Ohio Supercomputer Center

An OH·TECH Consortium Member

MPI+PGAS Hybrid Programming

Karen Tomko

ktomko@osc.edu

In collaboration with DK Panda and the Networked-Based Computing Research group at Ohio State University

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

Ohio Supercomputer Center

An OH·TECH Consortium Member

Background: Systems and Programming Models

Drivers of Modern HPC Cluster Architectures

Multi-core Processors

High Performance Interconnects - InfiniBand
<1usec latency, >100Gbps Bandwidth

Accelerators / Coprocessors
high compute density, high performance/watt
>1 TFlop DP on a chip

- Multi-core processors are ubiquitous
- InfiniBand very popular in HPC clusters
- Accelerators/Coprocessors becoming common in high-end systems
- Pushing the envelope for Exascale computing

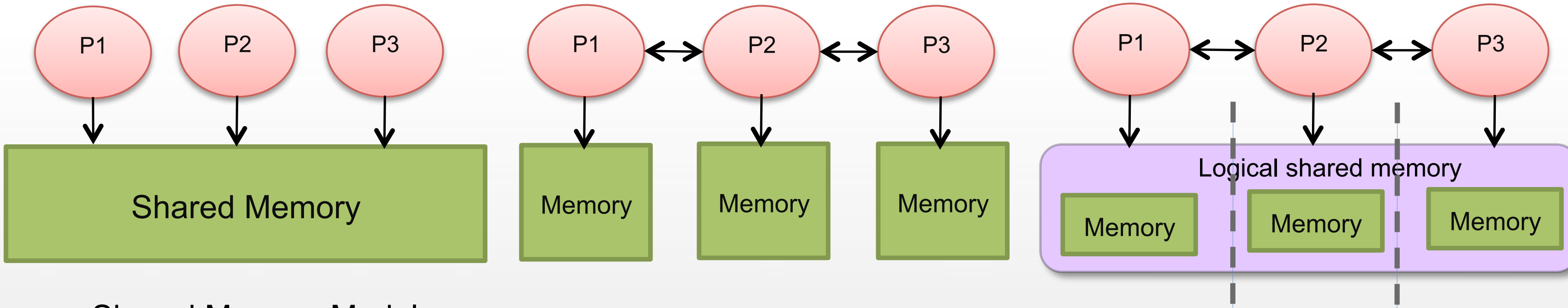
Tianhe – 2 (1)

Titan (2)

Piz Daint (CSCS) (6)

Stampede (7)

Parallel Programming Models Overview



Shared Memory Model

Distributed Memory Model

MPI (Message Passing Interface)

Partitioned Global Address Space (PGAS)

Global Arrays, UPC, OpenSHMEM, CAF, …

- Programming models provide abstract machine models
- Models can be mapped on different types of systems
  - e.g. Distributed Shared Memory (DSM), MPI within a node, etc.
- Additionally, OpenMP can be used to parallelize computation within the node
- Each model has strengths and drawbacks - suite different problems or applications

Partitioned Global Address Space (PGAS) Models

- Key features
  - Simple shared memory abstractions
  - Light weight one-sided communication
  - Easier to express irregular communication
- Different approaches to PGAS
  - Languages
    - Unified Parallel C (UPC)
    - Co-Array Fortran (CAF)
    - others
  - Libraries
    - OpenSHMEM
    - Global Arrays

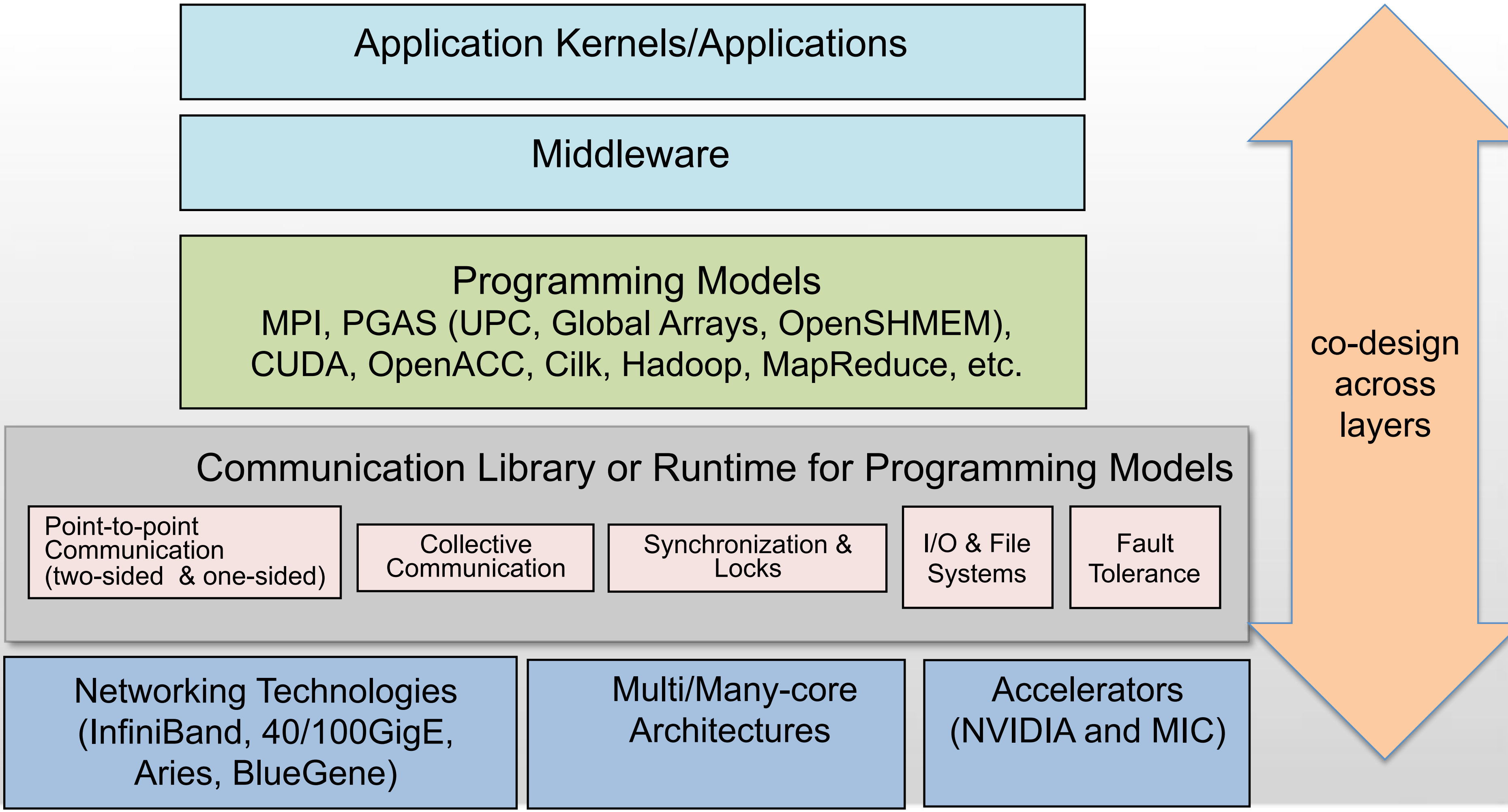
MPI+PGAS for Exascale Architectures and Applications

- Hierarchical architectures with multiple address spaces
- (MPI + PGAS) Model
  - MPI across address spaces
  - PGAS within an address space
- MPI is good at moving data between address spaces
- Within an address space, MPI can interoperate with shared memory programming models

- Applications can have kernels with different communication patterns
- Can benefit from different models

- Re-writing complete applications can be a huge effort
- Port critical kernels to the desired model instead

Supporting Programming Models for Multi-Petaflop and Exaflop Systems: Challenges



Can High-Performance Interconnects, Protocols and Accelerators Benefit from PGAS and Hybrid MPI+PGAS Models?

- MPI designs have been able to take advantage of high-performance interconnects, protocols and accelerators
- Can PGAS and Hybrid MPI+PGAS models take advantage of these technologies?
- What are the challenges?
- Where do the bottlenecks lie?
- Can these bottlenecks be alleviated with new designs (similar to the designs adopted for MPI)?

Ohio Supercomputer Center
An OH·TECH Consortium Member

PGAS Programming Models – OpenSHMEM Library

SHMEM

- SHMEM: Symmetric Hierarchical MEMory library
- One-sided communications library – had been around for a while
- Similar to MPI, processes are called PEs, data movement is explicit through library calls
- Provides globally addressable memory using symmetric memory objects (more in later slides)
- Library routines for
  - Symmetric object creation and management
  - One-sided data movement
  - Atomics
  - Collectives
  - Synchronization

OpenSHMEM

- SHMEM implementations – Cray SHMEM, SGI SHMEM, Quadrics SHMEM, HP SHMEM, GSHMEM
- Subtle differences in API, across versions – example:

	SGI SHMEM	Quadrics SHMEM	Cray SHMEM
Initialization	*start_pes(0)*	*shmem_init*	*start_pes*
Process ID	*_my_pe*	*my_pe*	*shmem_my_pe*

- Made applications codes non-portable
- OpenSHMEM is an effort to address this:

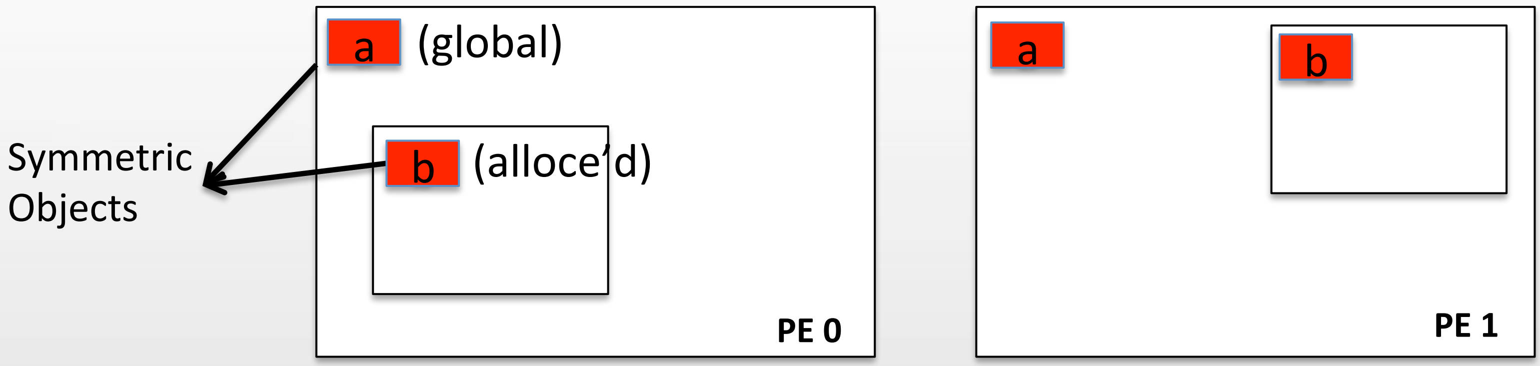
"A new, open specification to consolidate the various extant SHMEM versions into a widely accepted standard." – OpenSHMEM Specification v1.0

by University of Houston and Oak Ridge National Lab

SGI SHMEM is the baseline

The OpenSHMEM Memory Model

- Symmetric data objects
  - Global Variables
  - Allocated using collective *shmalloc, shmemalign, shrealloc* routine

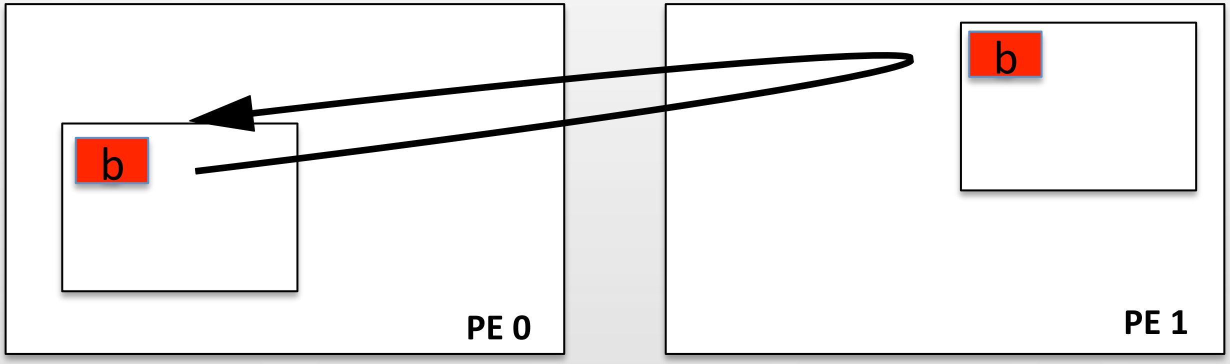


Virtual Address Space

- Globally addressable – objects have same
  - Type
  - Size
  - Same virtual address or offset at all PEs
  - Address of a remote object can be calculated based on info of local object

Data Movement: Basic

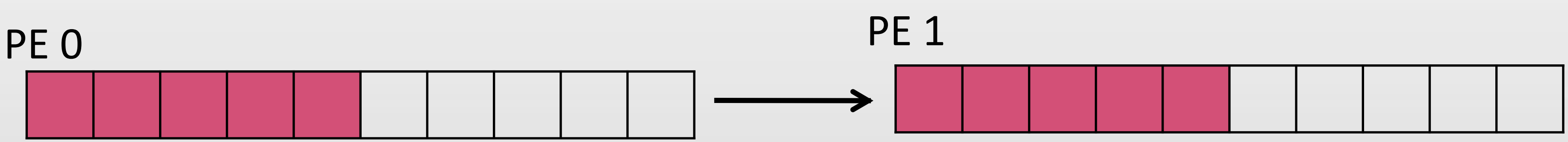
- Put and Get – single element
  - void shmem_*TYPE*_p (*TYPE* *ptr, int PE)
  - void shmem_*TYPE*_g (*TYPE* *ptr, int PE)
  - *TYPE* can be short, int, long, float, double, longlong, longdouble




```
int *b; int c;
b =  (int *) shmalloc (sizeof(int));

if ((_my_pe() == 0) {
    c = shmem_int_g (b, 1);
}
```

Data Movement: Contiguous

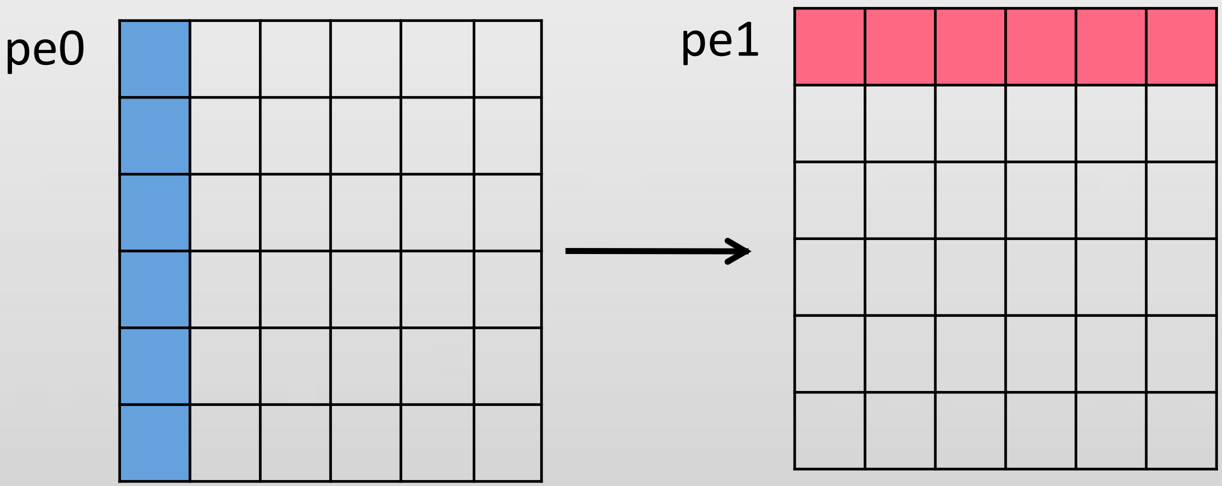
- Block Put and Get – Contiguous
  - void shmem_*TYPE*_put (TYPE* target, const TYPE*source, size_t nelems, int pe)
    - *TYPE* can be char, short, int, long, float, double, longlong, longdouble
  - shmem_put*SIZE* – elements of SIZE: 32/64/128
  - shmem_putmem - bytes
  - Similar get operations




```
int *b;
b =  (int *) shmalloc (10*sizeof(int));

if ((_my_pe() == 0) {
   shmem_int_put (b, b, 5, 1);
}
```

Data Movement: Non-contiguous

- Strided Put and Get
  - shmem_*TYPE*_iput (TYPE* target, const TYPE*source, ptrdiff_t tst, ptrdiff_t sst, size_t nelems, int pe)
    - sst is stride at source, tst is stride at target
    - *TYPE* can be char, short, int, long, float, double, longlong, longdouble
  - Similar get operations



Target stride: 1
Source stride: 6
Num. of elements: 6
shmem_int_iput(t, t, 1, 6, 6, 1)

Data Movement - Completion

- When Put operations return
  - Data has been copied out of the source buffer object
  - Not necessarily written to the target buffer object
  - Additional synchronization to ensure remote completion
- When Get operations return
  - Data has been copied into the local target buffer
  - Ready to be used

Collective Synchronization

- Barrier ensures completion of all previous operations
- Global Barrier
  - void shmem_barrier_all()
  - Does not return until called by all PEs
- Group Barrier
  - Involves only an *"ACTIVE SET"* of PEs
  - Does not return until called by all PEs in the *"ACTIVE SET"*
  - void shmem_barrier (int PE_start, */* first PE in the set */*
    int logPE_stride, */* distance between two PEs*/*
    int PE_size, */*size of the set*/*
    long *pSync /*symmetric work array*/);
  - pSync allows for overlapping collective communication

One-sided Synchronization

- Fence
  - void shmem_fence (void)
  - Enforces ordering on Put operations issued by a PE to each destination PE
  - Does not ensure ordering between Put operations to multiple PEs
- Quiet
  - void shmem_quiet (void)
  - Ensures remote completion of Put operations to all PEs
- Other point-to-point synchronization
  - shmem_wait and shmem_wait_until – poll on a local variable

Collective Operations and Atomics

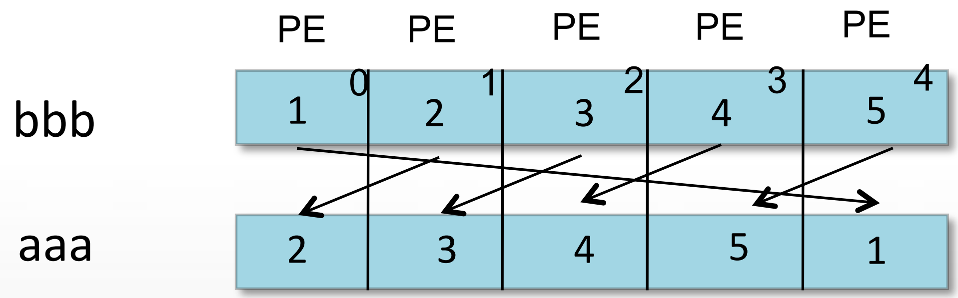
- Broadcast – one-to-all
- Collect – allgather
- Reduction – allreduce (and, or, xor; max, min; sum, product)
- Work on an active set – start, stride, count

- Unconditional - Swap Operation
  - long shmem_swap (long *target, *long* value, int pe)
  - *TYPE* shmem_TYPE_swap (*TYPE* *target, *TYPE* value, int pe)
  - *TYPE* can be int, long, longlong, float, double
- Conditional - Compare and Swap Operation
- Arithmetic – Fetch & Add, Fetch & Increment, Add, Increment

Remote Pointer Operations

- void *shmem_ptr (void *target, int pe)
  - Allows direct load/stores on remote memory
  - Useful when PEs are running on same node
  - Not supported in all implementations
  - Returns NULL if not accessible for loads/stores

A Sample code: Circular Shift




```
#include <shmem.h>

int aaa, bbb;

int main (int argc, char *argv[])
{
    int target_pe;

    start_pes(0);
    target_pe = (_my_pe() + 1)% _num_pes();

    bbb = _my_pe() + 1
     shmem_barrier_all();

    shmem_int_get (&aaa, &bbb, 1, target_pe);
    shmem_barrier_all();
}
```

Ohio Supercomputer Center
An OH·TECH Consortium Member

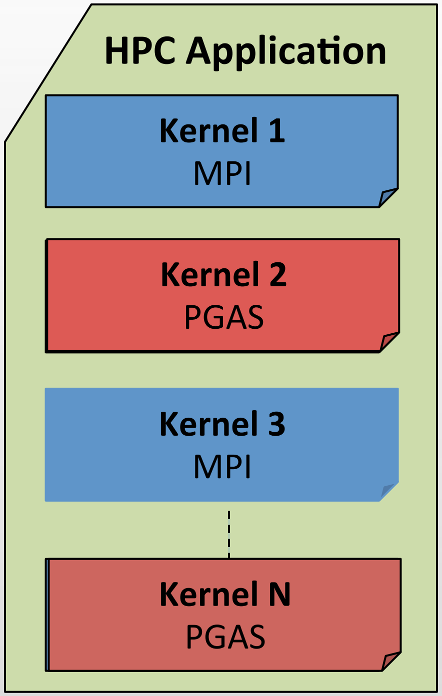
The MVAPICH2-X Hybrid MPI-PGAS Runtime

Maturity of Runtimes and Application Requirements

- MPI has been the most popular model for a long time
  - Available on every major machine
  - Portability, performance and scaling
  - Most parallel HPC code is designed using MPI
  - Simplicity - structured and iterative communication patterns
- PGAS Models
  - Increasing interest in community
  - Simple shared memory abstractions and one-sided communication
  - Easier to express irregular communication
- **Need for hybrid MPI + PGAS**
  - Application can have kernels with different communication characteristics
  - Porting only part of the applications to reduce programming effort

Hybrid (MPI+PGAS) Programming

- Application sub-kernels can be re-written in MPI/PGAS based on communication characteristics
- Benefits:
  - Best of Distributed Computing Model
  - Best of Shared Memory Computing Model
- Exascale Roadmap*:
  - "Hybrid Programming is a practical way to program exascale systems"



** The International Exascale Software Roadmap, Dongarra, J., Beckman, P. et al., Volume 25, Number 1, 2011, International Journal of High Performance Computer Applications, ISSN 1094-3420*

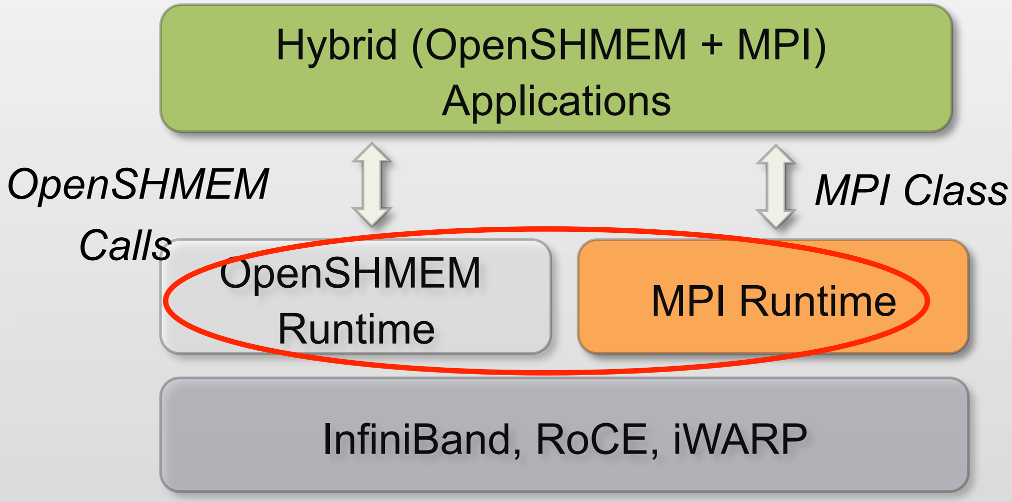
Simple MPI + OpenSHMEM Hybrid Example

```
int main(int c, char *argv[])
{
   int rank, size;

   /* SHMEM init */
   start_pes(0);

   /* fetch-and-add at root */
   shmem_int_fadd(&sum, rank, 0);

   /* MPI barrier */
   MPI_Barrier(MPI_COMM_WORLD);

   /* root broadcasts sum */
   MPI_Bcast(&sum, 1, MPI_INT, 0, MPI_COMM_WORLD);

   fprintf(stderr, "(%d): Sum: %d\n", rank, sum);

   shmem_barrier_all();
   return 0;
}
```

- OpenSHMEM atomic fetch-add
- MPI_Bcast for broadcasting result

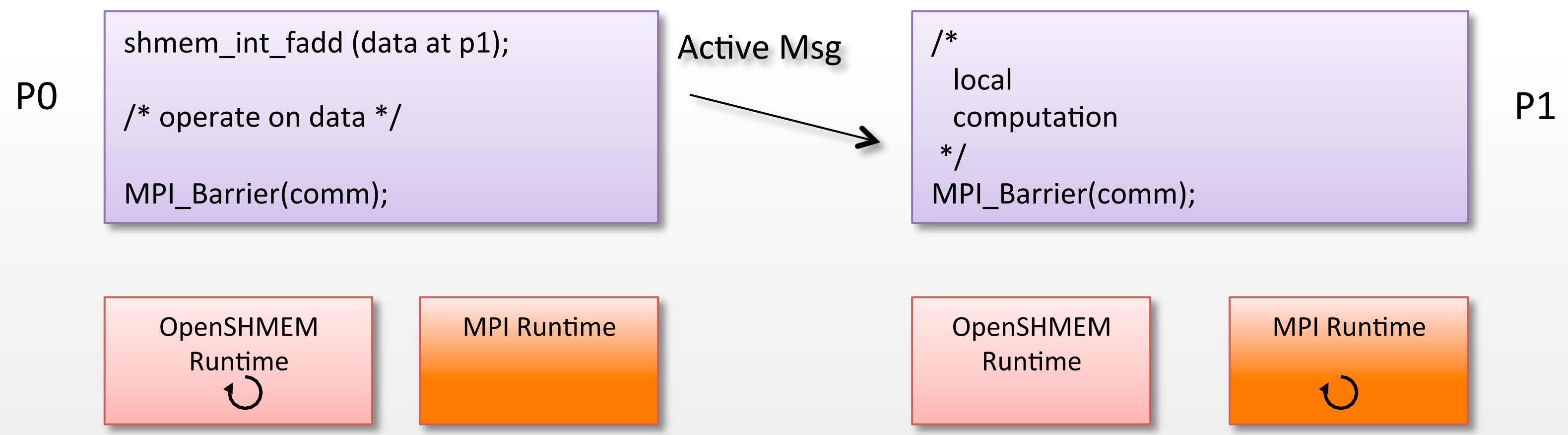
Current approaches for Hybrid Programming

- Layering one programming model over another
  - Poor performance due to semantics mismatch
  - MPI-3 RMA tries to address
- Separate runtime for each programming model



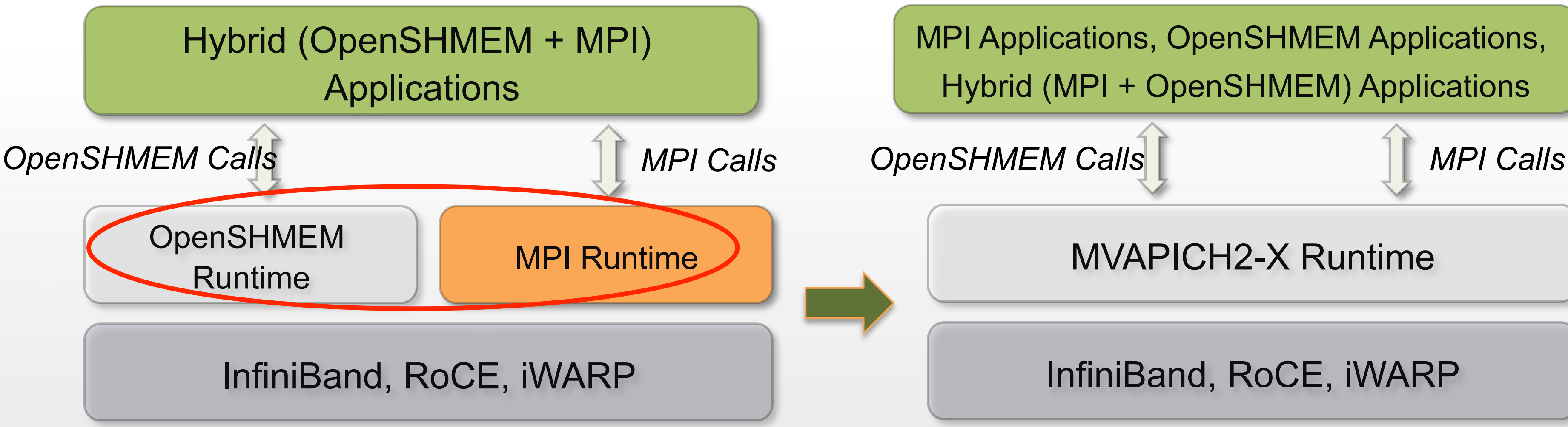
- Need more network and memory resources
- Might lead to deadlock!

The Need for a Unified Runtime



- Deadlock when a message is sitting in one runtime, but application calls the other runtime
- Prescription to avoid this is to barrier in one mode (either OpenSHMEM or MPI) before entering the other
- Or runtimes require dedicated progress threads
- Bad performance!!
- Similar issues for MPI + UPC applications over individual runtimes

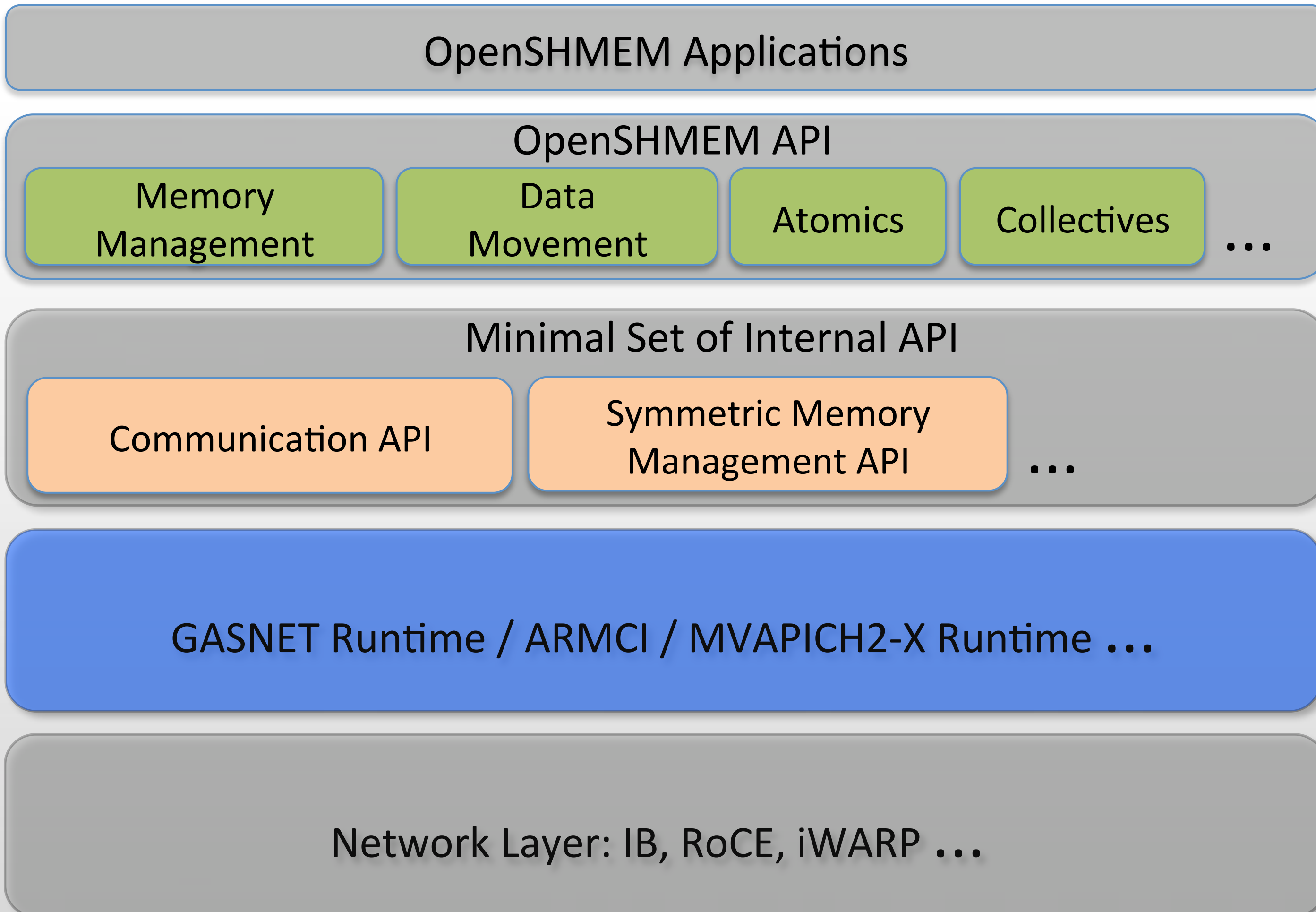
Unified Runtime for Hybrid MPI + OpenSHMEM Applications



- Goal: Provide high performance and scalability for
  - MPI Applications
  - PGAS Applications
  - Hybrid MPI+PGAS Applications
- Resulting runtime
  - Optimal network resource usage
  - No deadlock because of single runtime
  - Better performance

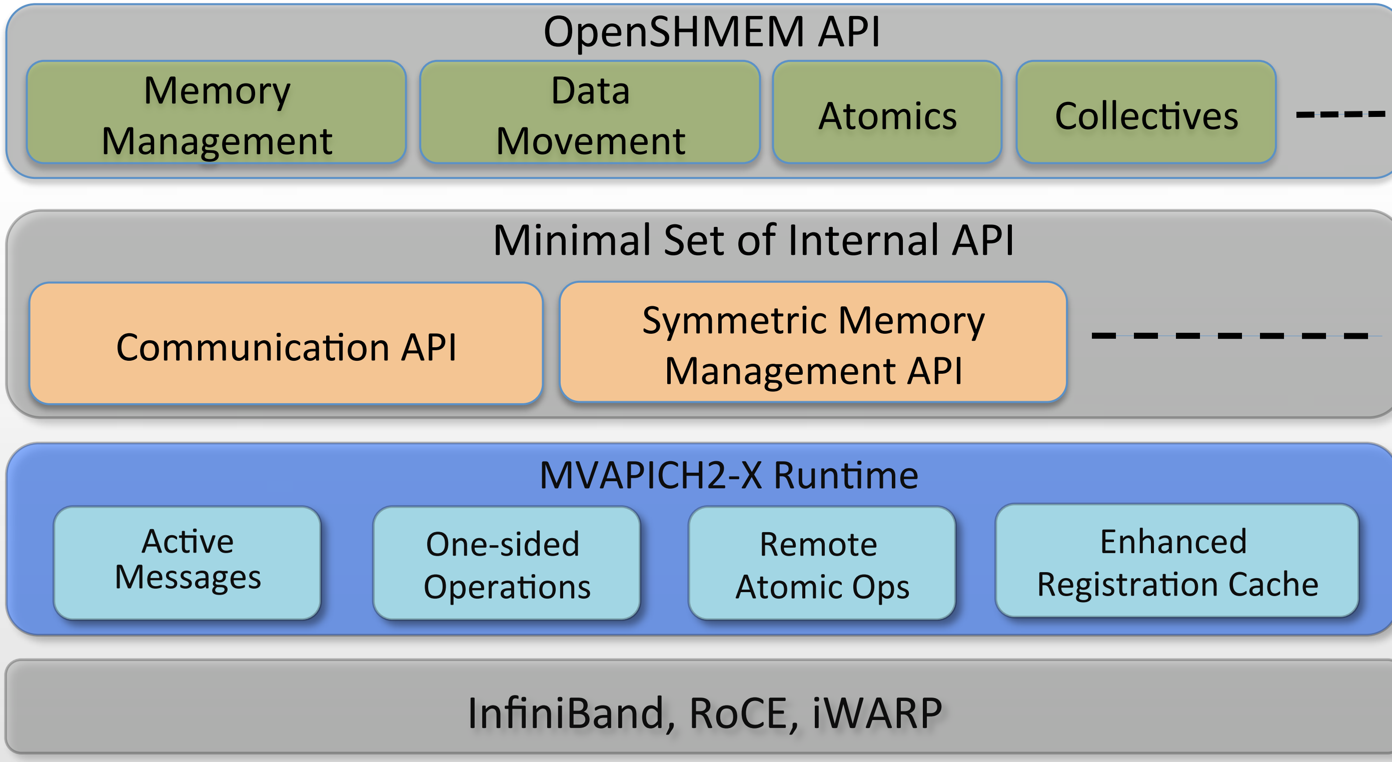
J. Jose, K. Kandalla, M. Luo and D. K. Panda, Supporting Hybrid MPI and OpenSHMEM over InfiniBand: Design and Performance Evaluation, Int'l Conference on Parallel Processing (ICPP '12), September 2012.

OpenSHMEM Reference Implementation Framework



Reference: OpenSHMEM: An Effort to Unify SHMEM API Library Development , Supercomputing 2010

OpenSHMEM Design in MVAPICH2-X



- OpenSHMEM Stack based on OpenSHMEM Reference Implementation
- OpenSHMEM Communication over MVAPICH2-X Runtime
  - Uses active messages, atomic and one-sided operations and remote registration cache

J. Jose, K. Kandalla, M. Luo and D. K. Panda, Supporting Hybrid MPI and OpenSHMEM over InfiniBand: Design and Performance Evaluation, Int'l Conference on Parallel Processing (ICPP '12), September 2012.

Implementations for InfiniBand Clusters

- Reference Implementation
  - University of Houston
  - Based on the GASNet runtime
- MVAPICH2-X
  - The Ohio State University
  - Uses the upper layer of reference implementations
  - Derives the runtime from widely used MVAPICH2 MPI library
  - Available for download: http://mvapich.cse.ohio-state.edu/download/mvapich2x
- OMPI-SHMEM
  - Based on OpenMPI runtime
  - Available in OpenMPI 1.7.5
- ScalableSHMEM
  - Mellanox technologies

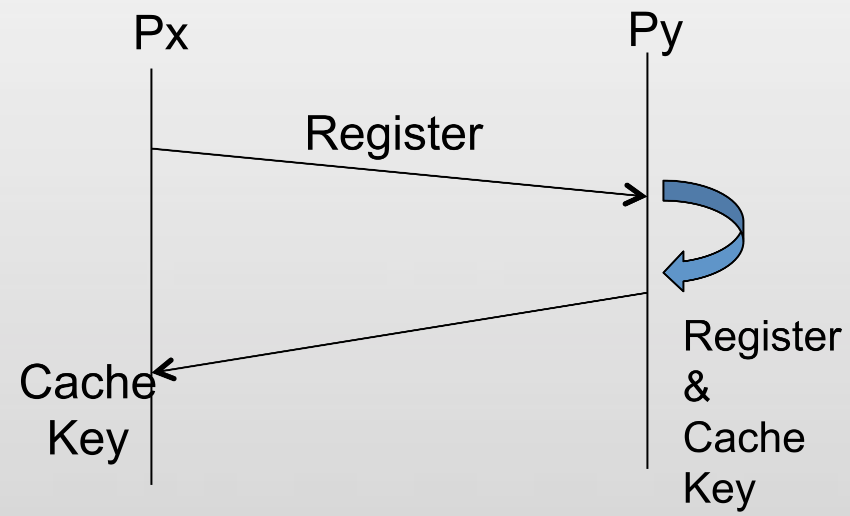
Support for OpenSHMEM Operations in OSU Micro-Benchmarks (OMB)

- Point-to-point Operations
  - osu_oshm_put – Put latency
  - osu_oshm_get – Get latency
  - osu_oshm_put_mr – Put message rate
  - osu_oshm_atomics – Atomics latency
- Collective Operations
  - osu_oshm_collect – Collect latency
  - osu_oshm_broadcast – Broadcast latency
  - osu_oshm_reduce - Reduce latency
  - osu_oshm_barrier - Barrier latency
- OMB is publicly available from:
  - http://mvapich.cse.ohio-state.edu/benchmarks/

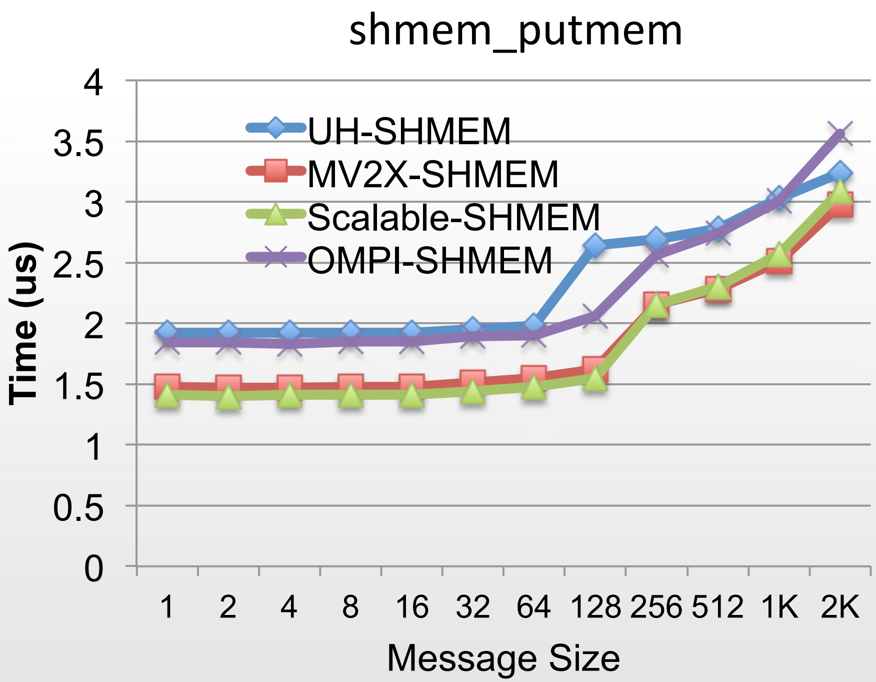
OpenSHMEM Data Movement in MVAPICH2-X

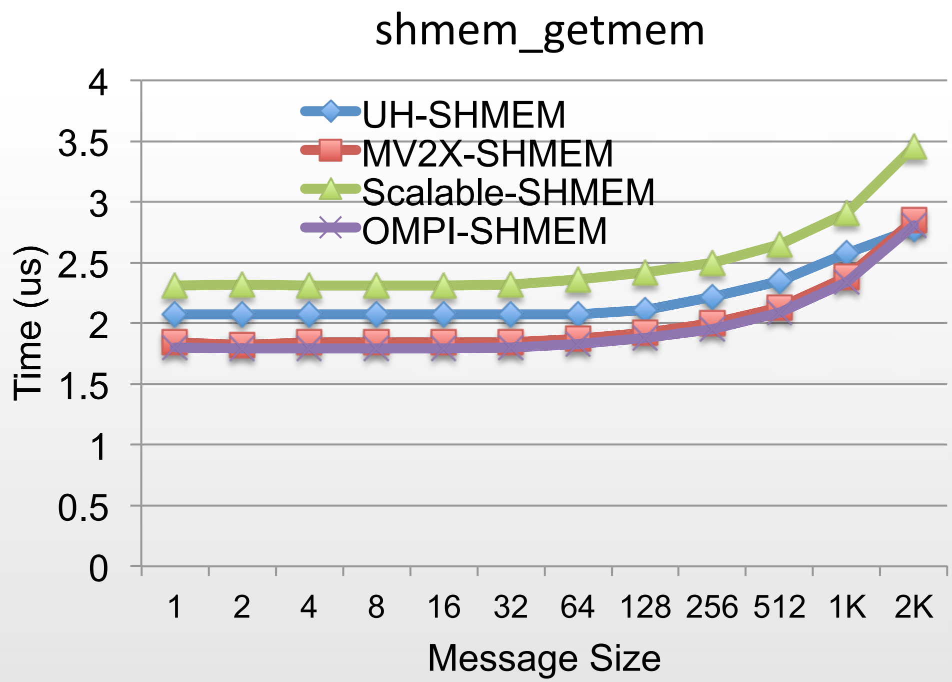
- Data Transfer Routines (put/get)
  - Implemented using RDMA transfers
  - Strided operations require multiple RDMA transfers
  - IB requires remote registration information for RDMA - expensive

- Remote Registration Cache
  - Registration request sent over "Active Message"
  - Remote process registers and responds with the key
  - Key is cached at local and remote sides
  - Hides registration costs



OpenSHMEM Data Movement: Performance



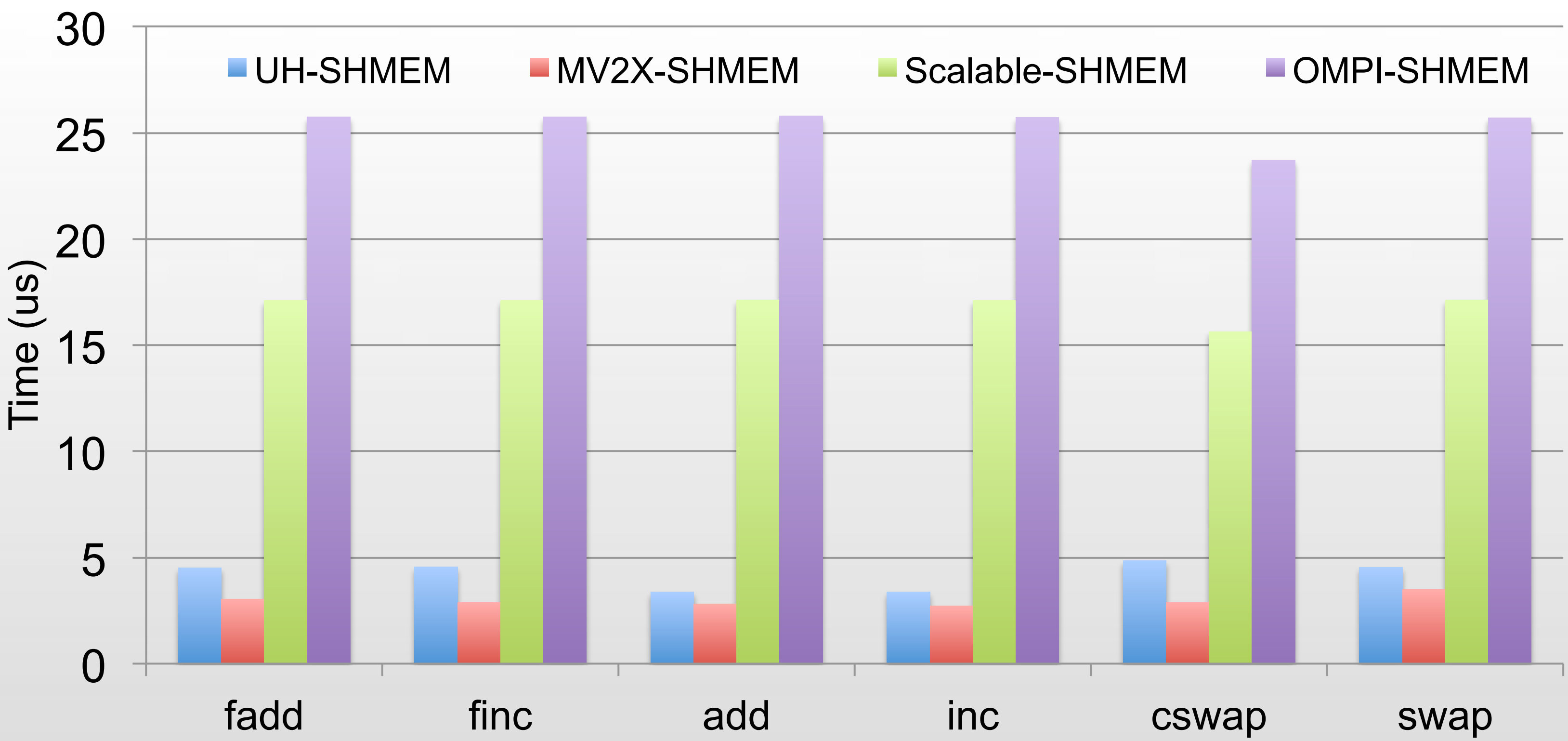


- OSU OpenSHMEM micro-benchmarks - http://mvapich.cse.ohio-state.edu/benchmarks/
- Slightly better performance for putmem and getmem with MVAPICH2-X

Atomic Operations in MVAPICH2-X

- Atomic Operations
  - Take advantage of IB network atomics
  - IB offer atomics for
    - compare-swap
    - fetch-add
    - limited to types of 64-bit length
  - Other operations and types are implemented using "Active Messages"
  - Better performance for 64-bit long types, eg: long

OpenSHMEM Atomic Operations: Performance



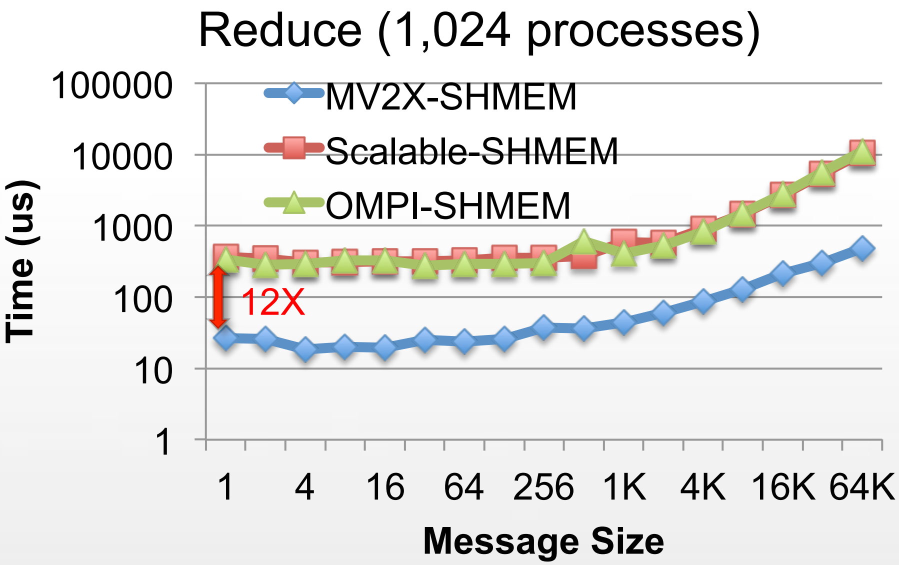
- OSU OpenSHMEM micro-benchmarks (OMB v4.1)
- MV2-X SHMEM performs up to 40% better compared to UH-SHMEM

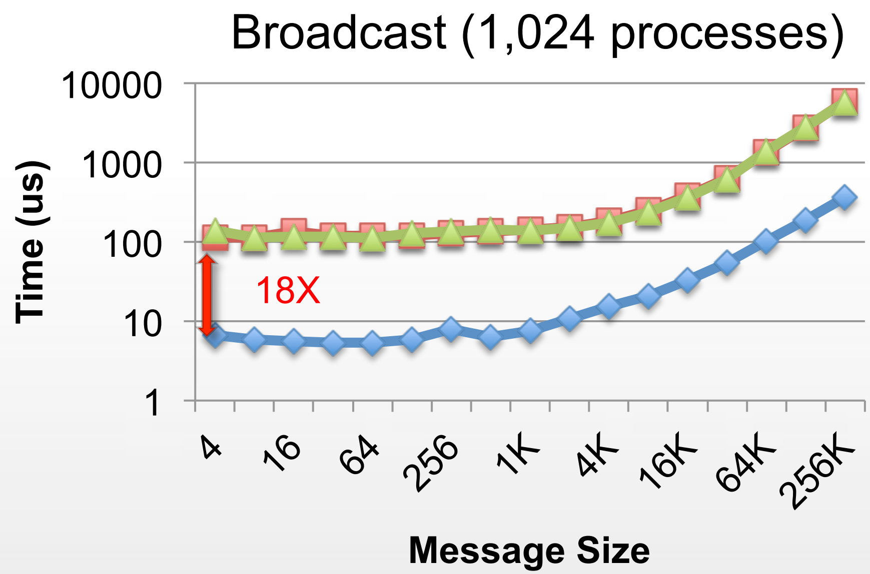
Collective Communication in MVAPICH2-X

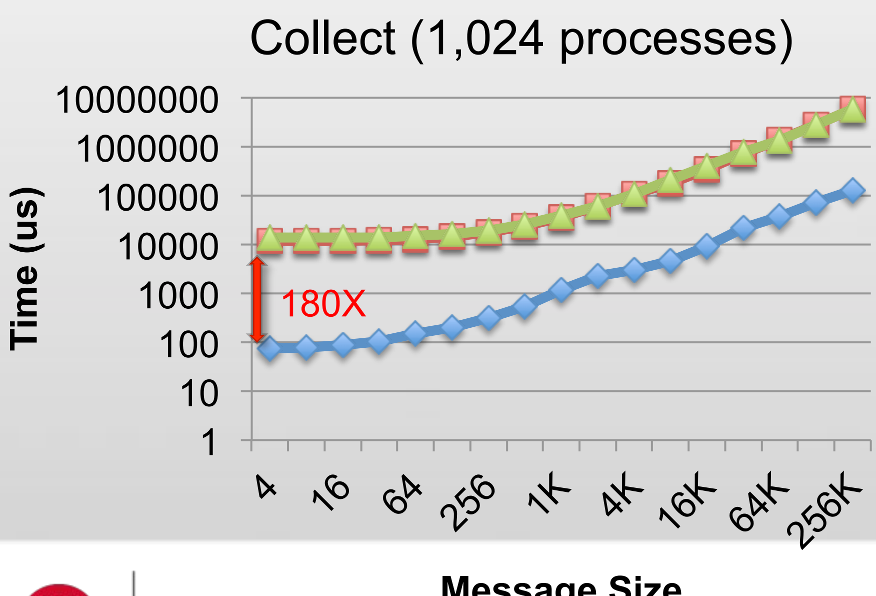
- Significant effort on optimizing MPI collectives to the hilt
- MVAPICH2-X derives from MVAPICH2 MPI runtime
- Implements OpenSHMEM collectives using infrastructure for MPI collectives
  - MPI collectives operate on "communicators" – rigid compared to active set
  - Communicator creation is collective - involves overheads – MPI-3 introduces group-based communicator creation
  - Light-weight and low overhead translation layer using this
  - Communicator cache to hide overheads of creation
- Collect over MPI_Gather, Broadcast over MPI_Bcast, Reduction operations over MPI_Reduce

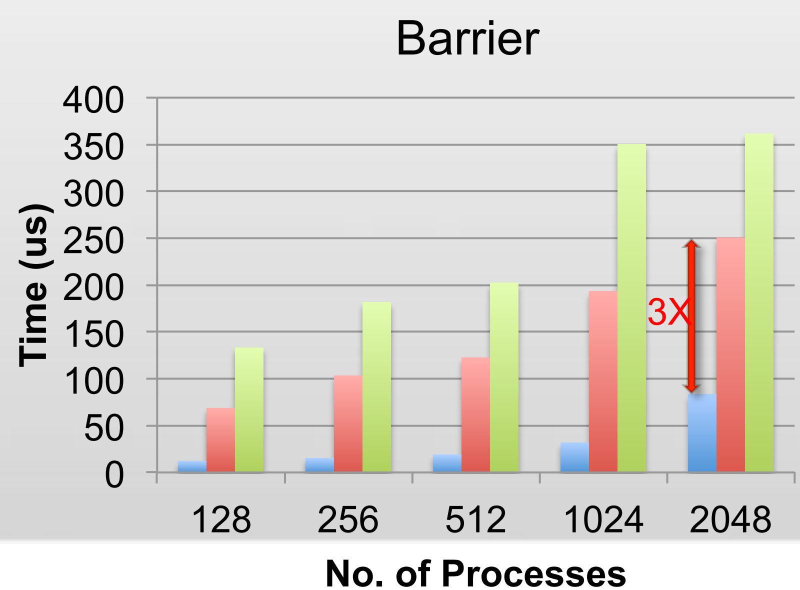
J. Jose, K. Kandalla, S. Potluri, J. Zhang and D. K. Panda, Optimizing Collective Communication in OpenSHMEM, PGAS'13

Collective Communication: Performance

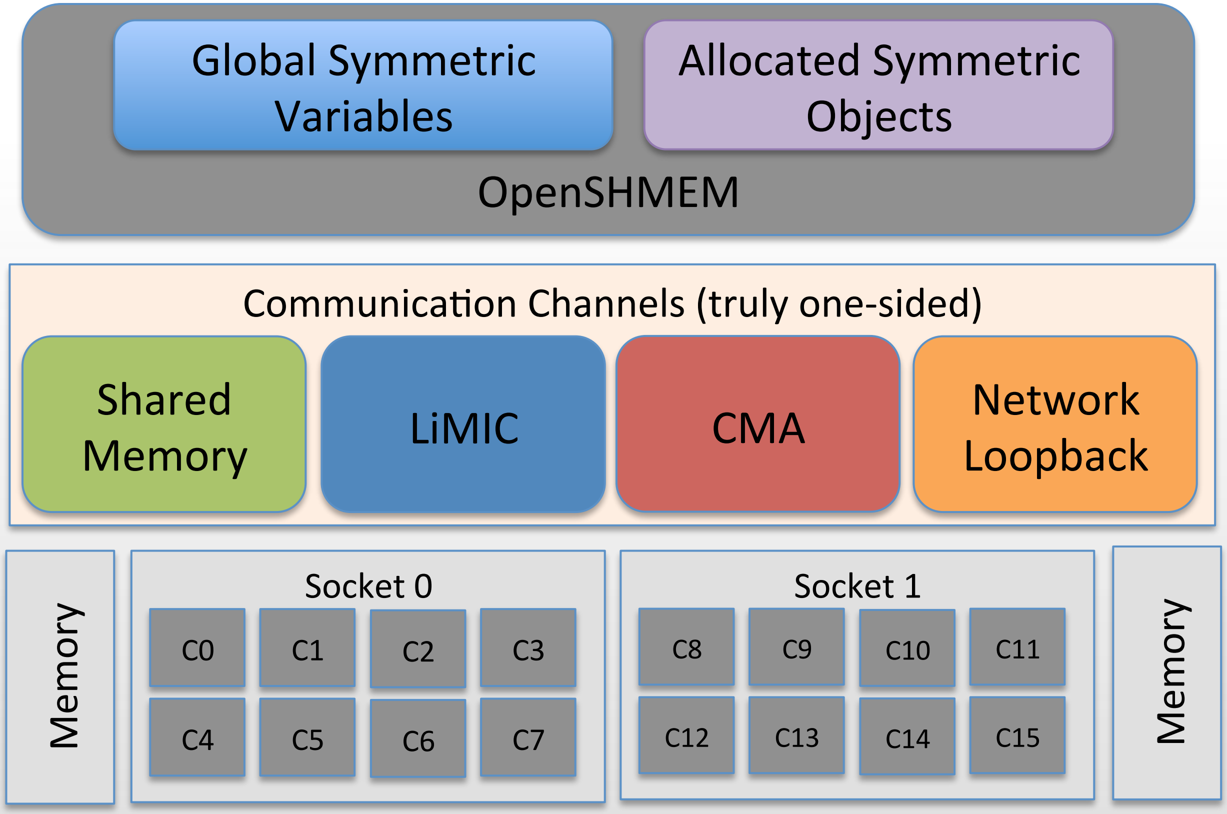








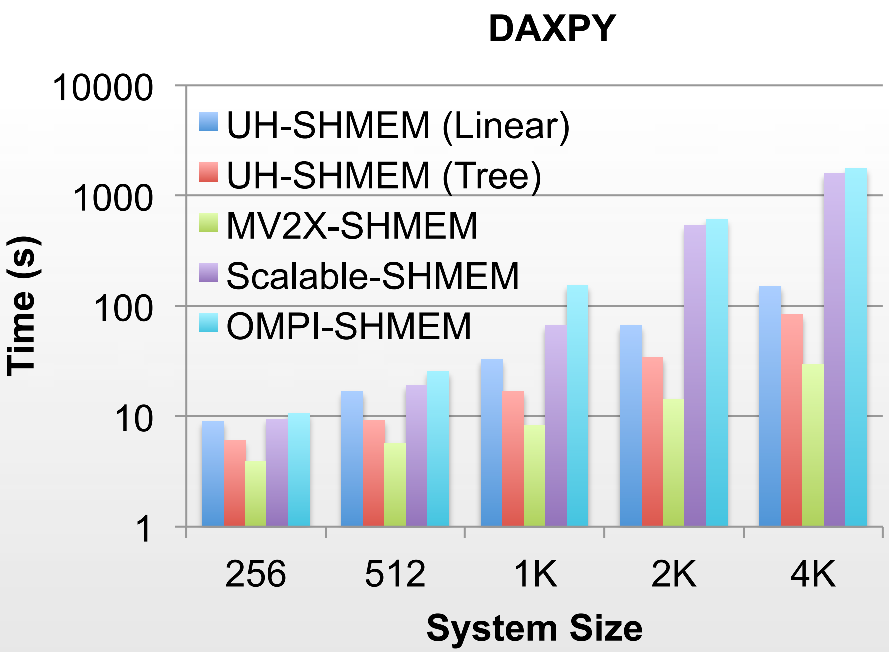
Intra-node Design Space for OpenSHMEM

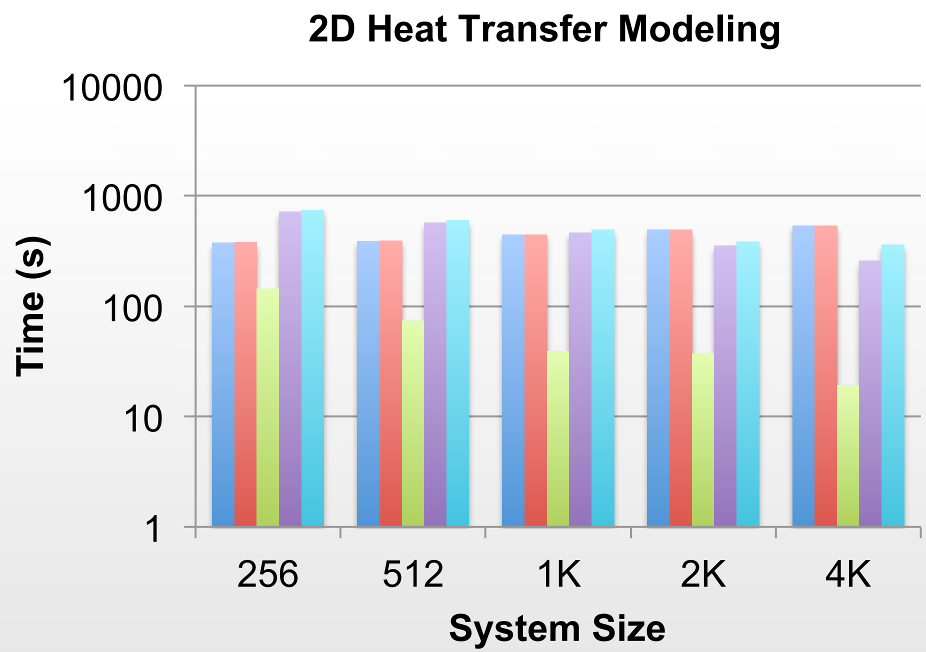


- LiMIC: kernel module developed at OSU for single copy IPC
- CMA: Cross Memory Attach - Linux 3.2 kernel feature for single copy IPC

S. Potluri, K. Kandalla, D. Bureddy, M. Li and D. K. Panda, Efficient Intranode Designs for OpenSHMEM on Multi-core Clusters, PGAS 2012

OpenSHMEM Application Performance





- DAXPY Kernel with 8K input matrix
  - 12X improved performance for 4K processes
- Heat Transfer Kernel (32K x 32K)
  - 45% improved performance for 4K processes

Ohio Supercomputer Center
An OH·TECH Consortium Member

A Hybrid MPI+PGAS Case Study: Graph 500

Incremental Approach to exploit one-sided operations

- Identify the communication critical section (mpiP, HPCToolkit)
- Allocate memory in shared address space
- Convert MPI Send/Recvs to assignment operations or one-sided operations
  - Non-blocking operations can be utilized
  - Coalescing for reducing the network operations
- Introduce synchronization operations for data consistency
  - After Put operations or before get operations
- Load balance through global view of data

Graph500 Benchmark – The Algorithm

- Breadth First Search (BFS) Traversal
- Uses 'Level Synchronized BFS Traversal Algorithm
  - Each process maintains – '*CurrQueue*' and '*NewQueue*'
  - Vertices in *CurrQueue* are traversed and newly discovered vertices are sent to their owner processes
  - Owner process receives edge information
    - If not visited; updates parent information and adds to *NewQueue*
  - Queues are swapped at end of each level
  - Initially the 'root' vertex is added to *currQueue*
  - Terminates when queues are empty

MPI-based Graph500 Benchmark

- MPI_Isend/MPI_Test-MPI_Irecv for transferring vertices
- Implicit barrier using zero length message
- MPI_Allreduce to count number *newqueue* elements
- Major Bottlenecks:
  - Overhead in send-recv communication model
    - More CPU cycles consumed, despite using non-blocking operations
    - Most of the time spent in MPI_Test
  - Implicit Linear Barrier
    - Linear barrier causes significant overheads

Hybrid Graph500 Design

- Communication and co-ordination using one-sided routines and fetch-add atomic operations
  - Every process keeps receive buffer
  - Synchronization using atomic fetch-add routines
- Level synchronization using non-blocking barrier
  - Enables more computation/communication overlap
- Load Balancing utilizing OpenSHMEM shmem_ptr
  - Adjacent processes can share work by reading shared memory

J. Jose, S. Potluri, K. Tomko and D. K. Panda, Designing Scalable Graph500 Benchmark with Hybrid MPI+OpenSHMEM Programming Models, International Supercomputing Conference (ISC '13), June 2013

Pseudo Code For Both MPI and Hybrid Versions

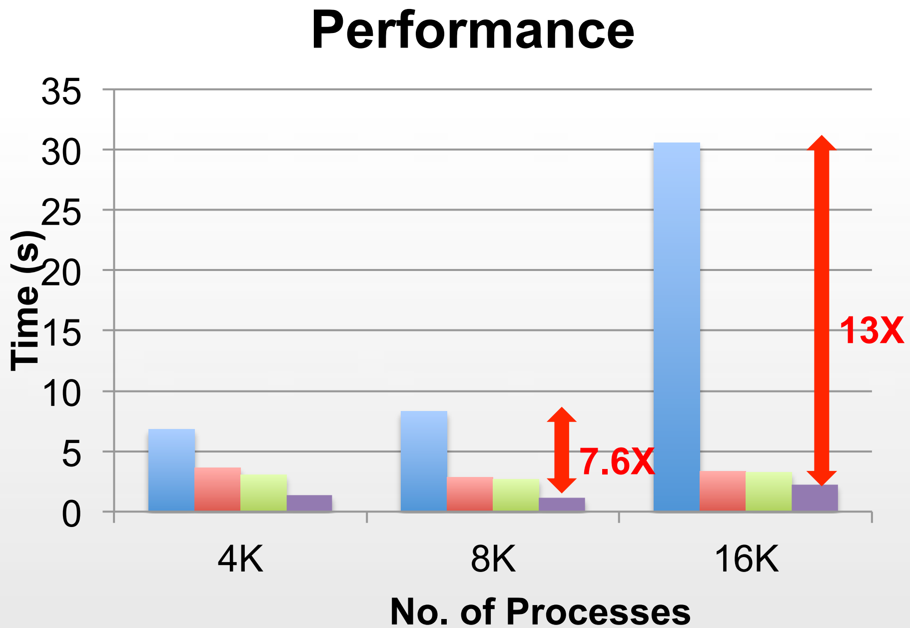
Algorithm 1: EXISTING MPI SEND/RECV

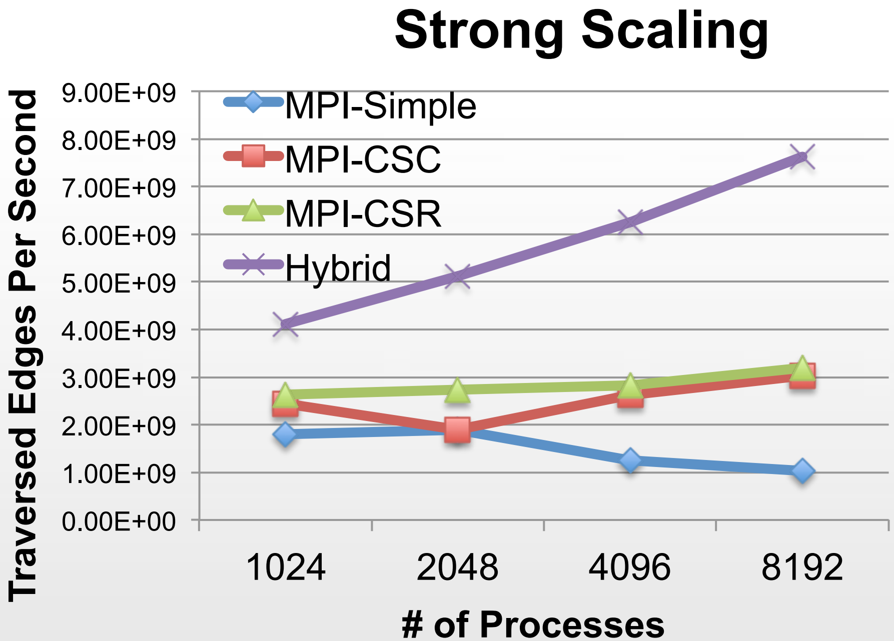
```
while true do
        while CurrQueue != NULL do
            for vertex u in CurrQueue do
             HandleReceive()
            u ← Dequeue(CurrQueue)
             Send(u, v) to owner
    end
  Send empty messages to all others
  while all_done != N − 1 do
     HandleReceive()
  end
        // Procedure: HandleReceive
 if rcv_count = 0 then
   al_ done ← all done + 1
else
   update (NewQueue, v)
```

Algorithm 2: HYBRID VERSION

```
while true do
  while CurrQueue != NULL do
    for vertex u in CurrQueue do
    u ← Dequeue(CurrQueue)
    for adjacent points to u do
     Shmem_fadd(owner, size,recv_index)
      shmem_put(owner, size,recv_buf)
     end
   end
  end
  if recv_buf[size] = done then
   Set ← 1
end
```

Graph500 - BFS Traversal Time





- Hybrid design performs better than MPI implementations
- 16,384 processes
  - 1.5X improvement over MPI-CSR
  - 13X improvement over MPI-Simple (Same communication characteristics)
- Strong Scaling

  Graph500 Problem Scale = 29, Edge Factor 16

Concluding Remarks

- Presented an overview of PGAS models and Hybrid MPI +PGAS models
- Outlined research challenges in designing an efficient runtime for these models on clusters with InfiniBand
- Demonstrated the benefits of Hybrid MPI+PGAS models for for an example application
- Hybrid MPI+PGAS model is an emerging paradigm which can lead to high-performance and scalable implementation of applications on exascale computing systems
- MVAPICH2-X http://mvapich.cse.ohio-state.edu/overview/

Networked-Based Computing Research Group Personnel

Current Students

- A. Awan (Ph.D.)
- A. Bhat (M.S.)
- S. Chakraborthy (Ph.D.)
- C.-H. Chu (Ph.D.)
- N. Islam (Ph.D.)
- M. Li (Ph.D.)
- M. Rahman (Ph.D.)
- D. Shankar (Ph.D.)
- A. Venkatesh (Ph.D.)
- J. Zhang (Ph.D.)

Current Senior Research Associates

- K. Hamidouche
- X. Lu
- H. Subramoni

Current Post-Doc

- J. Lin

Current Programmer

- J. Perkins

Current Research Specialist

- M. Arnold

Past Students

- P. Balaji (Ph.D.)
- D. Buntinas (Ph.D.)
- S. Bhagvat (M.S.)
- L. Chai (Ph.D.)
- B. Chandrasekharan (M.S.)
- N. Dandapanthula (M.S.)
- V. Dhanraj (M.S.)
- T. Gangadharappa (M.S.)
- K. Gopalakrishnan (M.S.)
- W. Huang (Ph.D.)
- W. Jiang (M.S.)
- J. Jose (Ph.D.)
- S. Kini (M.S.)
- M. Koop (Ph.D.)
- R. Kumar (M.S.)
- S. Krishnamoorthy (M.S.)
- K. Kandalla (Ph.D.)
- P. Lai (M.S.)
- J. Liu (Ph.D.)
- M. Luo (Ph.D.)
- A. Mamidala (Ph.D.)
- G. Marsh (M.S.)
- V. Meshram (M.S.)
- S. Naravula (Ph.D.)
- R. Noronha (Ph.D.)
- X. Ouyang (Ph.D.)
- S. Pai (M.S.)
- S. Potluri (Ph.D.)
- R. Rajachandrasekar (Ph.D.)
- G. Santhanaraman (Ph.D.)
- A. Singh (Ph.D.)
- J. Sridhar (M.S.)
- S. Sur (Ph.D.)
- H. Subramoni (Ph.D.)
- K. Vaidyanathan (Ph.D.)
- A. Vishnu (Ph.D.)
- J. Wu (Ph.D.)
- W. Yu (Ph.D.)

Past Post-Docs

- H. Wang
- X. Besseron
- H.-W. Jin
- M. Luo
- E. Mancini
- S. Marcarelli
- J. Vienne

Past Research Scientist

- S. Sur

Past Programmers

- D. Bureddy

Questions

Karen Tomko
Interim Director of Research
Scientific Applications Manager
Ohio Supercomputer Center
ktomko@osc.edu

1224 Kinnear Road
Columbus, OH 43212
Phone: (614) 292-2846

ohiosupercomputercenter

ohiosupercomputerctr