NETWORK-BASED COMPUTING LABORATORY

On-demand Connection Management for OpenSHMEM and OpenSHMEM+MPI

Sourav Chakraborty, Hari Subramoni, Jonathan Perkins, Ammar A. Awan, and Dhabaleswar K. Panda

Presented by: Md. Wasi-ur- Rahman

Department of Computer Science and Engineering
The Ohio State University

OHIO STATE

Overview

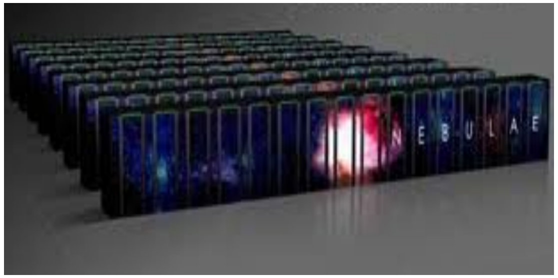
- **Introduction**
- Motivation
- Problem Statement
- Design Details
- Experimental Results
- Conclusion

Current Trends in HPC

- Supercomputing systems scaling rapidly
  - Multi-core architectures and
  - High-performance interconnects
- InfiniBand is a popular HPC interconnect
  - 224 systems (44.8%) in top 500
- PGAS and hybrid MPI+PGAS models becoming increasingly popular
- Supporting frameworks (e.g. Job Launchers) also need to become more scalable to handle this growth

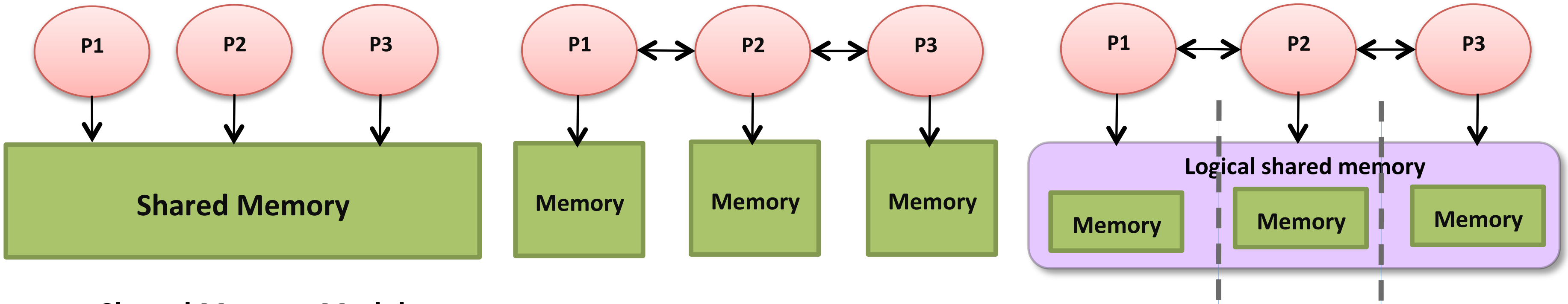
Stampede@TACC

SuperMUC@LRZ



Nebulae@NSCS

Parallel Programming Models

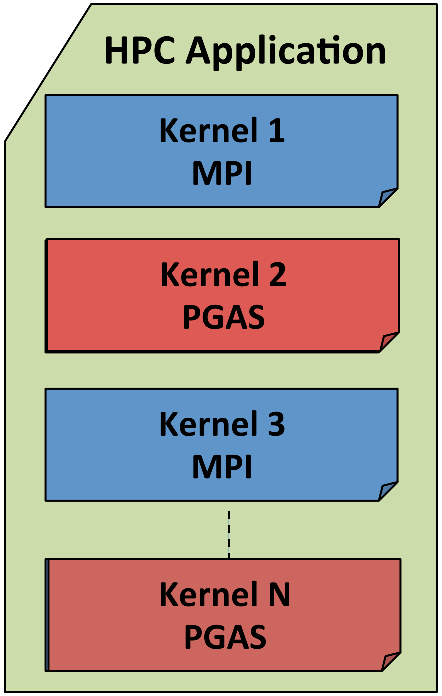


Shared Memory Model	Distributed Memory Model	Partitioned Global Address Space (PGAS)
OpenMP	MPI (Message Passing Interface)	Global Arrays, UPC, OpenSHMEM, CAF, …

- Key features of PGAS models –
  - Simple shared memory abstractions
  - Light weight one-sided communication
  - Easier to express irregular communication
- Different approaches to PGAS –
  - Languages – UPC, CAF, X10, Chapel
  - Library – OpenSHMEM, Global Arrays

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[1]:
  - "Hybrid Programming is a practical way to program exascale systems"

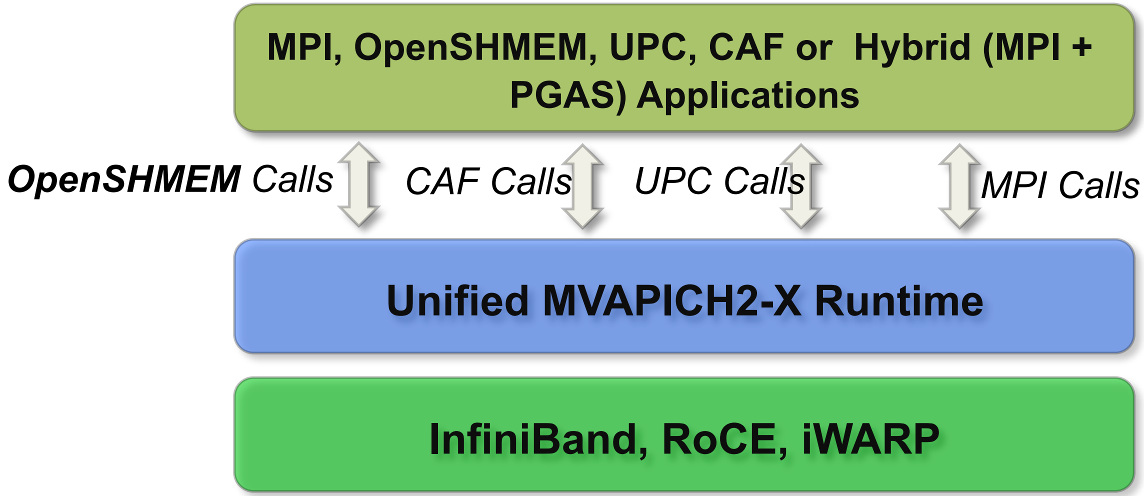


[1] The International Exascale Software Roadmap, Dongarra, J., Beckman, P. et al., Volume 25, Number 1, 2011, International Journal of High Performance Computer Applications, ISSN 1094-3420

MVAPICH2 Software

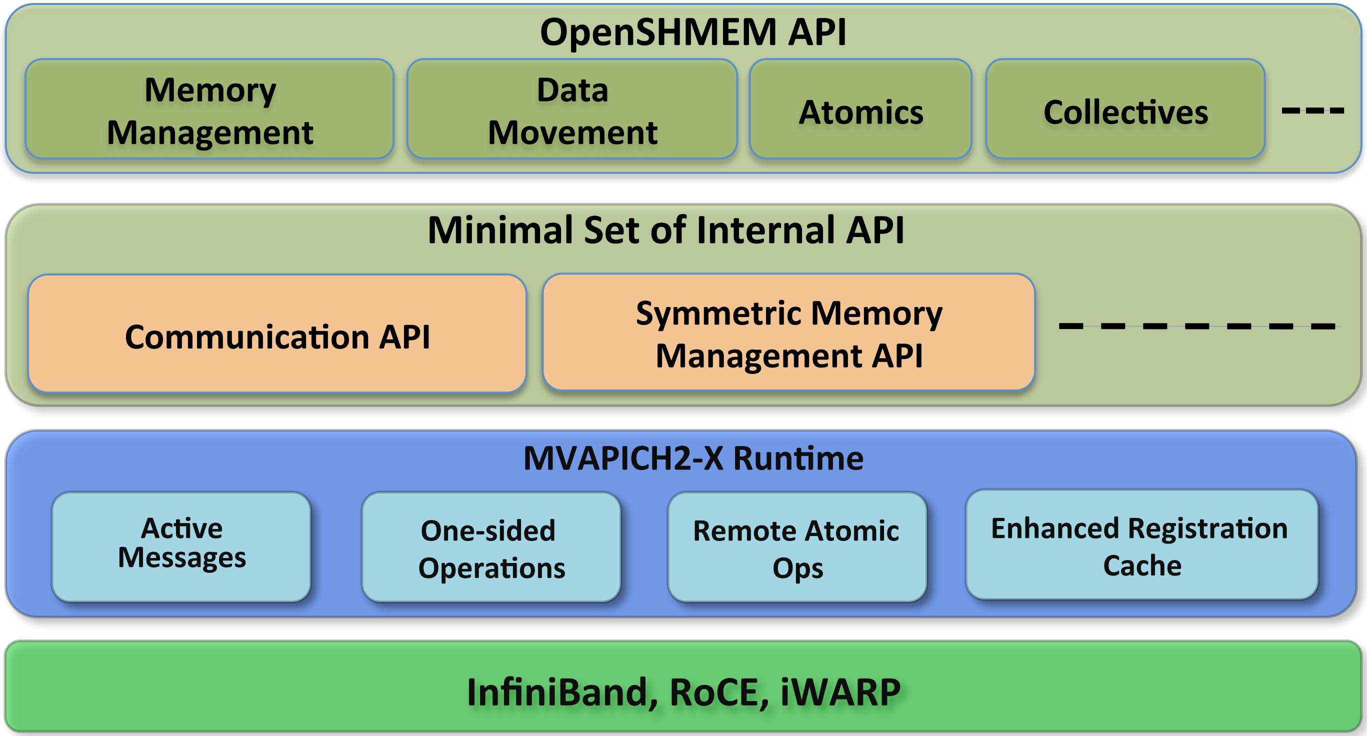
- High Performance open-source MPI Library for InfiniBand, 10Gig/iWARP, and RDMA over Converged Enhanced Ethernet (RoCE)
  - MVAPICH (MPI-1), MVAPICH2 (MPI-2.2 and MPI-3.0), Available since 2002
  - MVAPICH2-X (MPI + PGAS), Available since 2012
  - Support for GPGPUs (MVAPICH2-GDR) and MIC (MVAPICH2-MIC), Available since 2014
  - Used by more than 2,375 organizations in 75 countries
  - More than 259,000 downloads from OSU site directly
  - Empowering many TOP500 clusters (Nov '14 ranking)
    - 7th ranked 519,640-core cluster (Stampede) at TACC
    - 11th ranked 160,768-core cluster (Pleiades) at NASA
    - 15th ranked 76,032-core cluster (Tsubame 2.5) at Tokyo Institute of Technology and many others
  - Available with software stacks of many IB, HSE, and server vendors including Linux Distros (RedHat and SuSE)
  - http://mvapich.cse.ohio-state.edu

MVAPICH2-X for Hybrid MPI + PGAS Applications



- Unified communication runtime for MPI, OpenSHMEM, UPC, CAF available with MVAPICH2-X 1.9 (2011) onwards!
  - Supports MPI(+OpenMP), OpenSHMEM, UPC, CAF, MPI(+OpenMP) + OpenSHMEM
  - MPI-3 compliant, OpenSHMEM v1.0 standard compliant, UPC v1.2 standard compliant (with initial support for UPC 1.3), CAF 2008 standard (OpenUH)
  - Scalable Inter-node and intra-node communication – point-to-point and collectives

OpenSHMEM Design in MVAPICH2-X (Prior Work)



- OpenSHMEM Stack based on OpenSHMEM Reference Implementation
- OpenSHMEM Communication over MVAPICH2-X Runtime
  - Improves performance and scalability of pure OpenSHMEM and hybrid MPI+OpenSHMEM applications[2]

[2] 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.

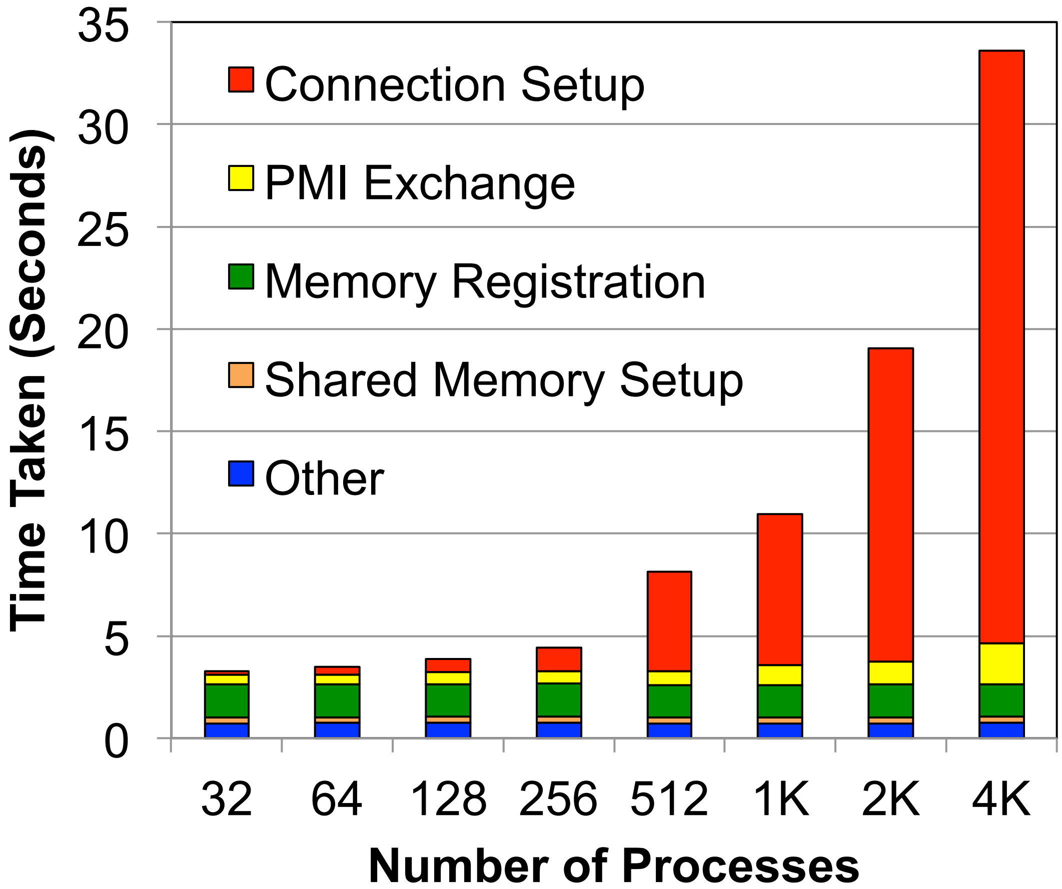
Overview

- Introduction
- **Motivation**
- Problem Statement
- Design Details
- Experimental Results
- Conclusion

Why is Fast Startup Important

- Developing and debugging
  - Developers spend a lot of time launching the application
  - Reducing job launch time saves developer-hours
- Regression testing
  - Complex software have a lot of features to test
  - Large number of short-running tests need to be launched
- System testing
  - Full-system size jobs to stress-test the network and software
- Checkpoint-restart
  - An application restart is similar to a launching a new job
  - Faster startup means less time recovering from a failure

Breakdown of Time Spent in OpenSHMEM Initialization



- Connection setup time the dominant factor
- PMI Exchange cost also increases at scale
- Other costs relatively constant

- All numbers taken on TACC Stampede with 16 processes per node
- MVAPICH2-X 2.1rc1 based on OpenSHMEM 1.0h and GASNet version 1.24.0

Communication Pattern in Common OpenSHMEM and Hybrid Applications

Application	Number of Processes	Average Number of Peers
BT	64	8.7
	1024	10.6
EP	64	3
	1024	5.01
MG	64	9.46
	1024	11.9
SP	64	8.75
	1024	10.7
2D Heat	64	5.28
	1024	5.40

- Current OpenSHMEM runtimes establish all-to-all connectivity
- Each process communicates with only a small number of peers
- Establishing all-to-all connectivity is unnecessary and wasteful
  - Takes more time
  - Consumes memory
  - Can impact performance of the HCA

Overview

- Introduction
- Motivation
- **Problem Statement**
- Design Details
- Experimental Results
- Conclusion

Problem Statement

- Each OpenSHMEM process registers memory segments with the HCA and broadcasts the segment information
  - Forces setting up all-to-all connectivity
  - Extra message transfer causes overhead
- OpenSHMEM uses global barriers during initialization
  - Causes connections to be established
  - Unnecessary synchronization between processes
- Does not take advantage of recently proposed non-blocking PMI extensions[3]
  - No overlap between PMI exchange and other operations

- Can we enhance the existing OpenSHMEM runtime design to address these challenges and improve the startup performance and scalability of pure OpenSHMEM and hybrid MPI+OpenSHMEM programs?

[3] Non-blocking PMI Extensions for Fast MPI Startup. S. Chakraborty, H. Subramoni, A. Moody, A. Venkatesh, J. Perkins and D. K. Panda CCGrid '15, May 2015

Overview

- Introduction
- Motivation
- Problem Statement
- **Design Details**
- Experimental Results
- Conclusion

Addressing the Challenges

1. All-to-all connectivity
   – On-demand connection setup scheme
2. Global Barrier Synchronization
   – Shared memory based intra-node barrier
3. PMI Exchange cost
   – Non-blocking PMI extensions

Connection Management in InfiniBand

- InfiniBand is a low-latency, high-bandwidth switched fabric interconnect widely used in high performance computing clusters
- Provides different transport protocols
  - RC: Reliable, connection oriented, requires one endpoint (QP) per peer
  - UD: Unreliable, connectionless, requires only one QP for all peers
- Requires an out-of-band channel to exchange connection information before in-band communication
- Provides Remote Direct Memory Access (RDMA) capabilities
  - Fits well with one sided semantics of OpenSHMEM
  - Only RC protocol is supported
  - Requires memory to be pre-registered with the HCA
- The initiating process needs to obtain the address, size, and an identifier (remote_key/rkey) from the target process

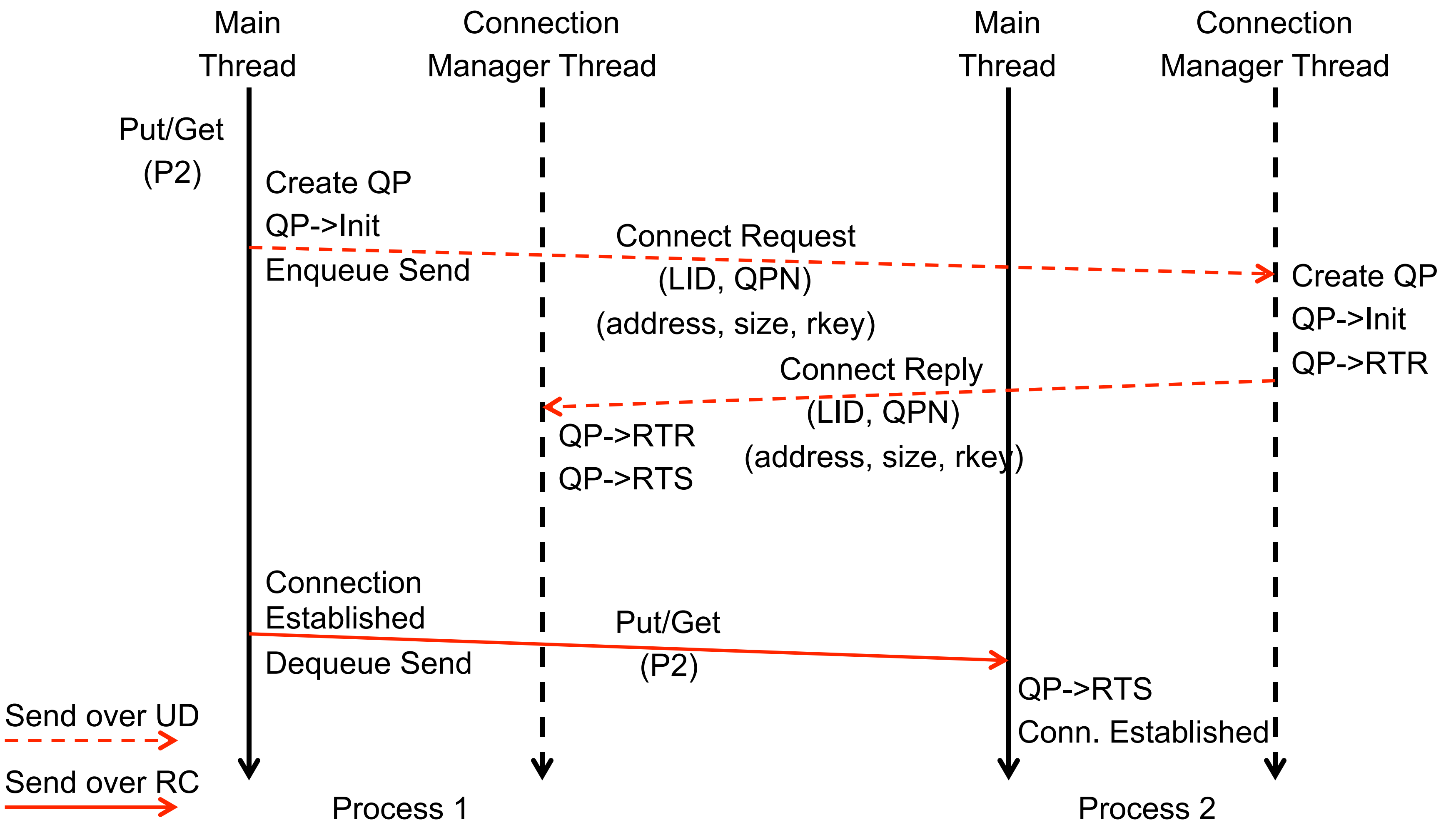
RDMA Communication in MVAPICH2-X

- PMI provides a key-value store, acts as the out-of-band channel for InfiniBand
  - Each process opens a UD endpoint and puts its address into the key-value store using PMI Put
  - PMI Fence broadcasts this information to other processes
- When a process P1 wants to communicate with another process P2
  - P1 looks up the UD address of P2 using PMI Get
  - P1 opens a RC endpoint and sends the address to P2 using UD
  - P2 also opens a corresponding RC endpoint and replies with its address to P1 over UD
  - P1 and P2 enables the RC connection and can do send/recv
  - P1, P2 exchange segment information (<address, size, rkey>)
  - P1 can do RDMA read/write operations from memory of P2

Supporting On-demand Connection Setup

- Each process no longer broadcasts the segment information (<address, size, rkey>)
- Segment information is serialized and stored in a buffer
  - Combined with the connect request/reply messages
  - Connection is established only when required
  - Overhead is reduced as one extra message is eliminated
- The connect request and reply messages are transmitted over the connectionless UD protocol
  - Underlying conduit (mvapich2x) guarantees reliable delivery

On-demand Connection Setup in GASNet-mvapich2x conduit



Shared Memory Based Intra-node Barrier

- A global barrier with P processes –
  - Requires at least O(log(P)) connections
  - Takes at least O(log(P)) time
  - Forces unnecessary synchronization
- With on-demand connection setup mechanism, global barriers are no longer required
  - Intra-node barriers are still necessary
- Replace global barriers with shared memory based intra-node barriers
- Requires the underlying conduit to handle message timeout and retransmissions

Using Non-blocking PMI Extensions[3]

Current

```
start_pes() {
   PMI2_KVS_Put();
   PMI2_KVS_Fence();
   /* Do unrelated
      tasks */
}
connect() {
   PMI2_KVS_Get();
   /* Use values */
}
```

Proposed

```
start_pes() {
   PMIX_Iallgather();
   /* Do unrelated
      tasks */
}

connect() {
   PMIX_Wait();
   /* Use values */
}
```

[3] Non-blocking PMI Extensions for Fast MPI Startup. S. Chakraborty, H. Subramoni, A. Moody, A. Venkatesh, J. Perkins and D. K. Panda CCGrid '15, May 2015

Using Non-blocking PMI Extensions

- PMI is used to exchange the UD endpoint addresses
- Different initialization related tasks can be overlapped with the PMI exchange
  - Registering memory with the HCA
  - Setting up shared memory channels
  - Allocating resources
- The data exchanged through PMI is only required when a process tries to communicate with another process. Many applications perform computation between start_pes and the first communication
  - Reading input files
  - Preprocessing the input
  - Dividing the problem into sub-problems

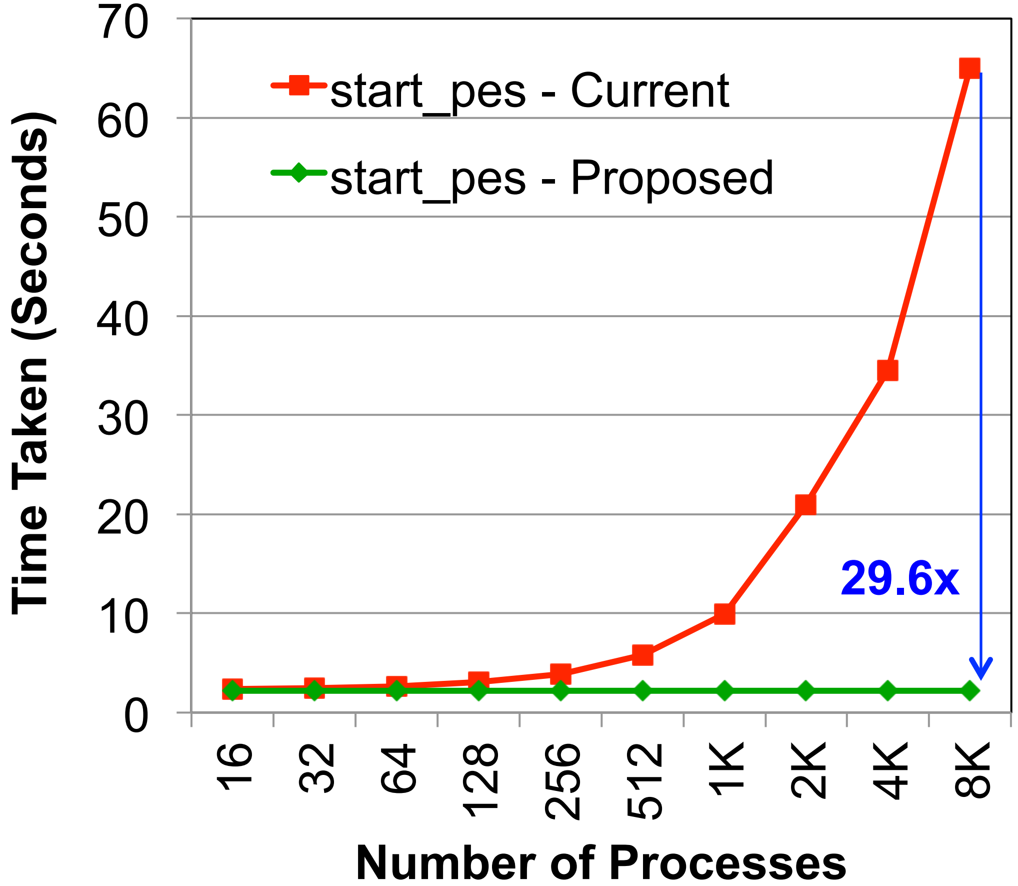
Overview

- Introduction
- Motivation
- Problem Statement
- Design Details
- **Experimental Results**
- Conclusion

Experimental Setup

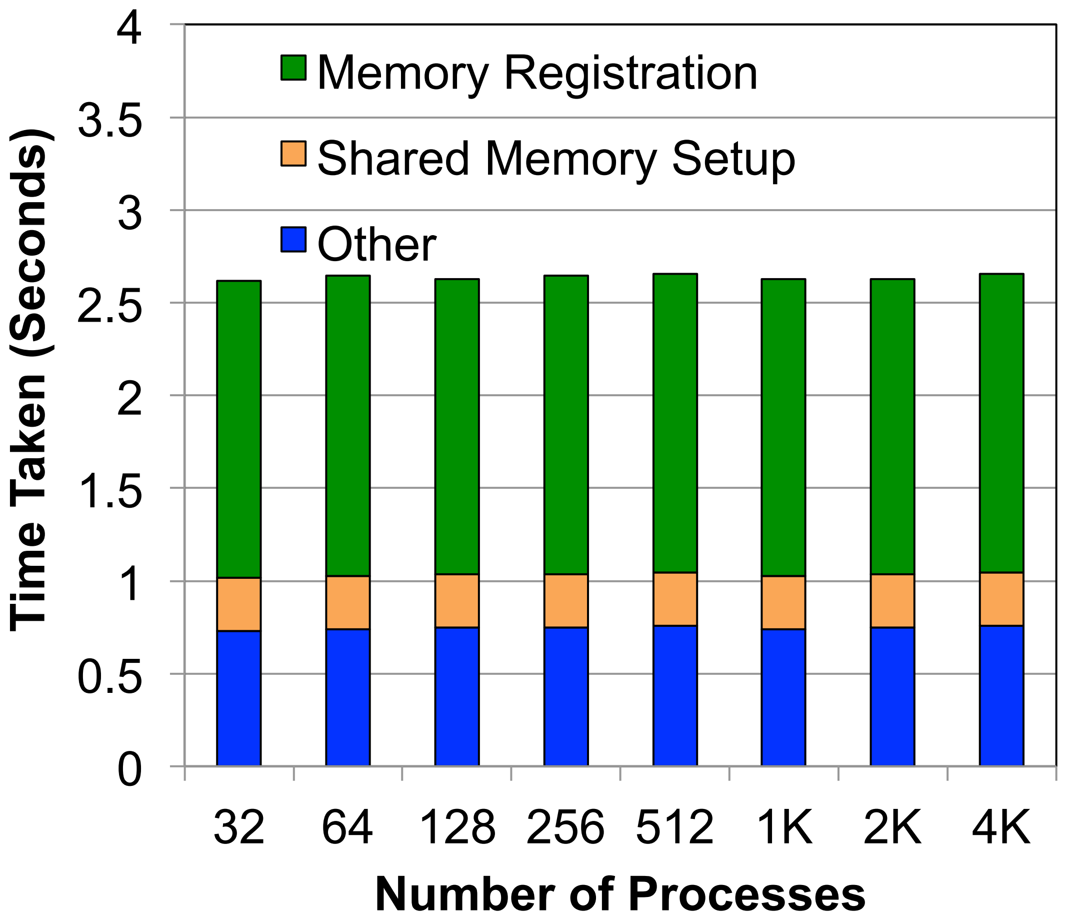
- **Cluster-A**
  - 2.67 GHz Intel Westmere dual socket, quad-core processors
  - 12 GB Physical memory per node
  - Mellanox MT26428 QDR Connect-X HCAs (32 Gbps data rate)
- **Cluster-B** (TACC Stampede)
  - 2.70 GHz Intel SandyBridge dual socket, eight-core processors
  - 32 GB physical memory per node
  - MT4099 FDR ConnectX-3 HCAs (54 Gbps data rate)
- All evaluations performed with MVAPICH2-X 2.1rc1, based on
  - OpenSHMEM reference implementation 1.0h
  - GASNet version 1.24.0
- OSU Microbenchmark Suite 4.4
  - Enhanced to measure OpenSHMEM initialization performance

Performance of OpenSHMEM Initialization



- Constant initialization time at any scale using non-blocking PMI calls
- start_pes() performs 29.6 times faster with 8,192 processes
- Taken on Cluster-B (TACC Stampede)
- 16 Processes per node

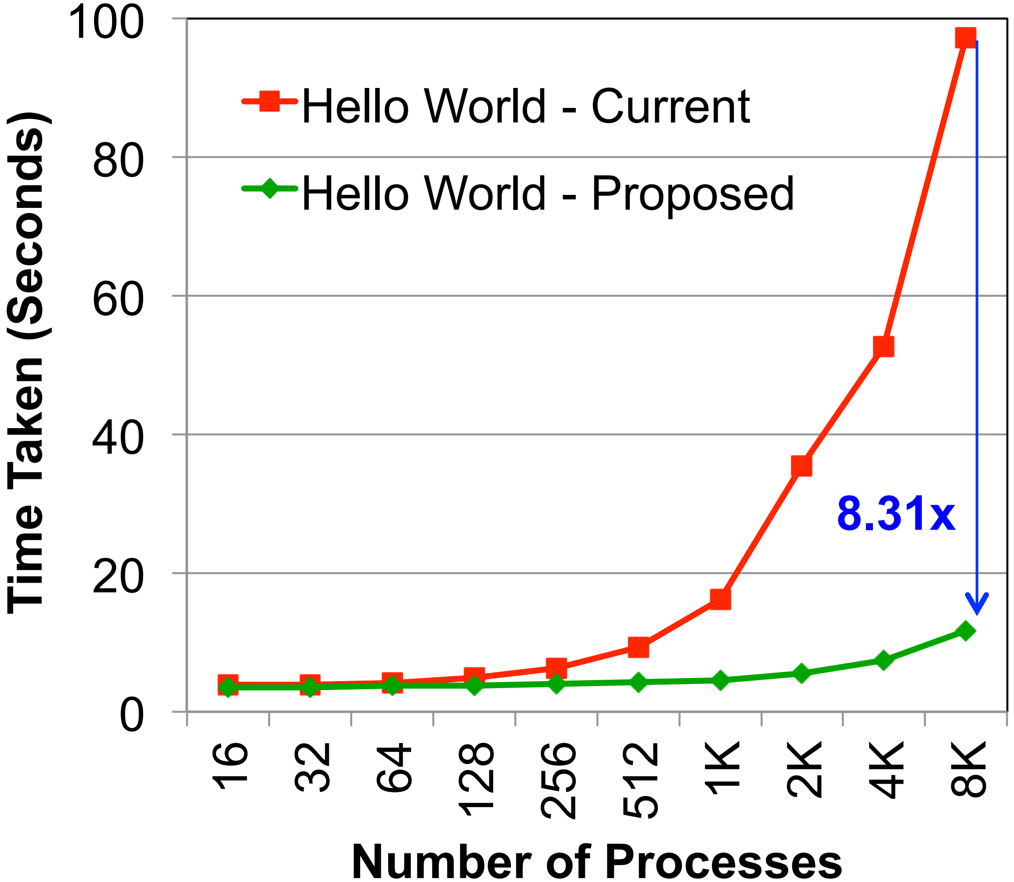
Breakdown of Time Spent in OpenSHMEM Initialization



- Connection setup cost eliminated
- PMI exchange cost overlapped
- Constant initialization cost at any scale

- "Other" costs include opening HCA, reading configuration files etc.
- Taken on Cluster-B (TACC Stampede)
- 16 Processes per node

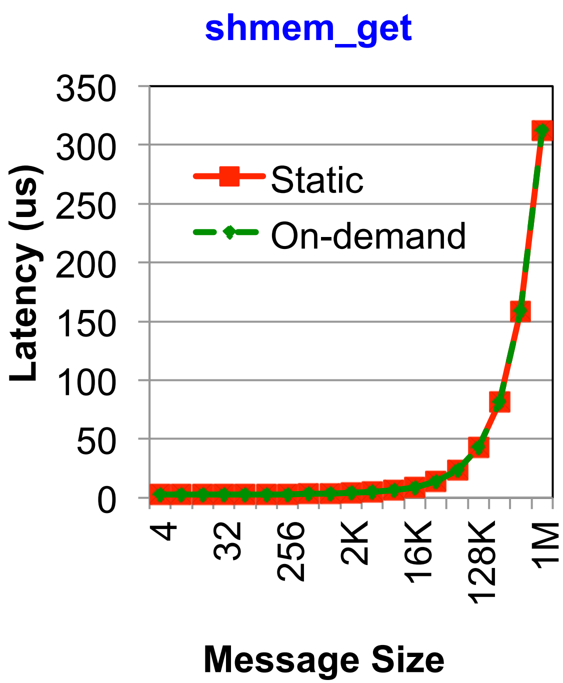
Performance of OpenSHMEM Hello World

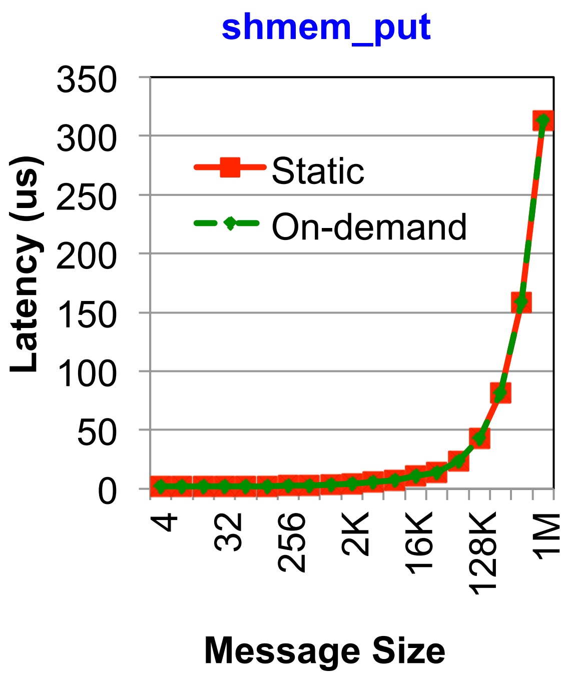


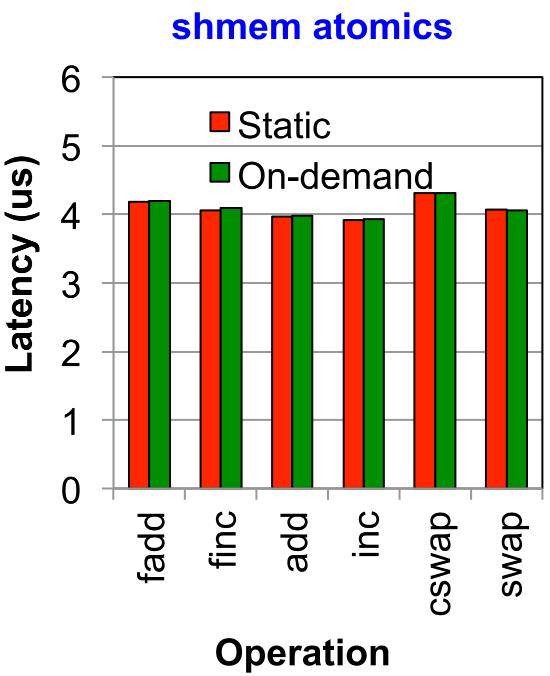
- Performance of Hello World improved by 8.31 times with 8,192 processes

- Taken on Cluster-B (TACC Stampede)
- 16 Processes per node

Overhead of On-demand Connection Setup on Performance of Point-to-Point Operations

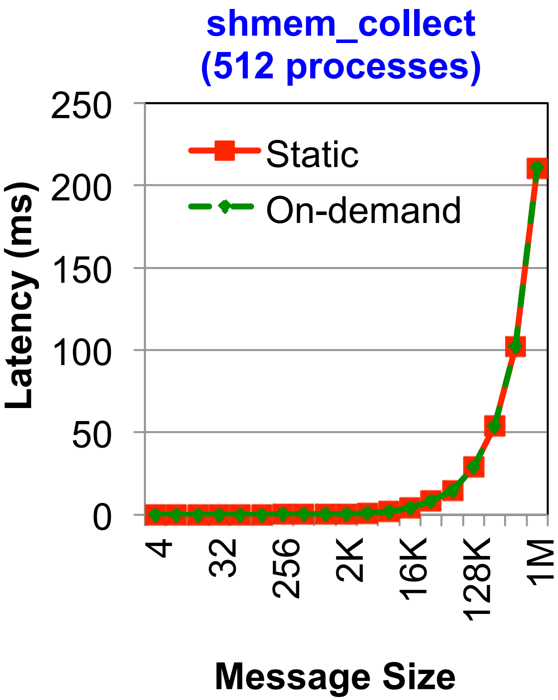


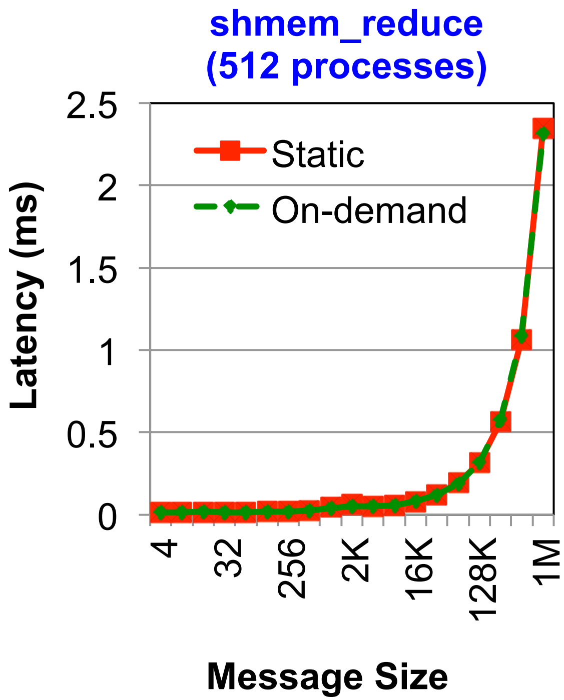


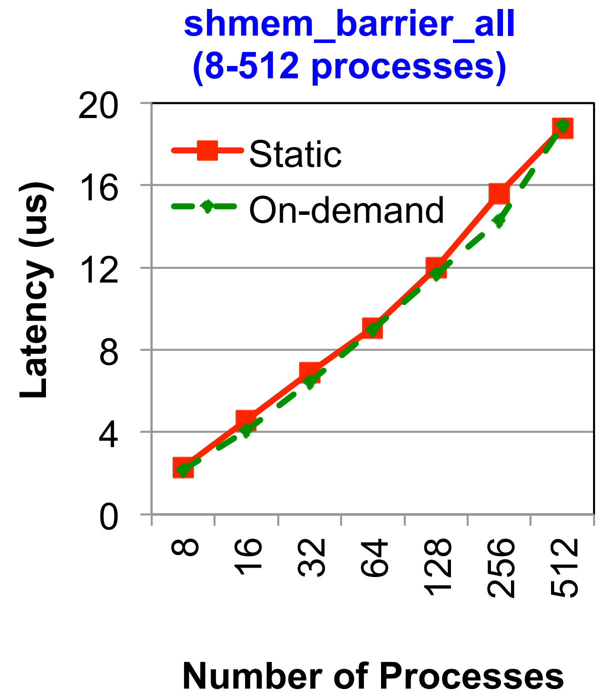


- Average of 1,000 iterations, 5 runs each
- Identical performance with static connection setup
- Taken on Cluster-A

Overhead of On-demand Connection Setup on Performance of Collective Operations

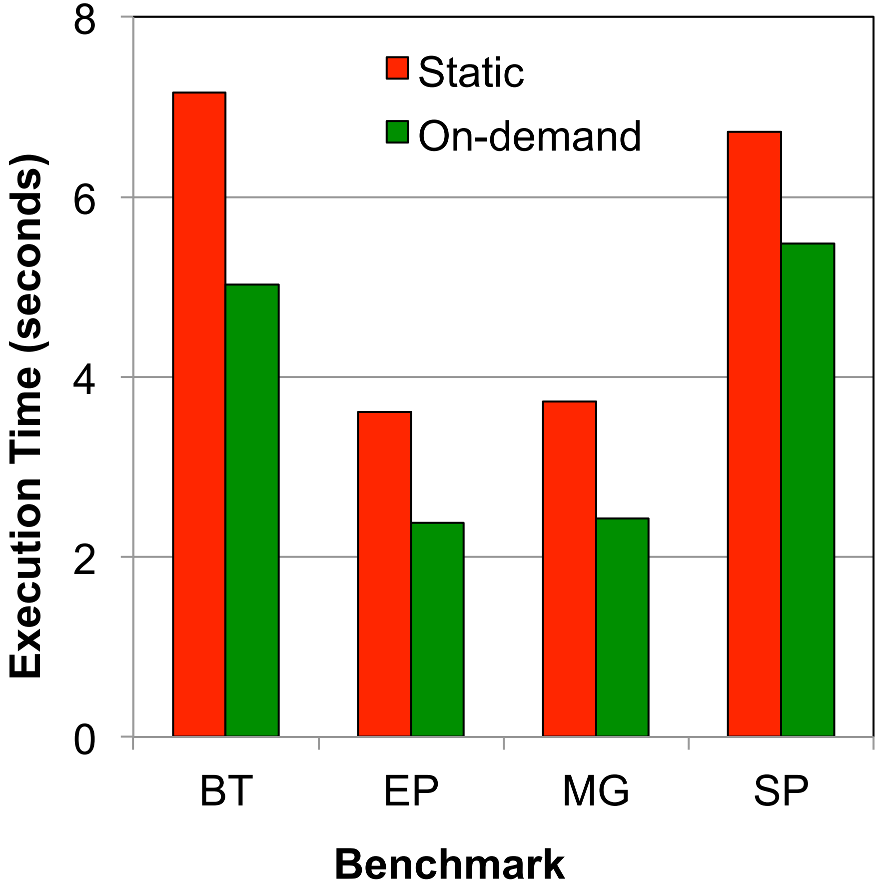






- Average of 1,000 iterations, 5 runs each
- Identical performance with static connection setup
- Taken on Cluster-A, 8 processes per node

Performance of Pure OpenSHMEM Applications (NAS Parallel Benchmarks)[4]

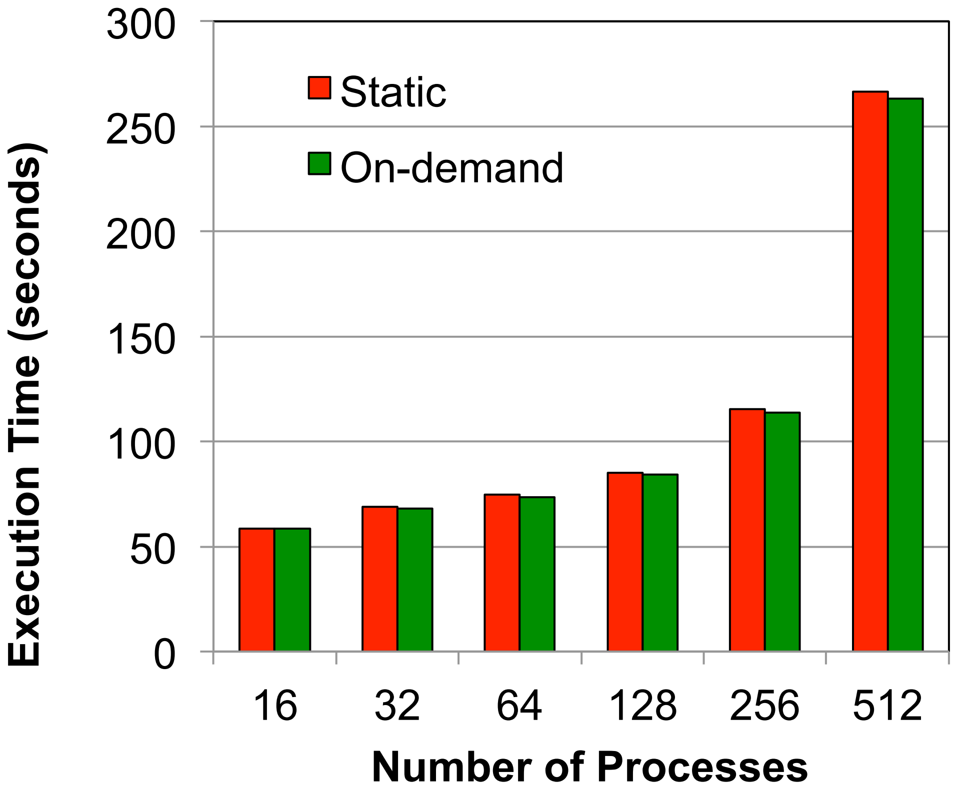


- Improvement observed depends on -
  - Average number of communicating peers
  - Time spent in computation before first communication
- 18-35% improvement in total execution time (reported by job launcher)

- 256 Processes
- Cluster-A
- 8 Processes per Node
- Class B data

[4] OpenSHMEM implementation of NAS Parallel Benchmarks available at https://github.com/openshmem-org/openshmem-npbs

Overhead on Performance of Hybrid MPI +OpenSHMEM Application (Graph500)

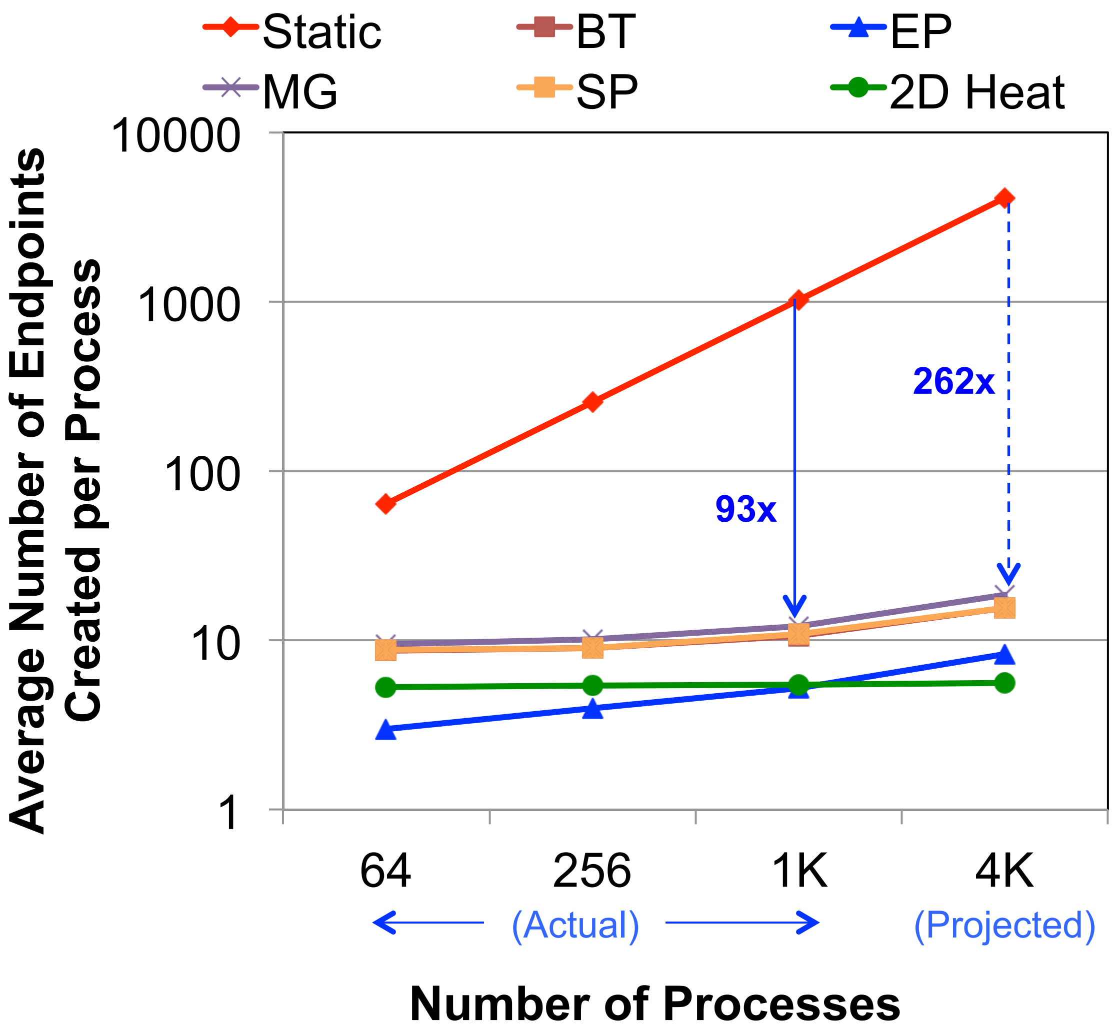


- MPI+OpenSHMEM implementation of Graph500[5] used
- Both static and on-demand connection setup schemes show identical performance

- Cluster-A
- 8 Processes per Node
- 1,024 Vertices
- 16,384 Edges

[5] J. Jose, S. Potluri, K. Tomko, and D. K. Panda, "Designing Scalable Graph500 Benchmark with Hybrid MPI+OpenSHMEM Programming Models," in ISC '13

Resource Usage and Scalability



- Number of connections limited by what is required
- More than 90% (8x-100x) reduction in number of connections and associated resources across applications
- Taken on Cluster-A
- 8 Processes per node

Overview

- Introduction
- Motivation
- Problem Statement
- Design Details
- Experimental Results
- **Conclusion**

Conclusion

- Static connection establishment is unnecessary and wasteful
- On-demand connection management in OpenSHMEM improves performance and saves memory
- start_pes can be completed in constant time at any scale using recently proposed non-blocking PMI extensions
- start_pes is 29.6x faster with 8,192 processes
- Hello World is 8.3x faster with 8,192 processes
- Total execution time of NAS benchmarks reduced by up to 35% with 256 processes
- Number of connections and endpoints reduced by > 90% (up to 100 times with 1,024 processes)
- Proposed designs already available since MVAPICH2-X 2.1rc1
- Support for UPC and other PGAS languages coming soon!

Thank you!

{chakrabs, subramon, perkinjo, awan, panda}
@cse.ohio-state.edu

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

THE OHIO STATE UNIVERSITY

MVAPICH
MPI, PGAS and Hybrid MPI+PGAS Library