Efficient and Truly Passive MPI-3 RMA Synchronization Using InfiniBand Atomics

Mingzhe Li Sreeram Potluri Khaled Hamidouche

Jithin Jose Dhabaleswar K. Panda

Network-Based Computing Laboratory
Department of Computer Science and Engineering
The Ohio State University

Outline

- Motivation
- Problem Statement
- Current MPI Passive Synchronization Implementations
- Efficient and Truly Passive Synchronization scheme
- Performance Evaluation
- Conclusion and Future Work

MPI Remote Memory Access (RMA) Model

- Minimizing communication overheads is key as applications scale to millions of processes/cores
- RMA model offers an alternative to Send/Recv based message passing model
  - Communication Epochs
    - Period between 2 synchronizations
    - One-sided communication
    - Windows area
- Promises better latency hiding, asynchronous progress and reduced synchronization overheads
- MPI-3 offers several extensions to provide more flexibility

MPI-3 RMA Passive Synchronization

- RMA offers flexible synchronization alternatives
  - Active: Fence and Post-Wait/Start-Complete
  - Passive: Lock/Unlock, Lock_all/Unlock_all
  - Shared/Exclusive (Lock/Unlock) and (Only Shared) (Lock_all/Unlock_all)
- Passive synchronization does not require involvement of target process
  - Less synchronization
  - Better overlap
- However, current implementations are based on two-sided operations
- Desirable to have a truly one-sided design offering
  - Performance (no remote polling)
  - Fairness (FIFO)

InfiniBand

- Interconnect of choice in high performance systems
- Offers RDMA
  - Read/Write
  - Atomics (Fetch-and-Add, Compare-and-Swap)
- Atomics are supported only on 64bit values
- Important to take advantage of these features to design the Passive synchronization

Outline

- Motivation
- Problem Statement
- Current MPI Passive Synchronization Implementations
- Efficient and Truly Passive Synchronization scheme
- Performance Evaluation
- Conclusion and Future Work

Problem Statement

Can a truly passive locking mechanism be designed for InfiniBand Clusters ?

How can this design provide :

- Performance (no remote Polling)
- Fairness (FIFO => no starvation)

Can the new locking mechanism benefits the performance of applications ?

Outline

- Motivation
- Problem Statement
- Current MPI Passive Synchronization Implementations
- Efficient and Truly Passive Synchronization scheme
- Performance Evaluation
- Conclusion and Future Work

Existing Passive Synchronization Semantics over IB

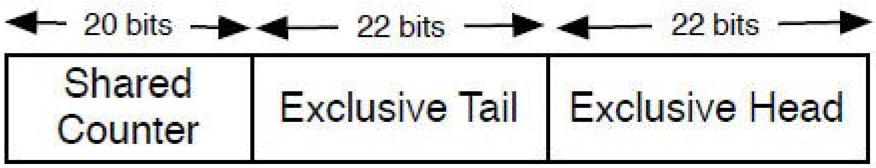
	Shared	Exclusive	Limitations
State-of-the art MPI Libraries	Send/Recv	Send/Recv	Restrict asynchronous progress
Jiang et.al (Compare_and_swap)	Atomics	Atomics	High network Traffic due to remote polling
Jiang et.al (MCS based)	--	Atomics/Put	Shared mode of locking is not handled
Santhanaraman et.al	Send/Recv	Atomics	Restrict asynchronous progress. High network Traffic

Outline

- Motivation
- Problem Statement
- Current MPI Passive Synchronization Implementations
- Efficient and Truly Passive Synchronization scheme
- Performance Evaluation
- Conclusion and Future Work

Lock Data Structures - 1

- Our locking mechanism depends on IB atomics to implement shared and exclusive mode of locking
- IB requires 64 bits buffer for atomic operations
- This 64 bits region is divided into three parts to handle different lock request



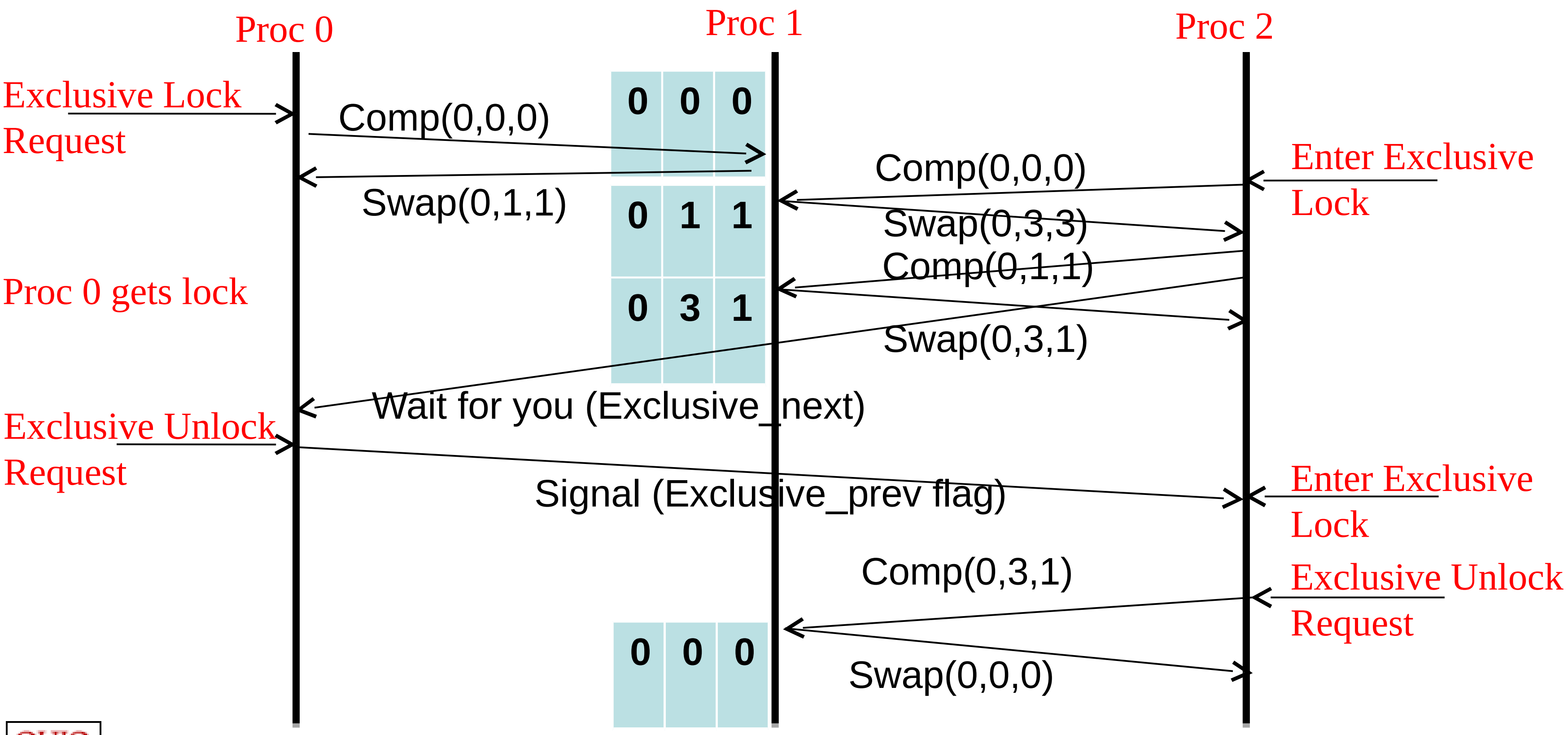
  - Shared Counter: count of the processes that own or have requested a shared lock
  - Exclusive Tail: rank of the process which is tail of the distributed queue
  - Exclusive Head: rank of the process which is head of the distributed queue

Lock Data Structures - 2

- In order to handle all possible lock requests, a distributed lock queue is maintained to ensure FIFO and avoid remote polling
- Data structures to implement the distributed lock queue:
  - Wait-for array: used when shared lock comes after exclusive lock. This exclusive lock knows the list of processes that request shared lock after it
  - Signal-to array: used when shared lock comes after exclusive lock. This exclusive lock wakes up pending processes that are waiting for the shared lock
  - Exclusive-next: two element integer array. Used by processes requesting exclusive lock to form a distributed lock queue
  - Exclusive-prev: one integer flag. Used by a process unlocking an exclusive lock to wake up another process waiting for an exclusive lock

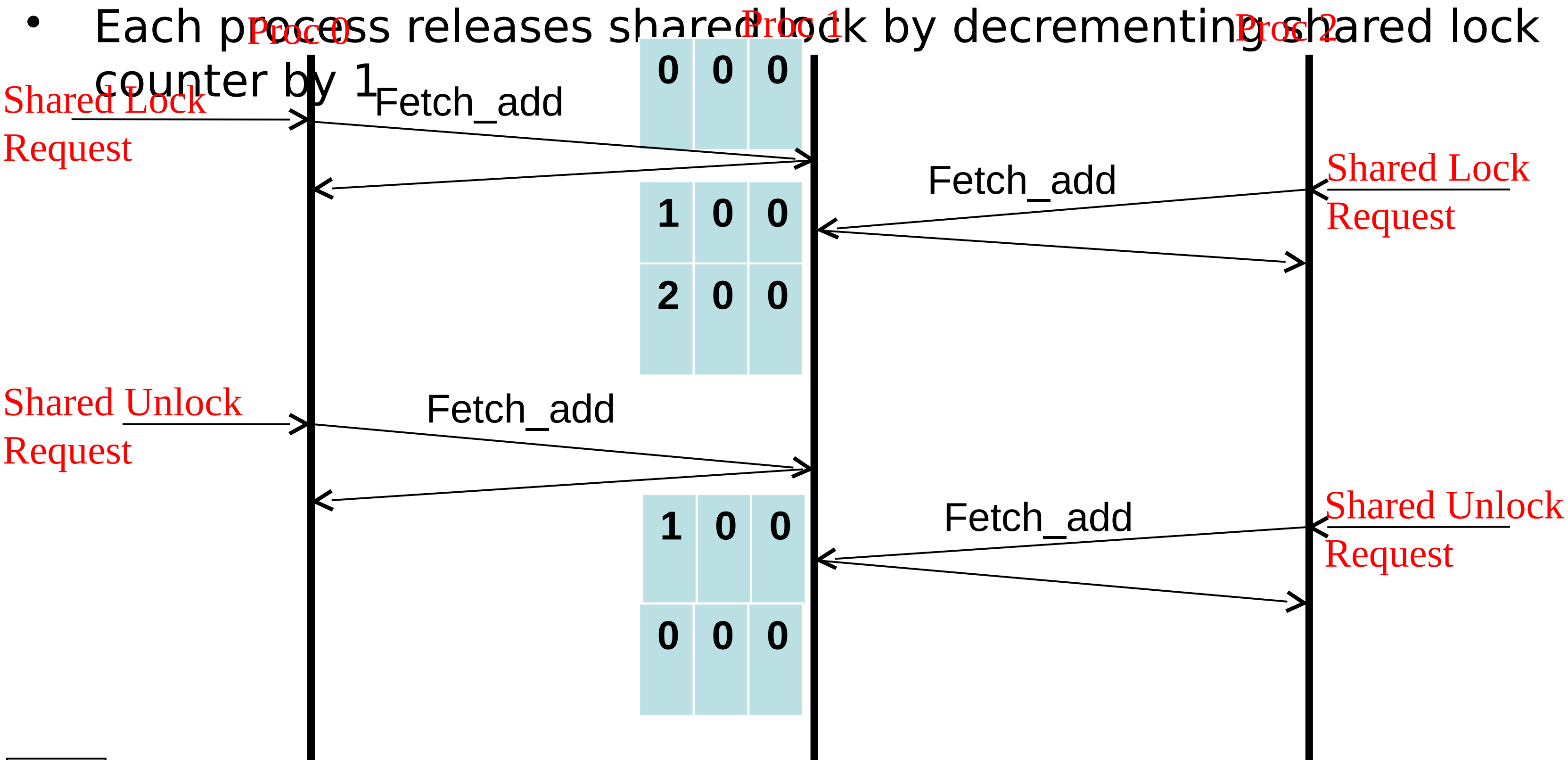
Exclusive Locking Only

- RDMA operations: compare_and_swap and Put
- Lock requests are ordered in distributed queue
- Exclusive locks are granted in FIFO order



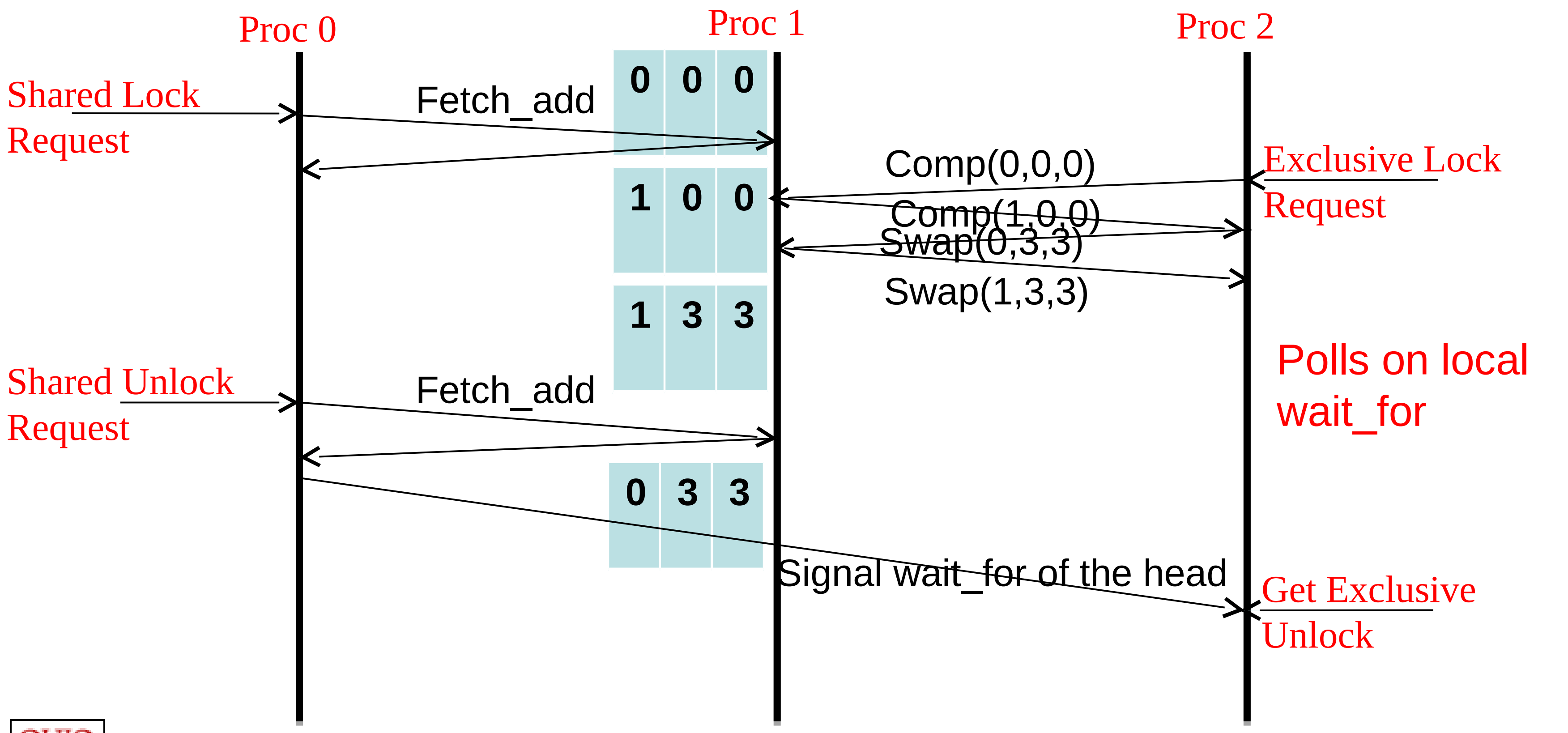
Shared Locking Only

- Atomic operation : Fetch_and_add. To decrement we add the MAX value
- Each process requires shared lock is able to get it after its atomic operation completes
- Each process releases shared lock by decrementing shared lock counter by 1



Interleaved Shared and Exclusive Locking

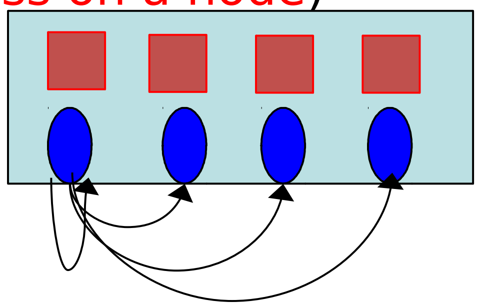
- Shared followed by exclusive lock: Process gets exclusive lock after all previously granted shared locks have been releases.
- Exclusive followed by shared lock: Process gets shared lock after the previous exclusive lock releases its lock



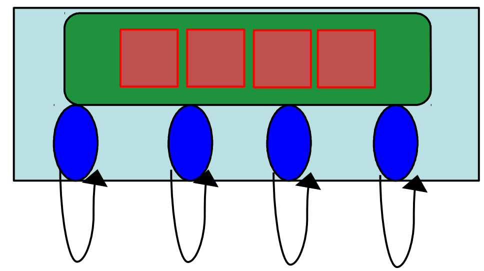
Intra-node Locking Design

- For intra-node locking, native loopback that needs a number of queue pairs

(p+(p*(p-1))/2) is not an efficient implementation (P= number of process on a node)



(P+(P*(P-1))/2) QPs



P QPs

- If the lock-unlock 64 bits data structures are allocated in the shared memory region, the number of queue pairs used is decreased from (p+(p*(p-1))/2) to p
  - Based on the intra-node locking design, if one process wants to acquire a lock from other process in the same node, it issue atomic operation to itself (loopback)
  - The locking/unlocking mechanisms are the same as discussed earlier

Lock_All/Unlock_All Implementation

- Lock_all and Unlock_all introduced in MPI-3 use only shared lock.
- In our design, they are implemented based on the lock/unlock mechanism discussed eariler.
- If **MPI_ MODE_ NOCHECK** is used, then they are implemented as No_Op
- Inside Lock_all function, call win_lock is explicitly called for every processes in the communicator
- For Unlock_all, the same mechanism is used to call unlock for every process in the communicator

Outline

- Motivation
- Problem Statement
- Current MPI Passive Synchronization Implementations
- Efficient and Truly Passive Synchronization scheme
- Performance Evaluation
- Conclusion and Future Work

Experimental Setup

- Cluster A
  - Xeon Dual quad-core processor (2.67 GHz) with 12GB RAM
  - Mellanox QDR ConnectX HCAs (32 Gbps data rate) with PCI_Ex Gen2 interface
- Software stack
  - Implemented on MVAPICH2-1.9 will be in future releases
- http://mvapich.cse.ohio-state.edu Latest releases **: MVAPICH2-2.0a**
- High Performance open-source MPI Library for InfiniBand, 10Gig/iWARP, and RDMA over Converged Enhanced Ethernet (RoCE)
  - MVAPICH (MPI-1) ,MVAPICH2 (MPI-2.2 and MPI-3.0), Available since 2002
  - MVAPICH2-X (MPI + PGAS), Available since 2012
  - Used by more than 2,077 organizations (HPC Centers, Industry and Universities) in 70 countries

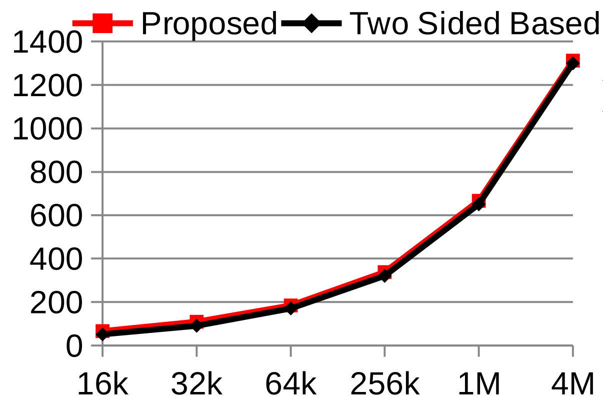
MPI_Get with Lock-Unlock

One MPI_Get Latency

Eight MPI_Get Latency

Latency (us)

Latency (us)



Latency (us)

Message size (Bytes)

Message size (Bytes)

Message size (Bytes)

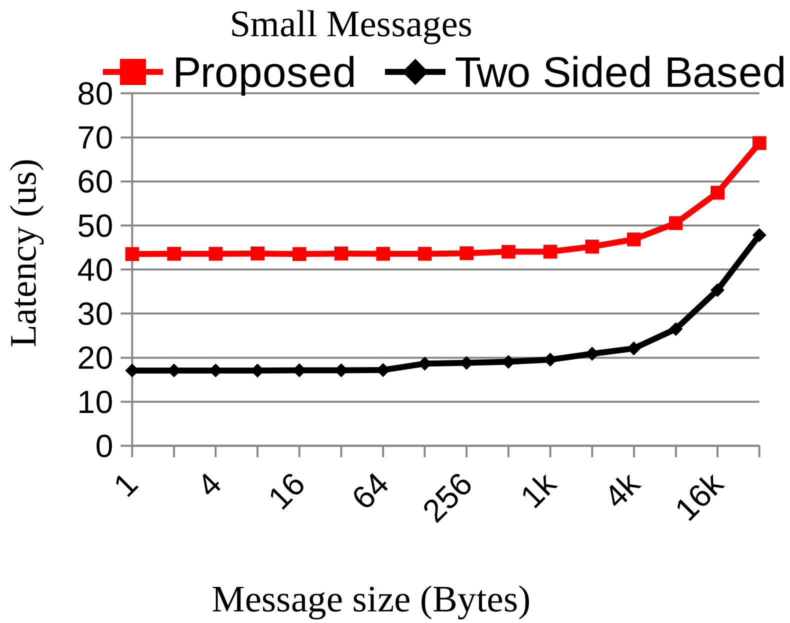
- For one MPI_Get latency:

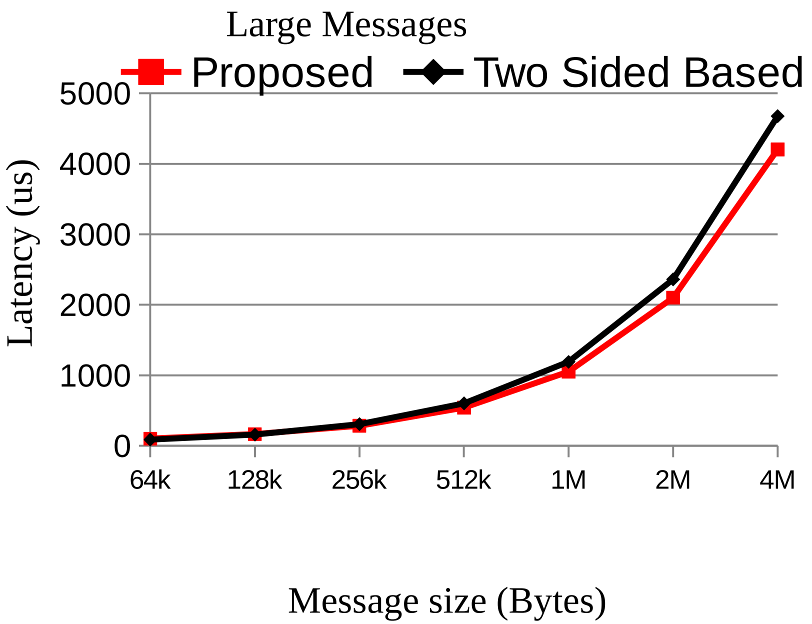
  - Small messages: atomic based design incurs an overhead compared to two-sided based design : two-sided design coalesces the 3 operations in one message

  - Large messages: Amortized the overhead and have similar performance

- For eight MPI_Get latency, the overhead is amortized and we see similar performance with both designs

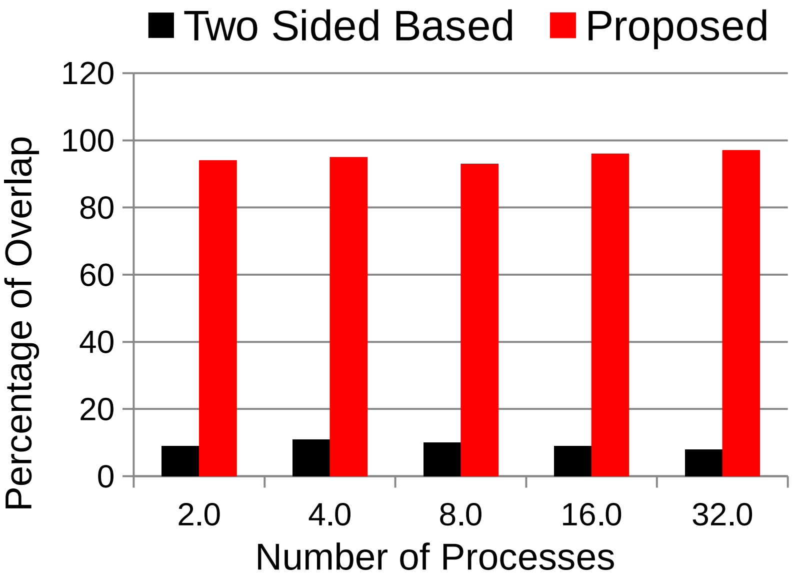
MPI_Get with Lock_all-Unlock_all



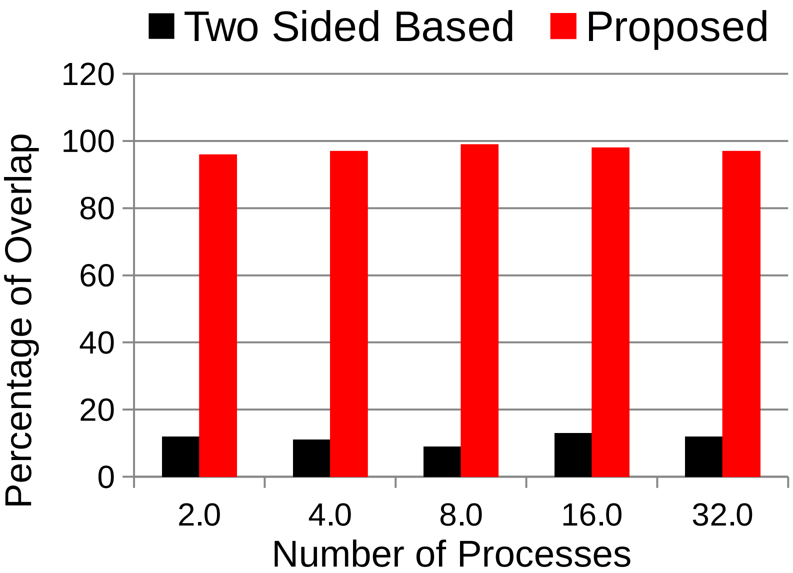


- We see the same trend for small messages
- Our design could benefit large messages by asynchronously issuing lock/unlock requests from different processes

Overlap Benchmark



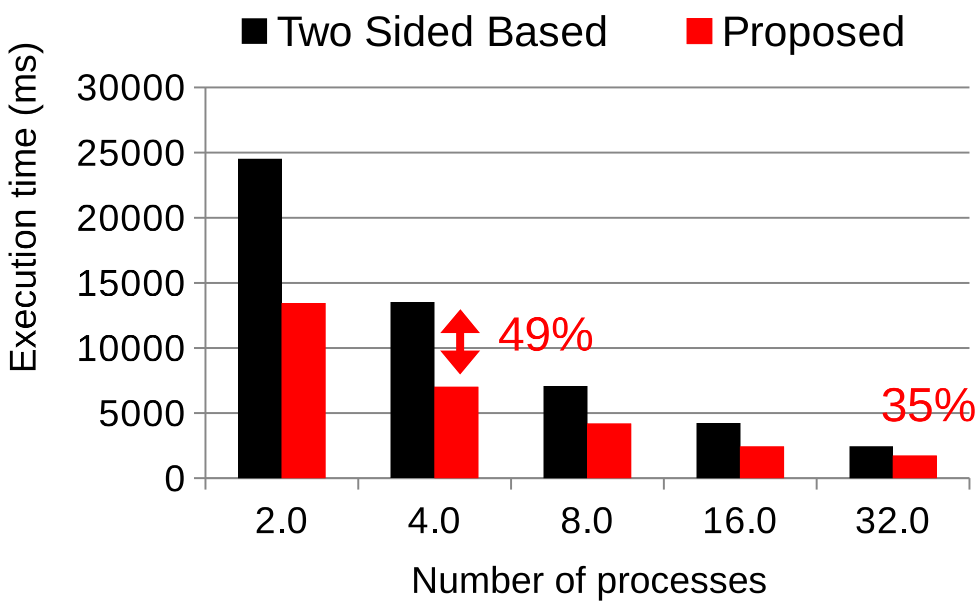
Communication Overlap-Lock



Communication Overlap-Lock_all

- Our design achieves almost optimal computation/communication overlapping

Splash LU Kernel



- This modified version of Splash LU Kernel does dense LU factorization
- Our design outperforms the two-sided approach by a factor or 49% and 35% on 4 and 32 processes

Conclusion and Future Work

- Proposed Locking mechanism to implement both shared and exclusive lock with RDMA InfiniBand Atomics:

  - No remote polling

  - FIFO order.

- Show optimal computation communication overlap
- Demonstrated up to 49% improvement using Splash LU Kernel
- Evaluate our designs with more applications/systems
- Provide RDMA based-designs for MPI-3 RMA over IB

Thank You!

{limin, potluri, hamidouc, jose, panda} @cse.ohio-state.edu

Network Based Computing Laboratory

MVAPICH

Network-Based Computing Laboratory

http://nowlab.cse.ohio-state.edu/

MVAPICH Web Page

http://mvapich.cse.ohio-state.edu/