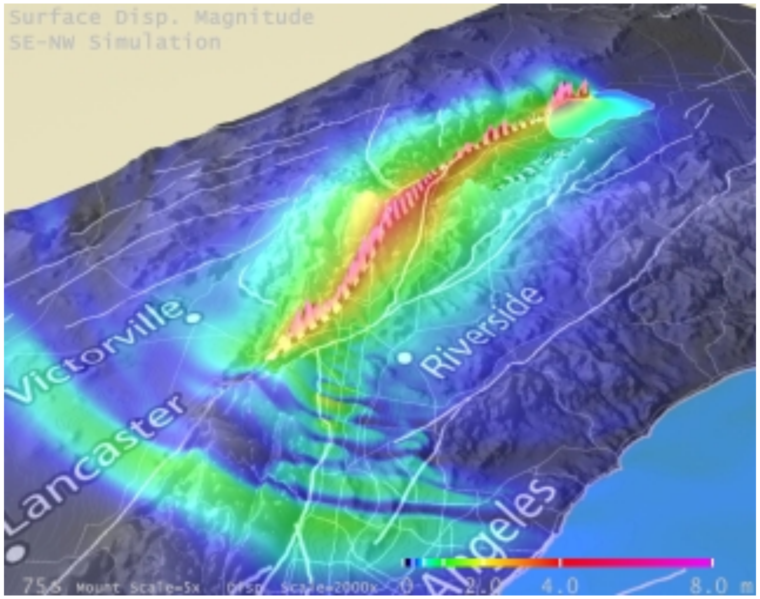
Designing Truly One-Sided MPI-2 RMA Intra-node Communication on Multi-core Systems

Ping Lai **Sayantan Sur** Dhabaleswar K. Panda

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

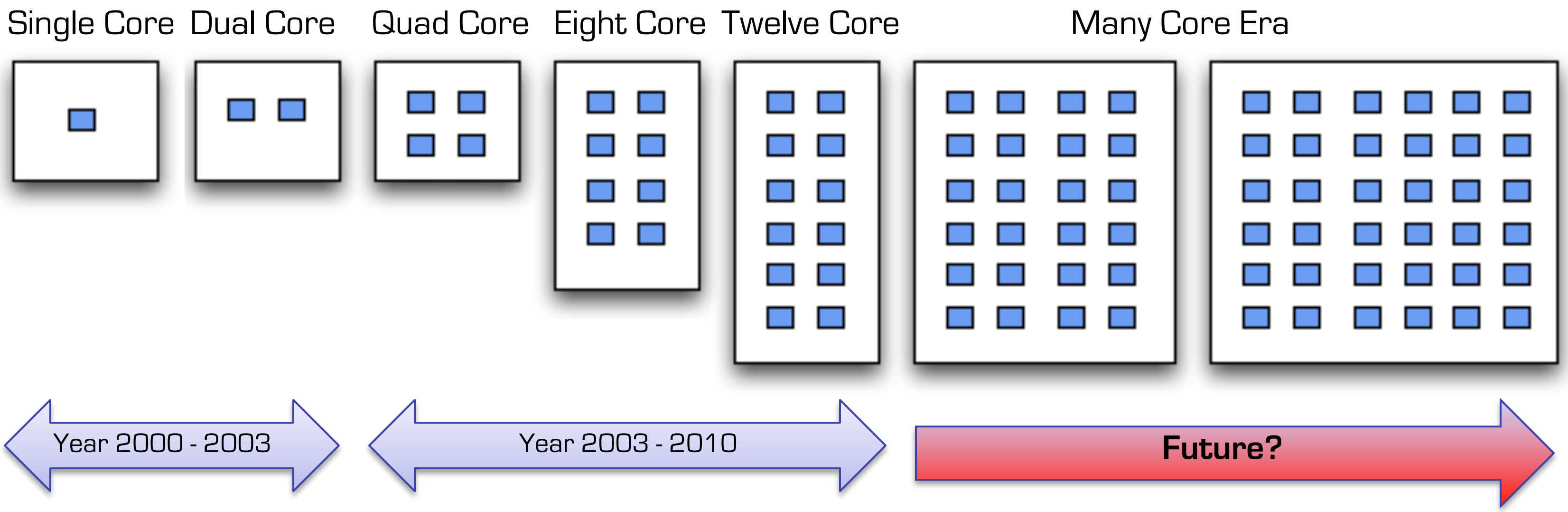
Introduction

- Scientific Applications
  - Earthquake Simulation, Weather prediction, computational fluid dynamics etc.
  - Use HPC systems to push boundaries of our understanding of nature
- Consume millions of hours on supercomputers world wide
- Most applications use MPI parallel programming model



Shakeout Earthquake Simulation
Visualization credits: Amit Chourasia, Visualization Services, SDSC
Simulation credits: Kim Olsen et. al. SCEC, Yifeng Cui et. al., SDSC

Commodity Multi-core Processors



Single Core Dual Core Quad Core Eight Core Twelve Core Many Core Era

Year 2000 - 2003

Year 2003 - 2010

Future?

- Communications inside the node (intra-node) becoming increasingly important
- Going forward, we need to deal with several issues:
  - Communication and computation overlap
  - Synchronization overheads
  - Cache misses (dependence on scarce memory bandwidth)

The promise of MPI-2 RMA

- MPI-2 RMA model holds much promise for multi-core
- *Communication and computation overlap*
  - Non-blocking data moving primitives – Put, Get, Accumulate
- *Synchronization overheads*
  - Two different synchronization methods – Active, Passive
  - Active synchronization can use sub-groups
  - Passive synchronization can help irregular patterns
- *Cache misses*
  - MPI Implementations can strive to reduce message copies and to the extent possible reduce cache misses

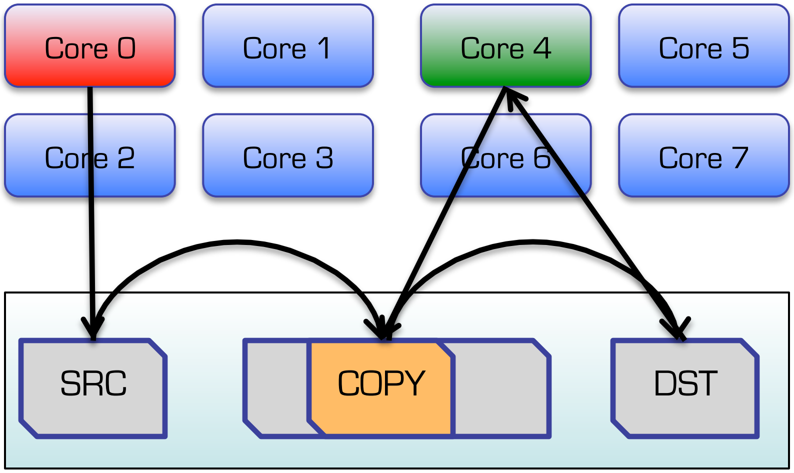
Outline

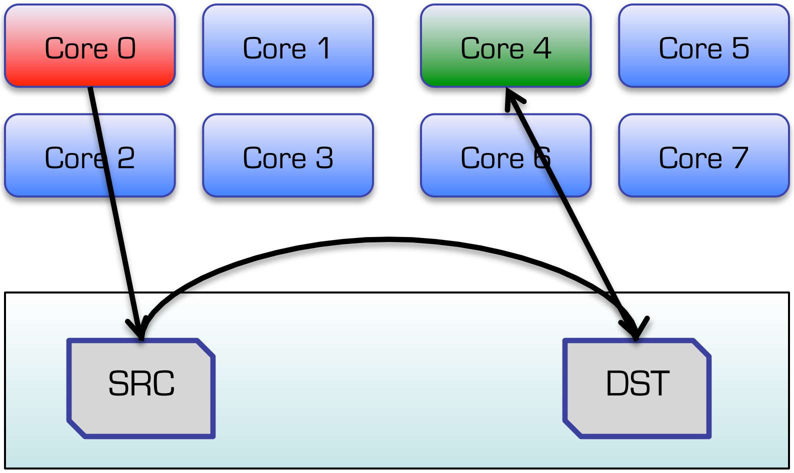
- Introduction
- Problem Statement
- Proposed Design
- Experimental Results & Analysis
- Conclusions & Future Work

The state of current MPI-2 implementations and Applications

- Scientific applications tend to evolve slowly
- Slow to adopt MPI-2
- Since not many scientific applications do not use RMA, implementers do not focus on it
  - RMA for intra-node implemented on top of two-sided
    - Portability
    - Speed of development
- Two-sided implementations do not provide promised benefits of RMA model
  - As a result application developers tend not to use it
- Deadlock!

Intra-node One-sided Communication





- User-level shared memory techniques lead to two copies
- One copy methods
  - Kernel based (LiMIC2, KNEM)
  - On-board DMA engines, such as Intel I/OAT

Problem Statement

- Can we design “true” one-sided support for MPI-2 RMA operations?
  - Can it improve communication and computation overlap?
  - Can it reduce synchronization overheads?
  - Can it reduce cache misses?
- Can real applications benefit from this true one-sided operations?

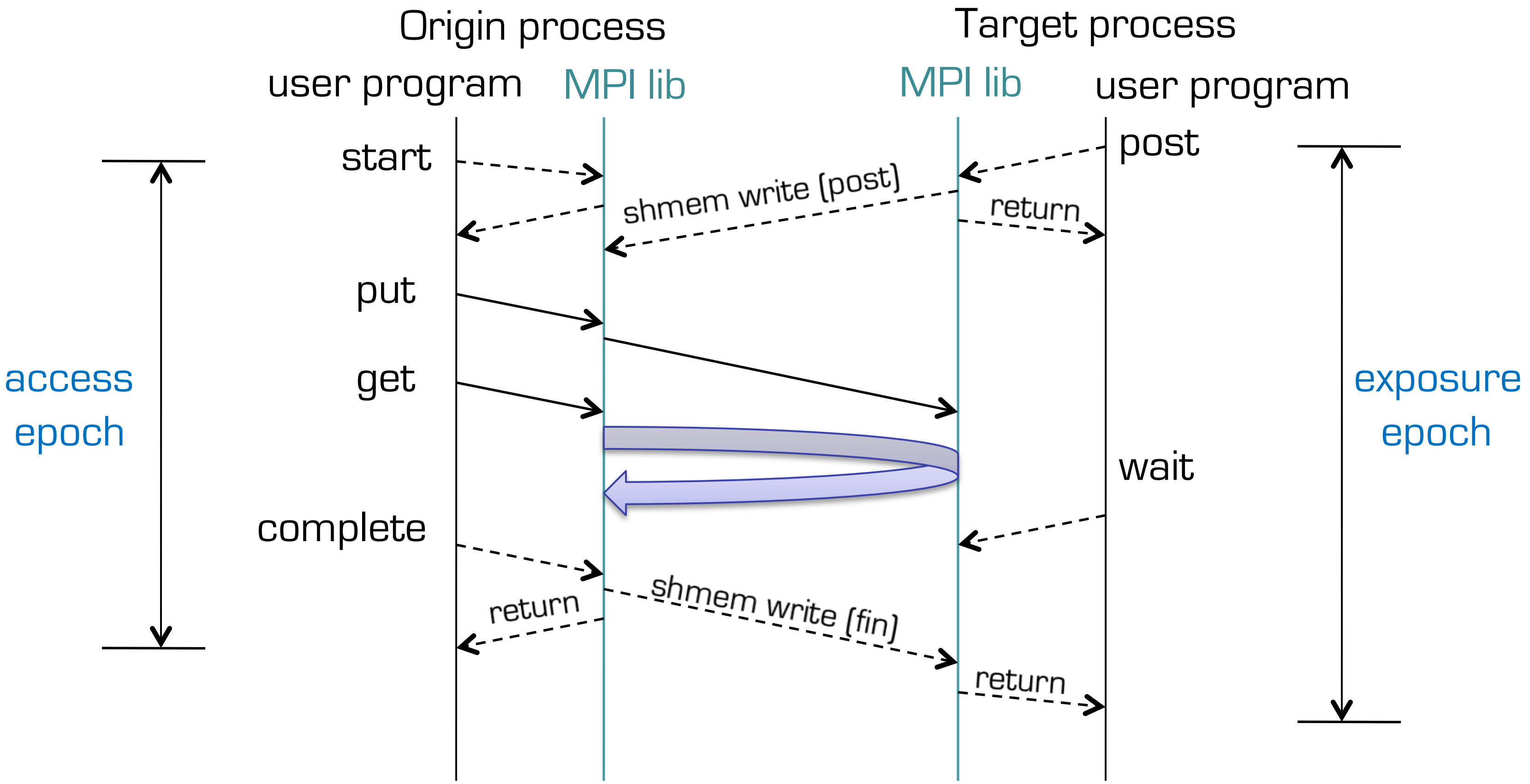
Outline

- Introduction
- Problem Statement
- Proposed Design
- Experimental Results & Analysis
- Conclusions & Future Work

Basic Approaches for Intra-node Communication

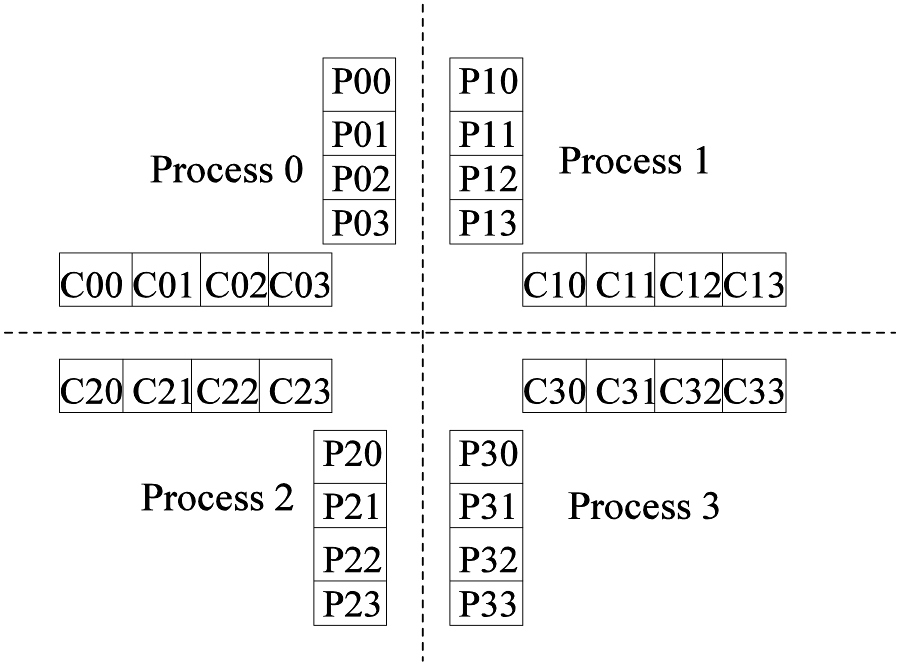
- Shared memory approach
  - Communicating processes share a buffer
  - Two copies : sender copy-in; receiver copy-out
  - Good for small messages
- Kernel assisted direct-copy approach
  - Kernel directly copies the data from src to dst
  - One copy, but has kernel overhead
  - Publicly available modules
    - Purely using kernel-assisted copy : LiMIC2
    - Using both kernel-assisted and I/OAT-assisted copy: KNEM

Design Goals



- Realize true one-sided synchronization and data transfer
- Design using MVAPICH2 code base

One-Sided Synchronization Design



Process 0

P00
P01
P02
P03

C00 C01 C02 C03

Process 1

P10
P11
P12
P13

C10 C11 C12 C13

C20 C21 C22 C23

Process 2

P20
P21
P22
P23

C30 C31 C32 C33

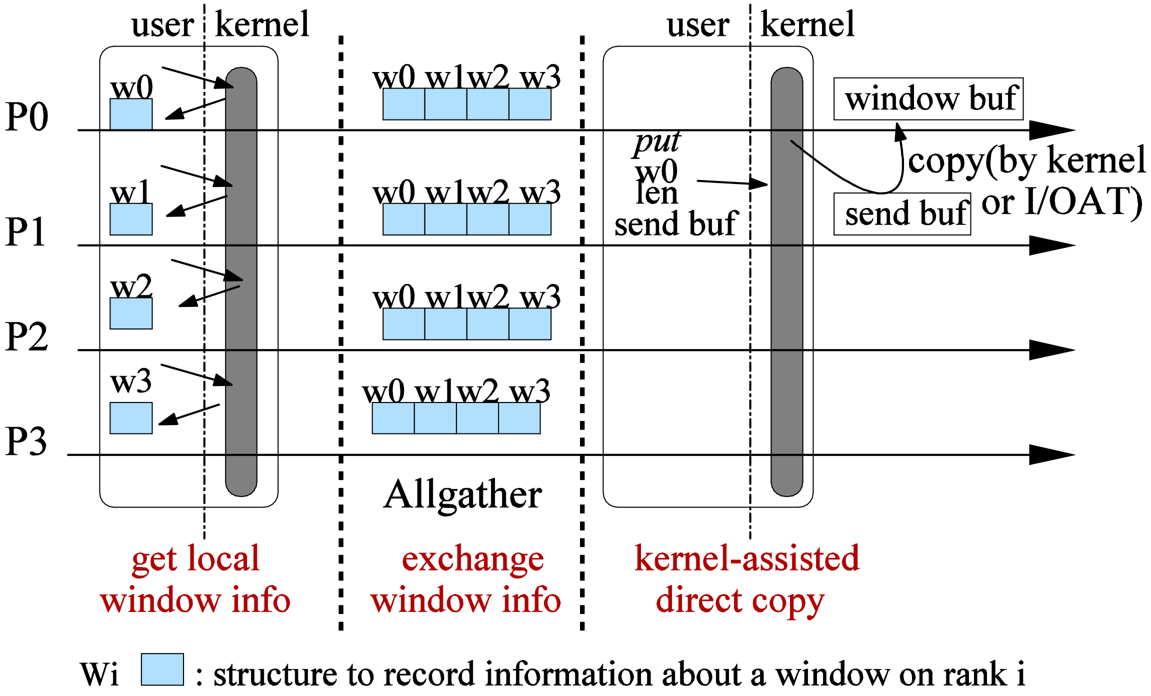
Process 3

P30
P31
P32
P33

Pnm: shared memory for process m to expose post to process n
Cnm: shared memory for process m to write completion to process n

- Pair-wise shared memory for “post” and “complete”
  - Bit vectors
- Shared memory read and write for communication
- No send/recv operations needed

One-Sided Data Transfer Design



- Step 1: get information about the own window
- Step 2: exchange window information among intra-node processes
- Step 3: direct copy as needed – use kernel or I/OAT

Design Issues and Solutions

- Lock buffer pages during the copy
  - Use *get_user_pages*
  - Both src and dst buffers are locked for I/OAT
  - Only target window is locked for basic kernel module
- Locking cost is high
  - Enhancement: cache the locked window pages
- I/OAT completion notification
  - I/OAT returns cookie for user to poll completion
  - Frequent polling is not good
  - Only poll before *origin* writes "complete" to *target*

MVAPICH2 and MVAPICH2-LiMIC2

- MVAPICH2
  - High-performance, scalable, and fault-tolerant MPI library for InfiniBand/ 10GigE/iWARP and other RDMA enabled interconnects
  - Developed by Network-Based Computing Laboratory, OSU
  - Being used by more than 1,150 organizations world wide, including many of the top 500 supercomputers (Nov' 09 ranking)
    - 5th ranked NUDT Tianhe –71,680-core system
    - 9th ranked Ranger system at Texas Advanced Computing Center (TACC)
  - Current release versions use two-sided based approach for intra-node RMA communication
  - Proposed design will be incorporated in MVAPICH2

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

- MVAPICH2-LiMIC2
  - LiMIC2 is used for two-sided large message intra-node communication
  - Developed by Hyun-Wook Jin at Konkuk University, Korea

http://sslab.konkuk.ac.kr/

Outline

- Introduction
- Problem Statement
- Proposed Design
- Experimental Results & Analysis
- Conclusions & Future Work

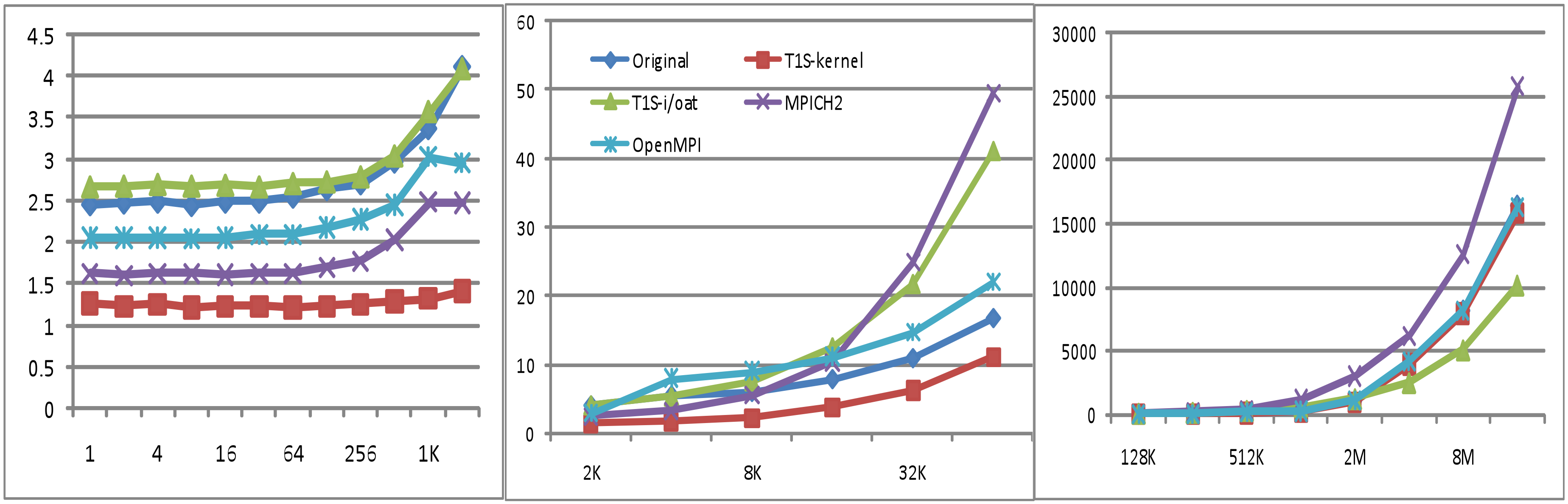
Experimental Setup

- Multi-core Test bed
  - Type A
    - Intel Clovertown, support I/OAT
    - Dual-socket quad-core Xeon E5345 processors (2.33 GHz)
    - Each pair of cores share L2 cache
    - Inter-socket, intra-socket, shared cache intra-node communication
  - Type B
    - Intel Nehalem
    - Dual-socket quad-core Xeon E5530 processors (2.40 GHz)
    - Exclusive L2 cache
    - Inter-socket, intra-socket intra-node communication
  - Type C
    - AMD Barcelona
    - Quad-socket quad-core Opteron 8530 processors
    - Exclusive L2 cache
    - Inter-socket, intra-socket intra-node communication

Experiment Overview

- Basic latency & bandwidth performance
- More micro benchmarks
  - Reduced process skew effect
  - Increased communication/computation overlap
  - Improved scalability
  - Decreased cache misses
- Application level performance
- Legend
  - Original: current design in MVAPICH2
  - T1S-kernel: proposed design using basic kernel module
  - T1S-i/oat: proposed design using I/OAT-assisted module
  - MPICH2: two-sided based ; shared-memory based send/recv
  - OpenMPI: two-sided based; KNEM large message send/recv

Intra-socket *Get* Latency on Intel Clovertown



small message latency (usec)

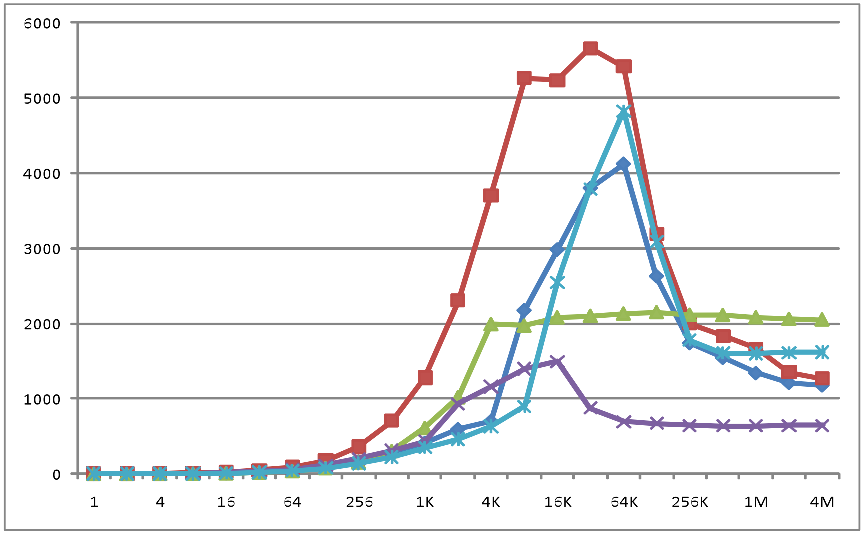
medium message latency (usec)

large message latency (usec)

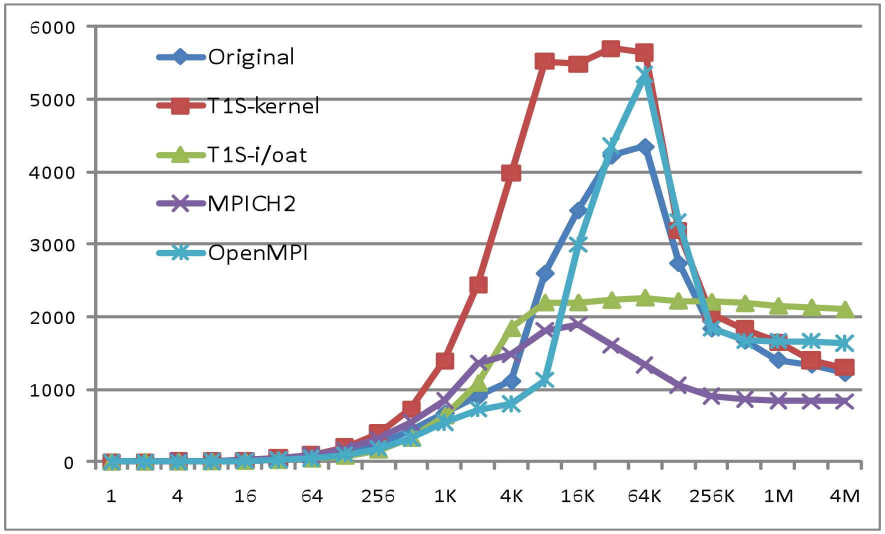
- T1S-kernel improves small and medium message latency up to 39%
- T1S-i/oat design improves latency of very large messages up to 38%
- Similar results for *put* latency

Get Bandwidth on Intel Clovertown

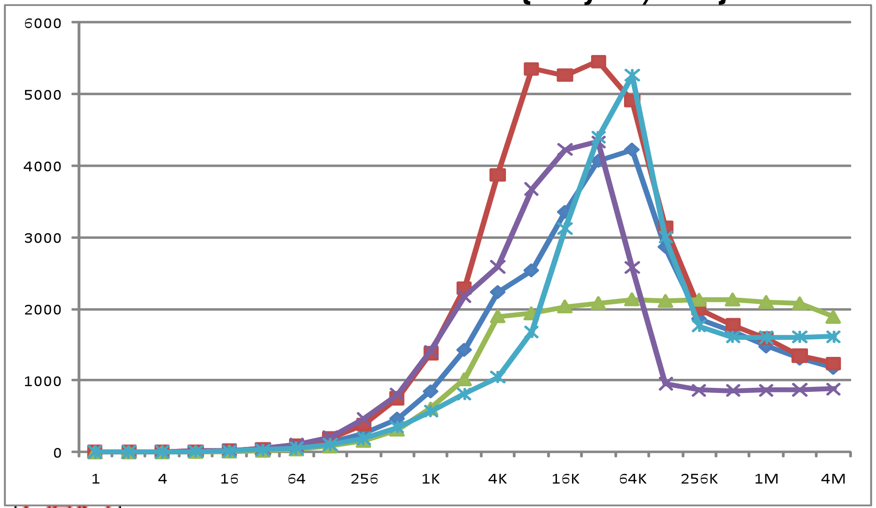
Inter-socket bandwidth (Mbytes/sec)



Intra-socket bandwidth (Mbytes/sec)

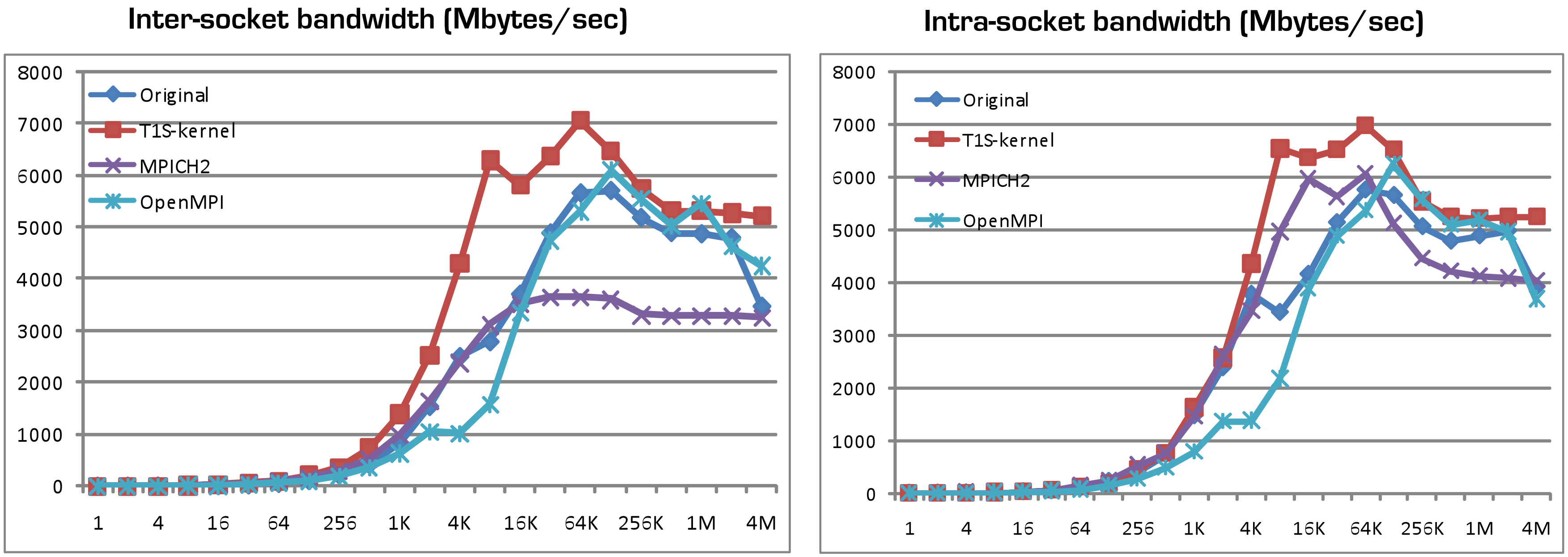


Shared-cache bandwidth (Mbytes/sec)



- T1S-kernel design improves medium message BW
- T1S-i/oat starts gaining benefit beyond 256 KB
- *Put* has similar performance

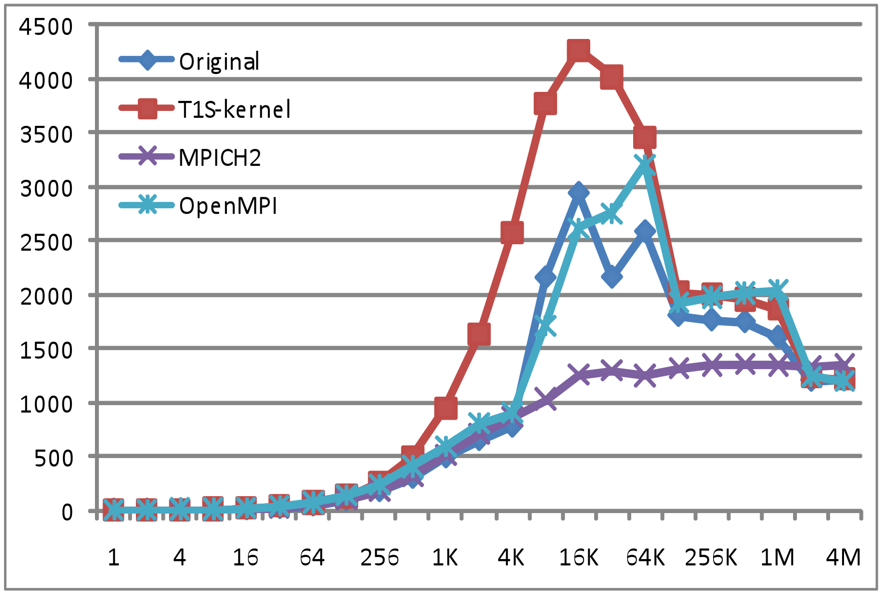
Get Bandwidth on Intel Nehalem



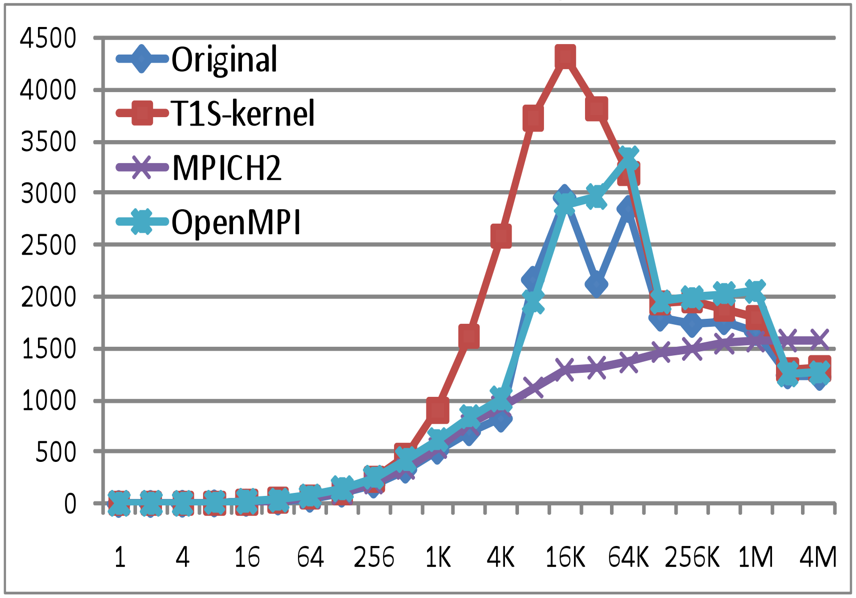
- T1S-kernel design improves medium message BW
- *Put* has similar performance

Get Bandwidth on AMD Barcelona

Inter-socket bandwidth (Mbytes/sec)

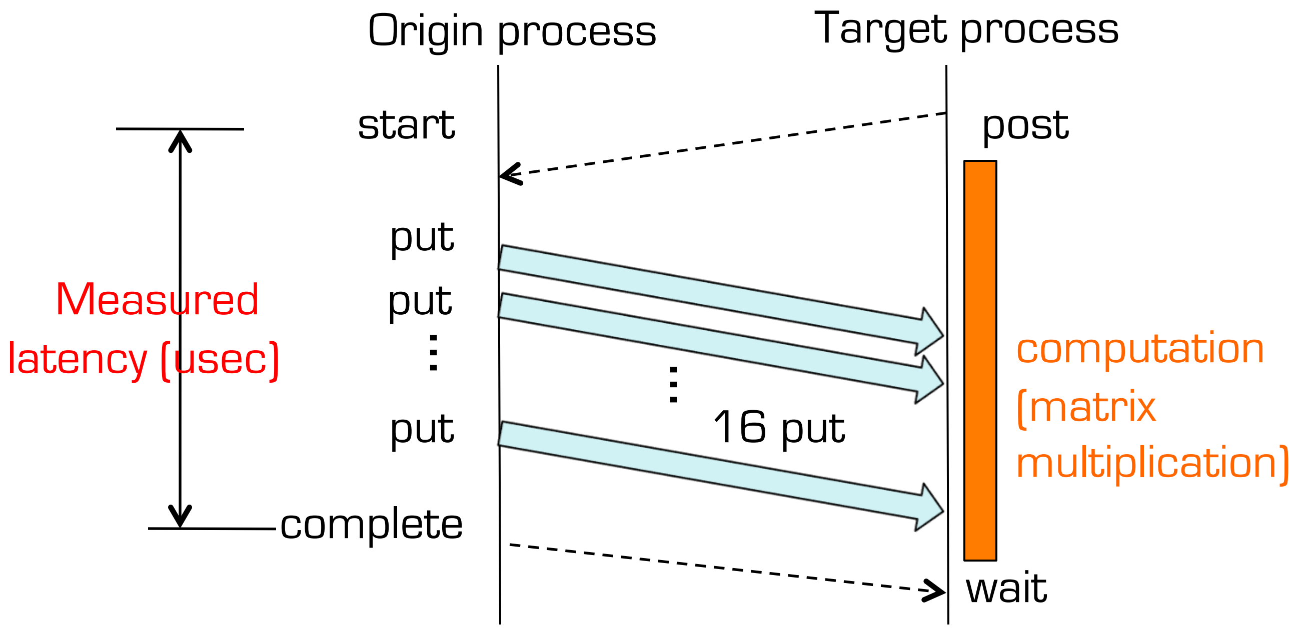


Intra-socket bandwidth (Mbytes/sec)



- T1S-kernel design improves medium message bandwidth
- *Put* has similar performance

Reduced Process Skew



Origin process

Target process

start

post

put

put

⋮

put

⋮

16 put

Measured latency (usec)

computation (matrix multiplication)

complete

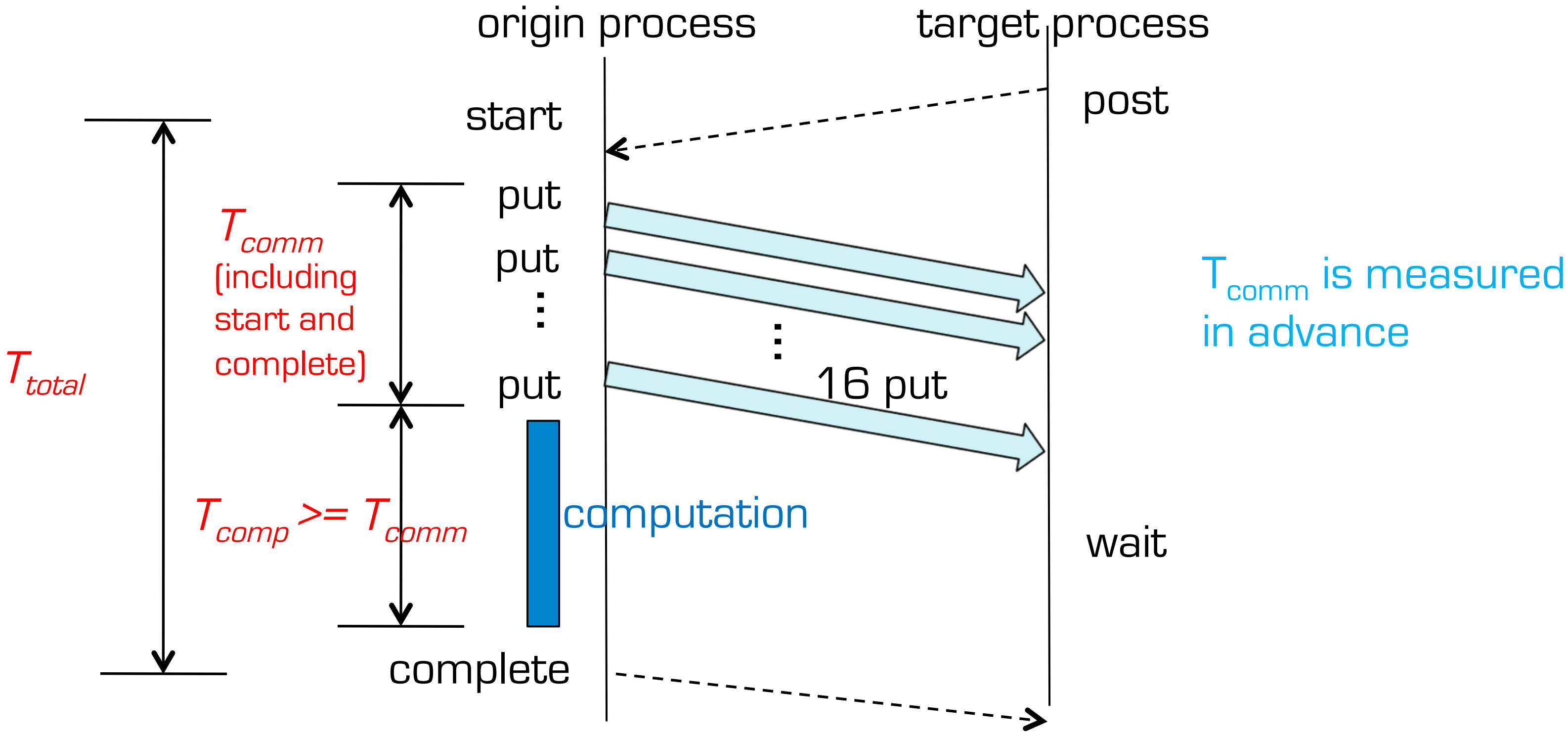
wait

Latency (usec) of 16 put with increasing process skew (message size = 256KB)

Matrix size	no comp	32x32	64x64	128x128	256x256
Original	3404	3780	6126	27023	194467
T1S-kernel	3365	3333	3398	3390	3572
T1S-i/oat	2291	2298	2310	2331	2389

- New designs remove dependency, more robust to process skew

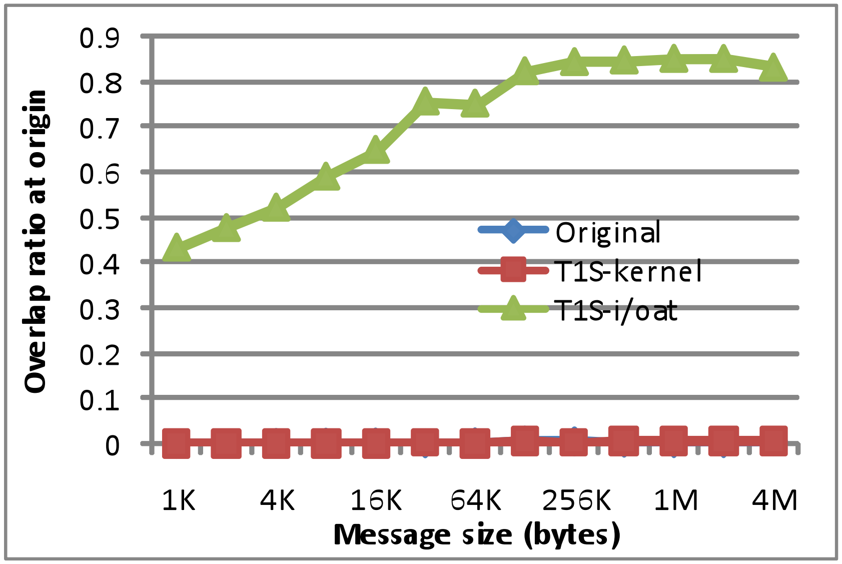
Increased Communication and Computation Overlap



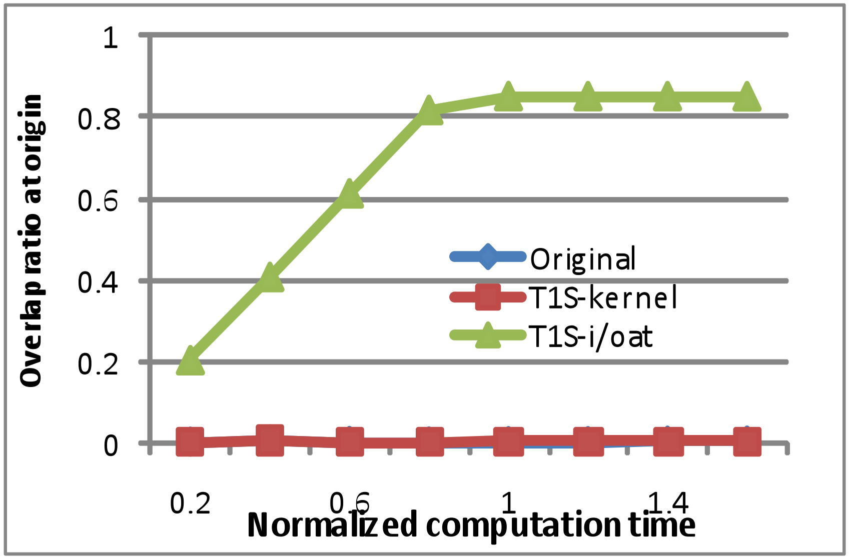
- Experiment design for measuring overlap at origin
- Overlap = $(T_{comm} + T_{comp} - T_{total}) / T_{comm}$
  - If $T_{comp} = T_{total}$, overlap = 1; fully overlapped
  - If $T_{comp} + T_{comm} = T_{total}$, overlap = 0; no overlap

Origin Side Overlap

Overlap with varying message size (Tcomp=1.2 Tcomm)

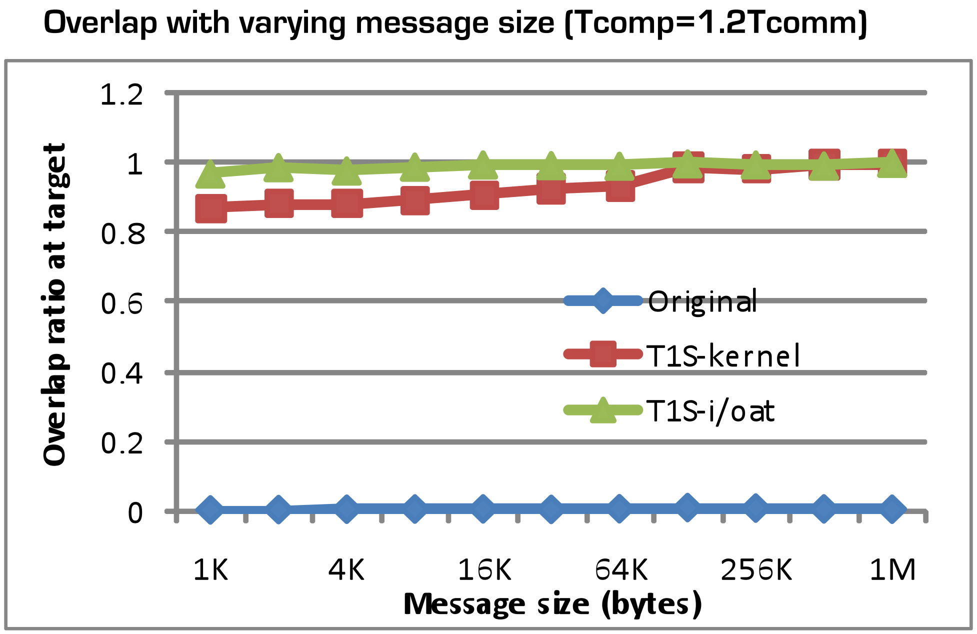


Overlap with varying computation time (msg size=1MB)



- I/OAT based design provides close to 90% overlap
  - Offload data movement to DMA engine
  - Release the CPU for computation

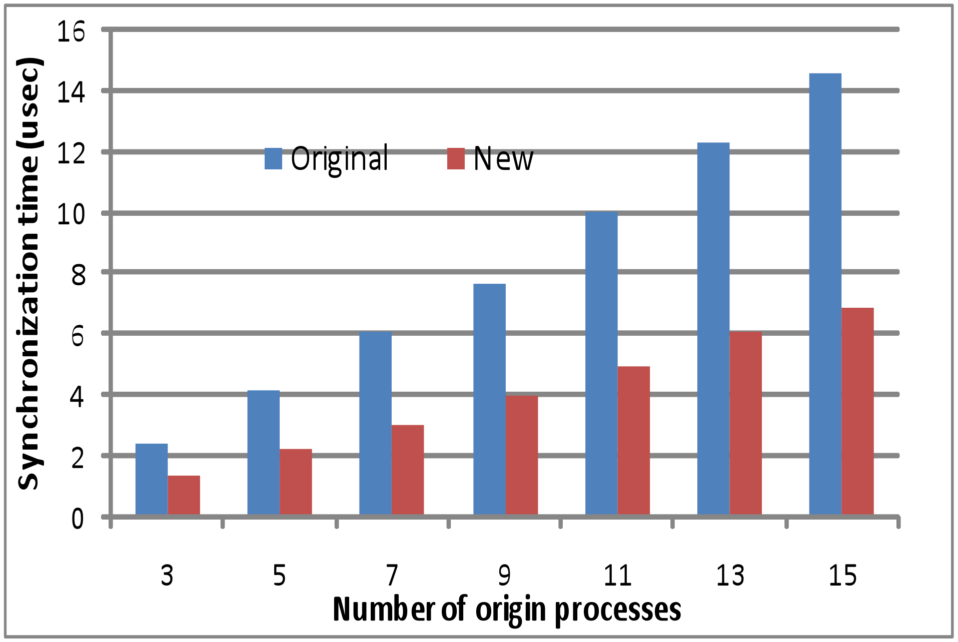
Target Side Overlap

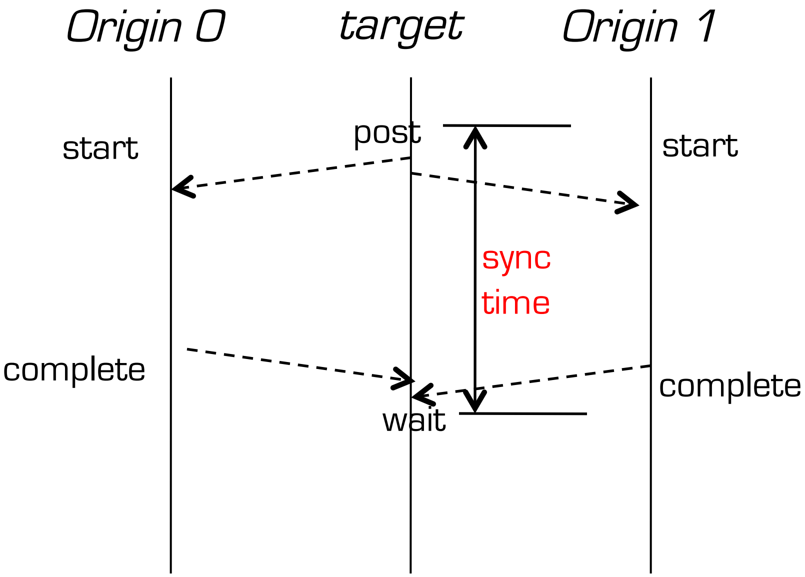


- Similar benchmark as previous benchmark
  - Insert computation at the target
- New designs provide up to 100% overlap
  - *Origin* does the communication (message copy)
  - *Target* does the computation simultaneously

Reduced Synchronization Cost

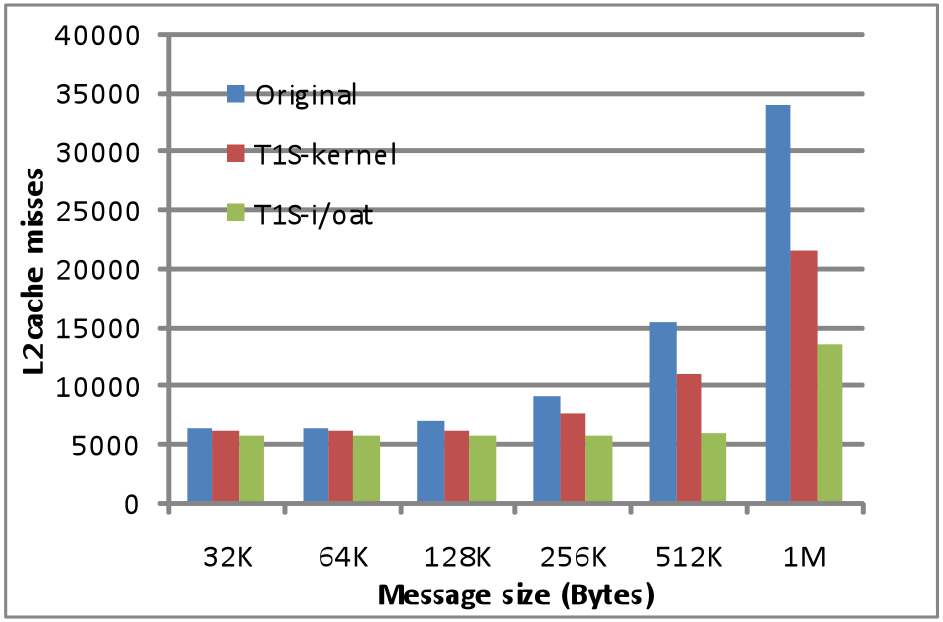
Synchronization time with multiple *origin* processes





- New designs decouple *origin* and *target*
  - *Target* is more capable of handling more *origin* processes

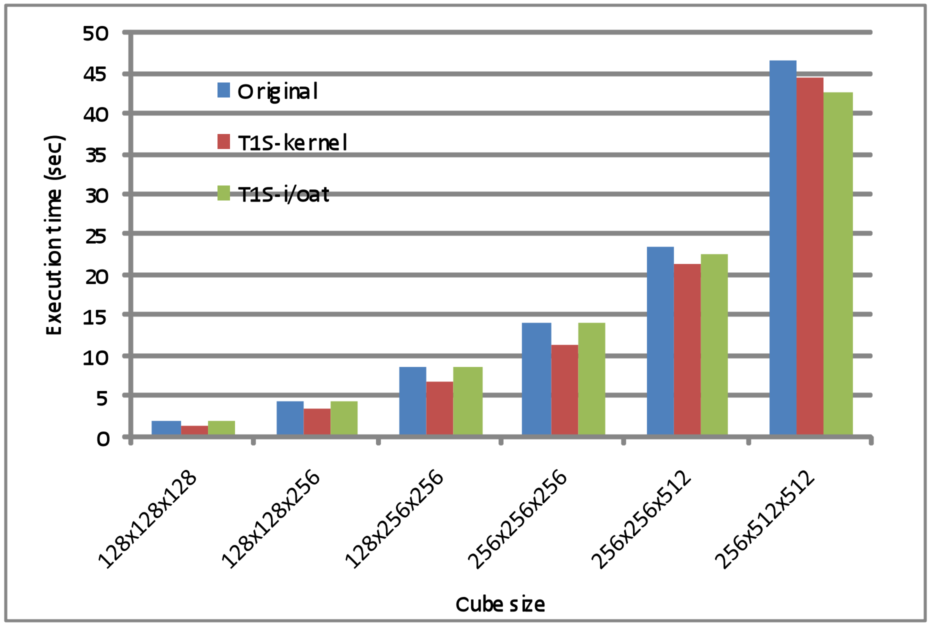
Decreased Cache Misses



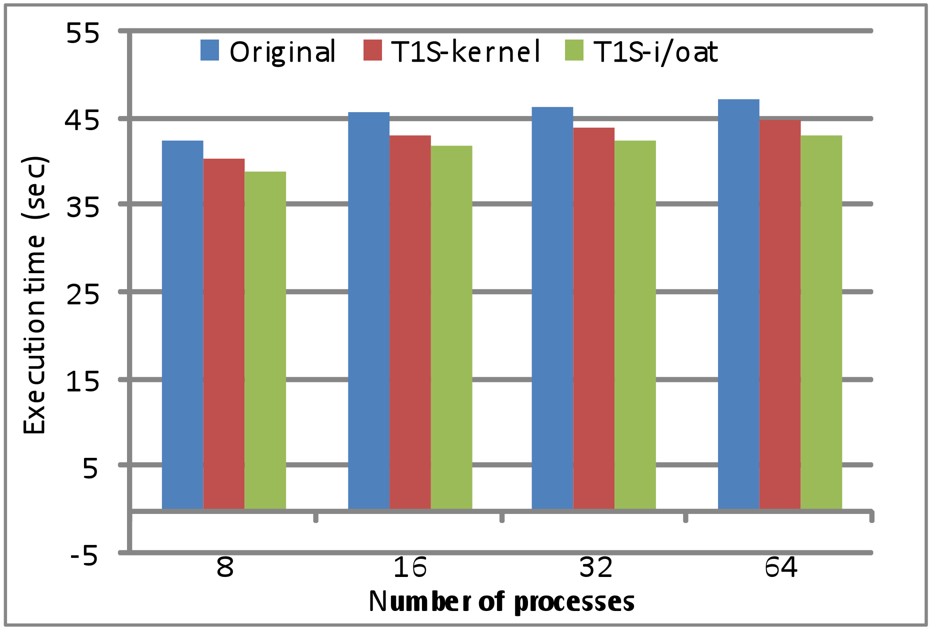
- Cache misses during the aggregated bandwidth test
  - Seven *origin* processes and one *target*
- T1S-i/oat has the least cache misses
- T1S-kernel also reduces cache misses a lot

Application Performance

Performance with varying data sets (32 processes)



Weak scaling performance (128x128x128 elements per process)



- AWM-Olsen: stencil-based earthquake simulation application
  - Nearest-neighbor communication; performs on 3-dimensional data set
  - Modified it to use MPI-2 one-sided semantics
  - S. Potluri, P. Lai, K. Tomko, S. Sur, Y. Cui, M. Tatineni, K. Schulz,W. Barth, A. Majumdar and D. K. Panda, "Quantifying Performance Benefits of Overlap using MPI-2 in a Seismic Modeling Application", International Conference on Supercomputing (ICS) 2010, Tsukuba, Japan
- New designs show 10% improvement for larger problem sizes

Outline

- Introduction
- Problem Statement
- Proposed Design
- Experimental Results & Analysis
- Conclusions & Future Work

Conclusions & Future Work

- We designed and implemented truly one-sided intra-node communication
  - one-sided synchronization
  - one-sided data transfer
    - Basic kernel-assisted approach
    - I/OAT-assisted approach
- Evaluated the performance on three multi-core systems (Intel Clovertown, Intel Nehalem, AMD Barcelona)
  - New designs offer better performance in terms of latency, bandwidth, communication and computation overlap, cache misses and application level benefits etc.
- Future work
  - Evaluate on other platforms and do large-scale evaluations
  - Include in public MVAPICH2 release

Thank You!

{laipi, surs, panda}@cse.ohio-state.edu

jinh@konkuk.ac.kr

Network Based Computing Laboratory

MVAPICH

System Software Laboratory
2006
Konkuk University

Network-Based Computing Laboratory

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

MVAPICH Web Page

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

System Software Lab

http://sslab.konkuk.ac.kr