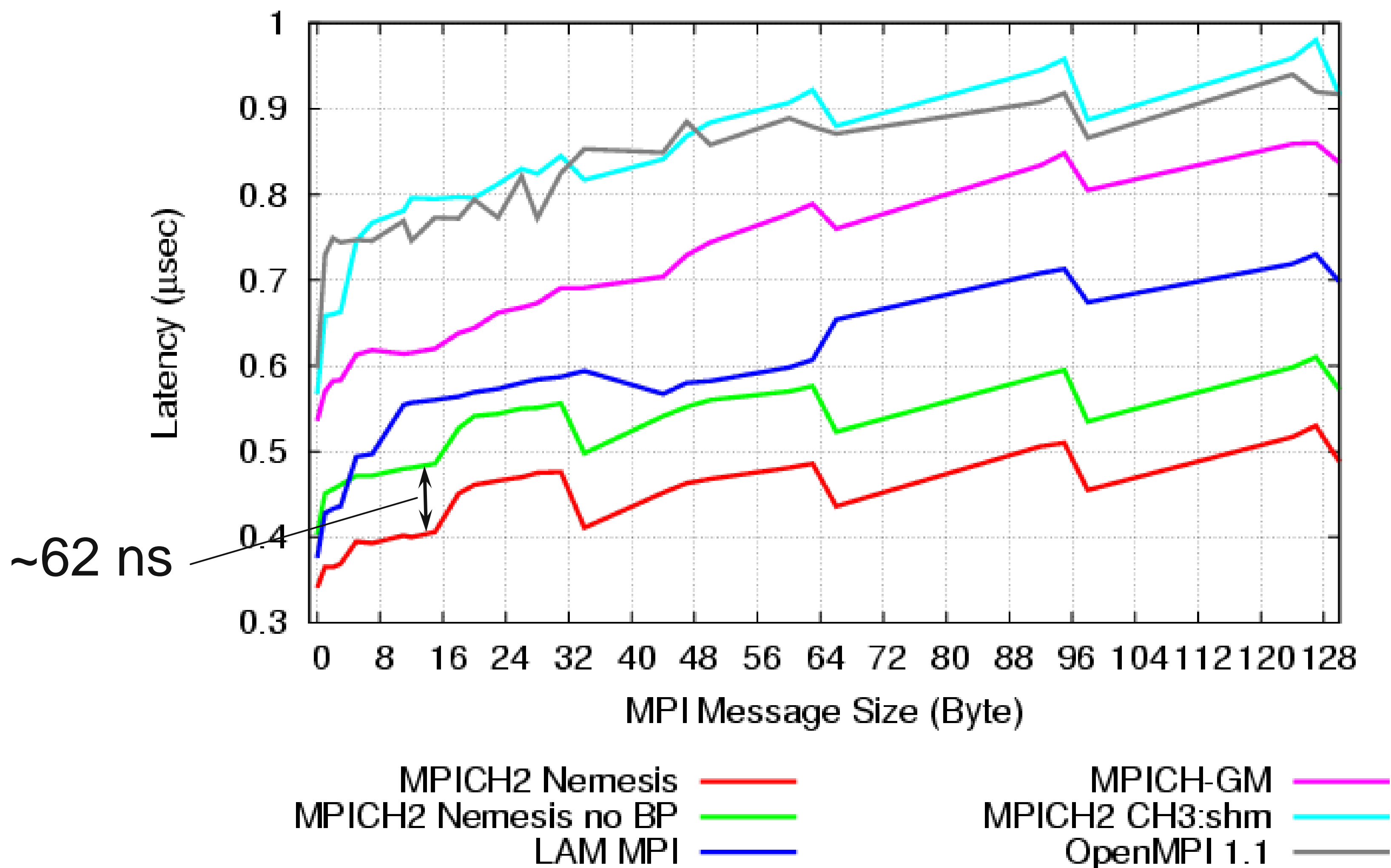
Performance Evaluation

- Latency and bandwidth
- Instruction count
- Halo benchmark
- NAS benchmarks

- 2.4 GHz dual-proc dual-core Opteron 280
- For NAS benchmarks
  - SGI Altix 350: 16 way NUMA 1.4 GHz Itanium 2

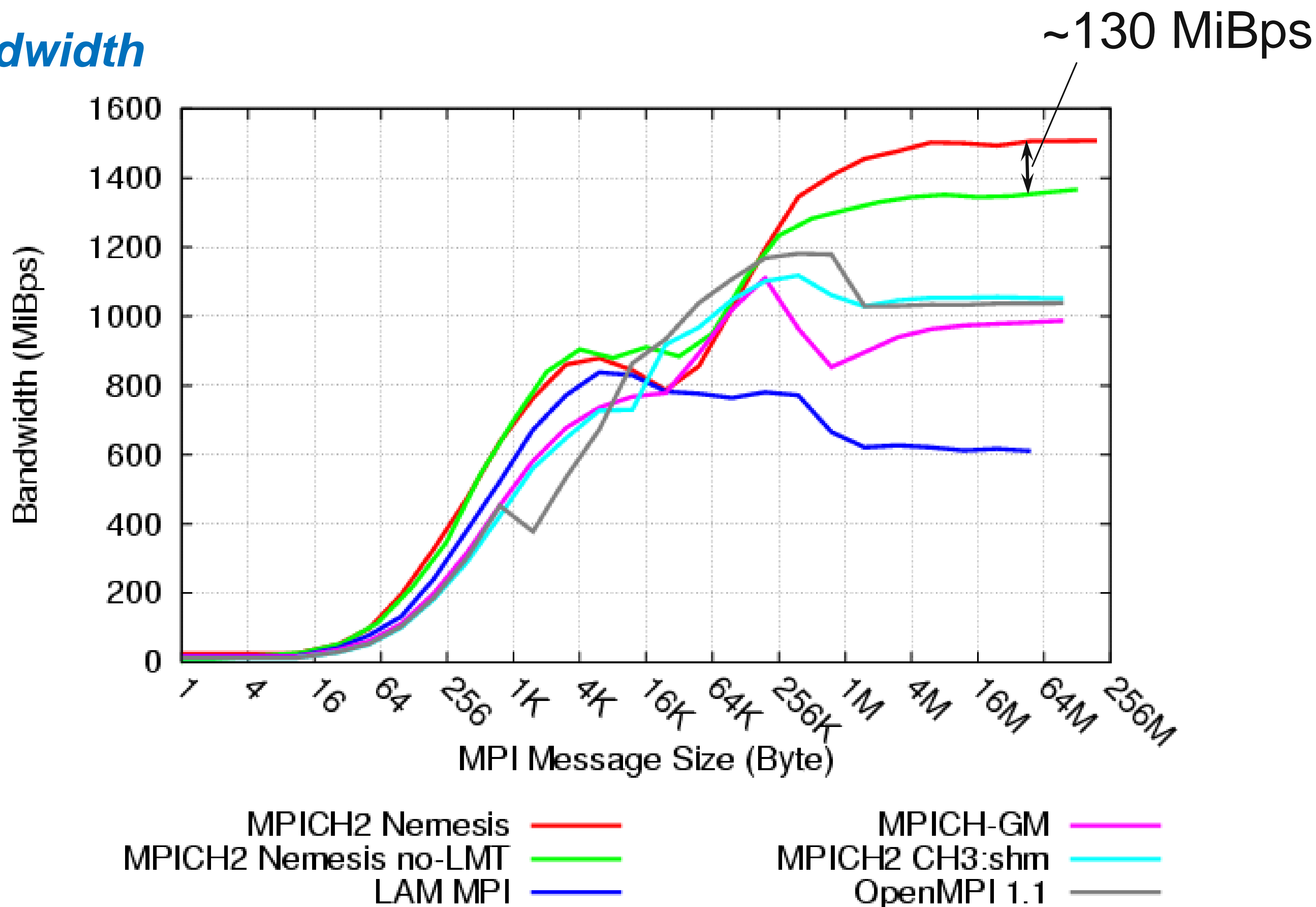
- All implementations were compiled with `-O3`
- MPICH2 configured with `--enable-fast`
  - Disables runtime error checking
- Open MPI configured to
  - Disable runtime error checking
  - Disable support for heterogeneous clusters

MPI Latency



- Nemesis: 341ns 0-byte MPI message latency
- 1.7 to 2.0 factor of improvement over shm channel

Bandwidth



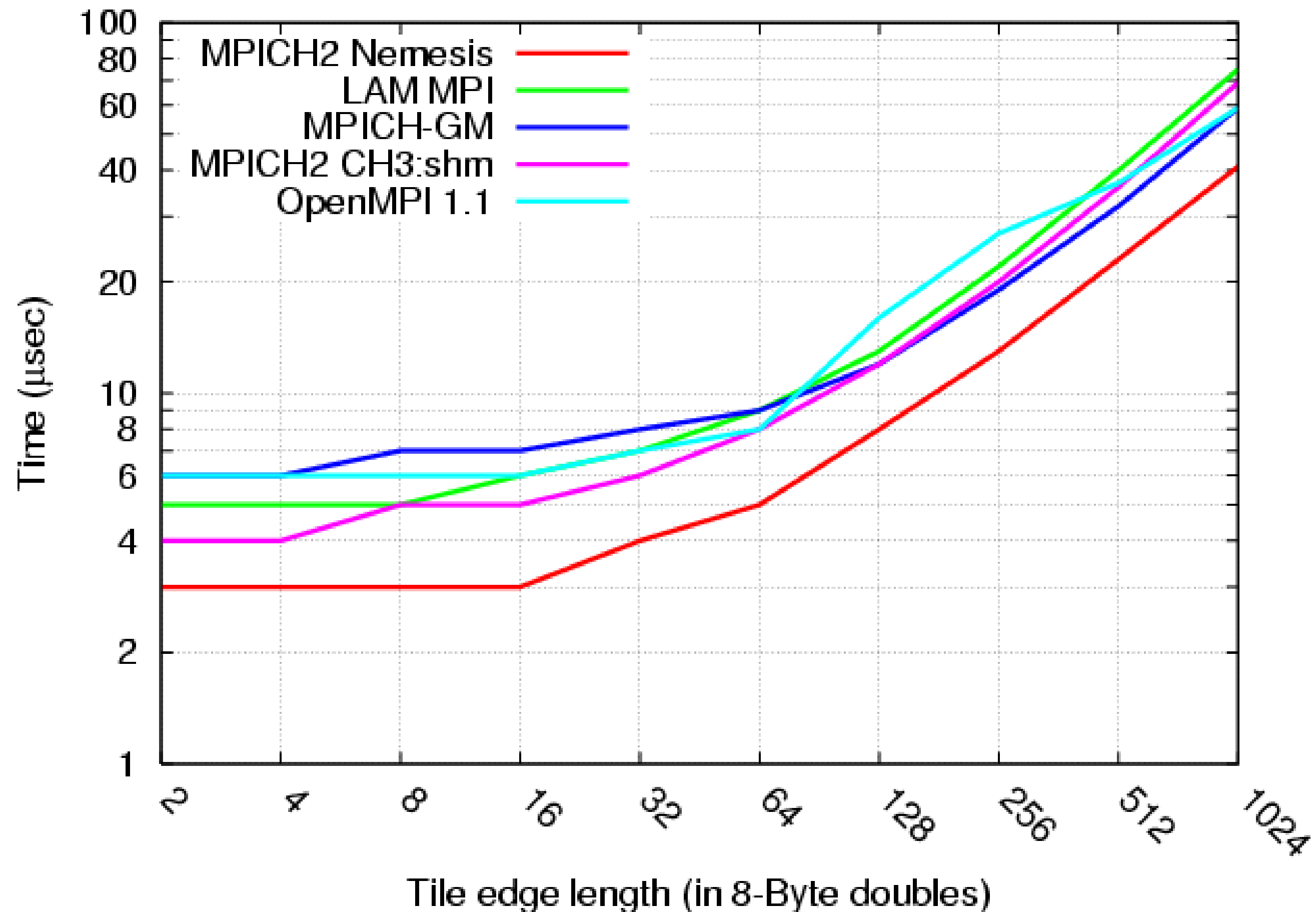
- Nemesis: over 1,500 MiBps MPI bandwidth
- Up to 1.4 factor of improvement over shm for large messages

Instruction Count

- Send-recv of 8-bytes
- Count instructions for MPI_Send
- Count instructions for MPI_Recv
  - Delay before calling MPI_Recv to ensure message has arrived

MPI Implementation	`MPI_Send`	`MPI_Recv`	Total	Improvement
Open MPI 1.1	550	1,745	2,295	78%
MPICH-GM	455	617	1,072	53%
LAM MPI	436	472	908	45%
MPICH2 CH3:shm	311	748	1,059	53%
MPICH2 Nemesis no BP	241	712	952	48%
MPICH2 Nemesis	**241**	**259**	**500**	——

Halo Benchmark



- Performs 2D "halo" exchange
- Sensitive to latency and bandwidth
- Benchmark correlates well with layered ocean model

NAS

- Ran on a 16 processor Altix at OSC
- Similar performance for all MPI implementations
- NAS benchmarks are not that sensitive to latency and bandwidth
  - Latency of MPI implementations differ by at most ~0.5 μs

Conclusion

- Presented a new implementation of MPICH2 over Nemesis
- Evaluated performance
  - Zero-byte latency of 341 ns
  - 128-byte latency of just over 500 ns
  - Peak bandwidth was over 1,500 MiBps
    - *Medium message bandwidth could be improved*
  - 500 instructions to send and receive 8-byte message
  - Outperformed other implementations in Halo benchmark
  - NAS benchmarks did not show much difference
- MPICH2 Nemesis would be suitable for small-grained applications sensitive to latency and bandwidth
- Continuing and future work
  - Implement a Nemesis ADI3 device
  - Continue optimizations
  - Checkpoint fault-tolerance
  - Optimize collectives and one-sided operations to take advantage of shared-memory

For Further Information

http://www.mcs.anl.gov/~buntinas

buntinas@mcs.anl.gov

MPICH2 homepage

http://www.mcs.anl.gov/mpi/mpich2

(or google mpich2)

- Nemesis is available in MPICH2 distributions starting with 1.0.4
  (Without LMT or recv queue bypass optimizations)
  - Configure using `--with-device=ch3:nemesis`