MVAPICH
MPI, PGAS and Hybrid MPI+PGAS Library

Efficient Asynchronous Communication Progress for MPI without Dedicated Resources

Amit Ruhela, Hari Subramoni, Sourav Chakraborty,

Mohammadreza Bayatpour, Pouya Kousha, and Dhabaleswar K. Panda.

{ ruhela.2, subramoni.1, chakraborty.52, bayatpour.1, kousha.2, panda.2 } @ osu.edu

Department of Computer Science and Engineering

THE OHIO STATE UNIVERSITY

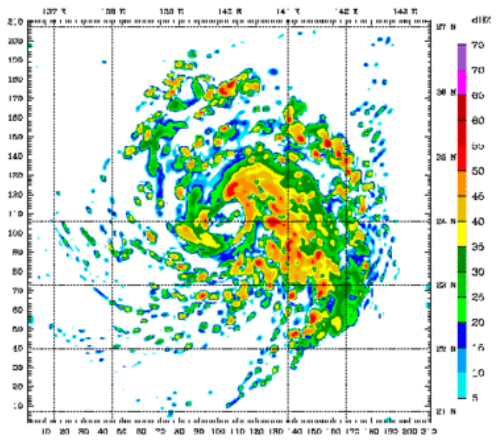
COLLEGE OF ENGINEERING

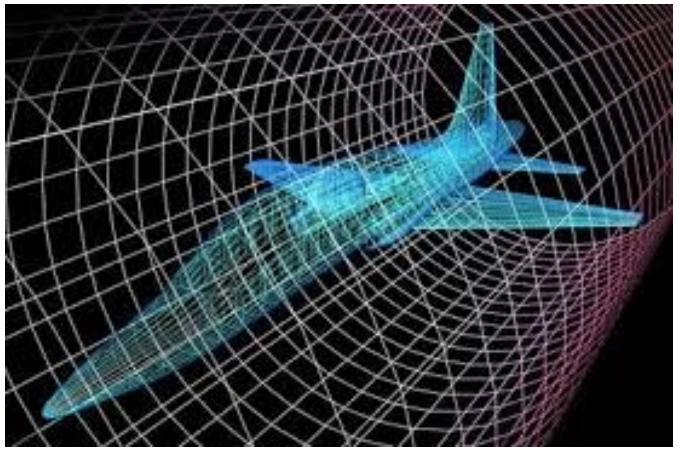
Outline

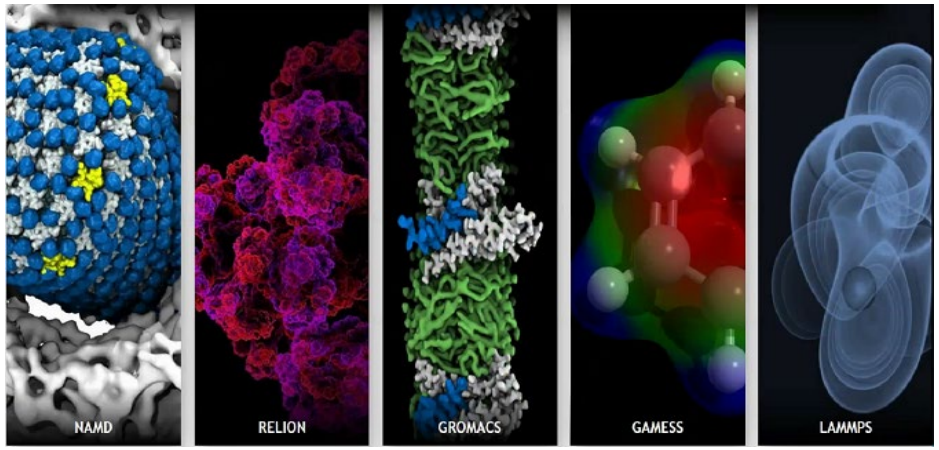
- Introduction
- Motivation
- Contributions
- Design Methodology
- Experimental Results
- Conclusions

Current and Next Generation Applications

- Growth of High Performance Computing
  - Growth in processor performance
    - Chip density doubles every 18 months
  - Growth in commodity networking
    - Increase in speed/features + reducing cost
- Clusters: popular choice for HPC
  - Scalability, Modularity and Upgradeability





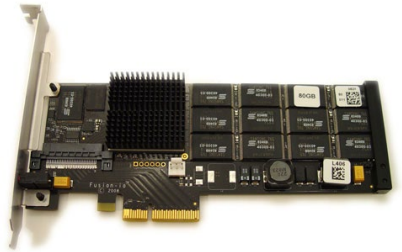


Drivers of Modern HPC Cluster Architectures - Hardware

Multi-/Many-core Processors

High Performance Interconnects – InfiniBand (with SR-IOV)
<1usec latency, 200Gbps Bandwidth>

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



SSD, NVMe-SSD, NVRAM

- Multi-core/many-core technologies
- Remote Direct Memory Access (RDMA)-enabled networking (InfiniBand and RoCE)
  - Single Root I/O Virtualization (SR-IOV)
- Solid State Drives (SSDs), NVM, Parallel Filesystems, Object Storage Clusters
- Accelerators (NVIDIA GPGPUs and Intel Xeon Phi)

Sierra@LLNL

Stampede2@TACC

Comet@SDSC

Drivers of Modern HPC Cluster Architectures - MPI

Major MPI features

- Point -to-point two-sided communication
- Collective Communication
- One-sided Communication

Message Passing Interface (MPI)

- MVAPICH2
- OpenMPI, IntelMPI, CrayMPI, IBM Spectrum MPI
- And many more...

CRAY
THE SUPERCOMPUTER COMPANY

MVAPICH

IBM

intel

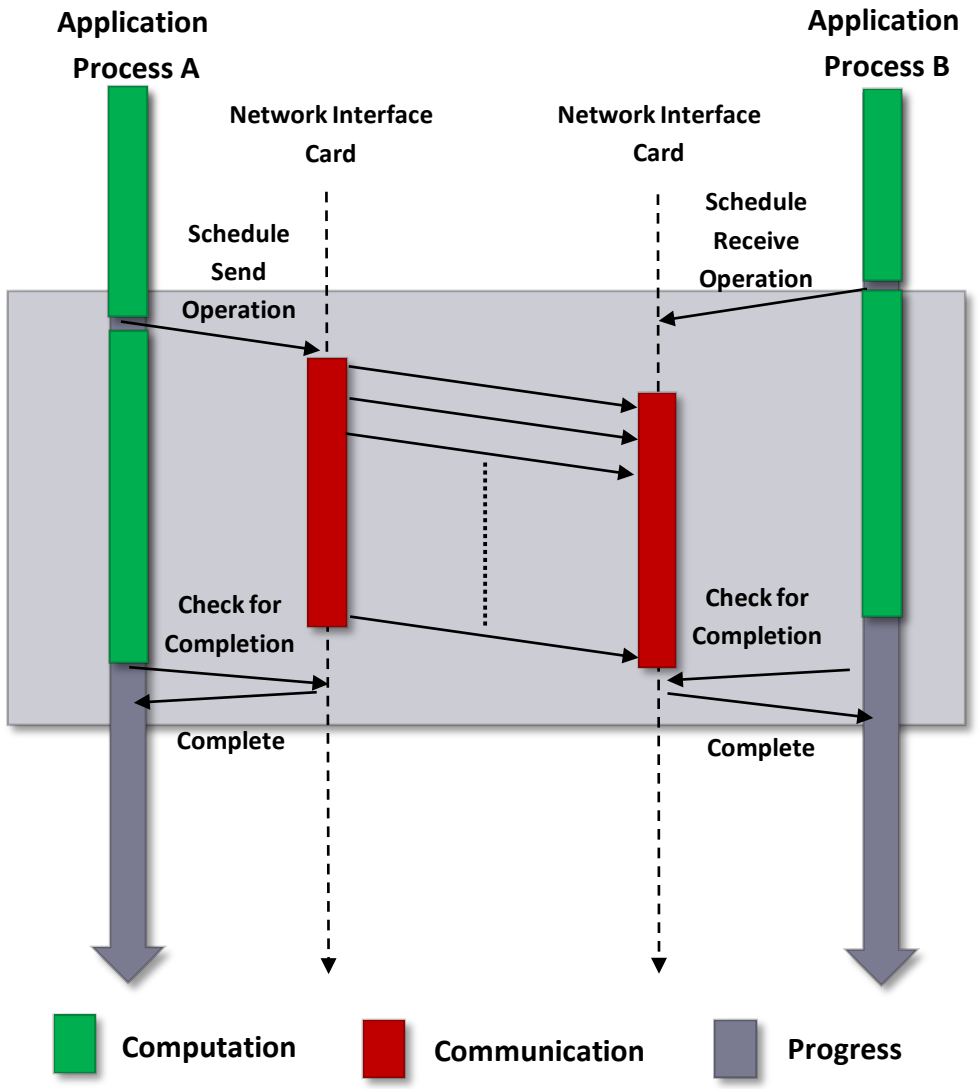
Point-to-point Communication Protocols in MPI

- Eager
  - Asynchronous protocol that allows a send operation to complete without acknowledgement from a matching receive
  - Best communication performance for smaller messages

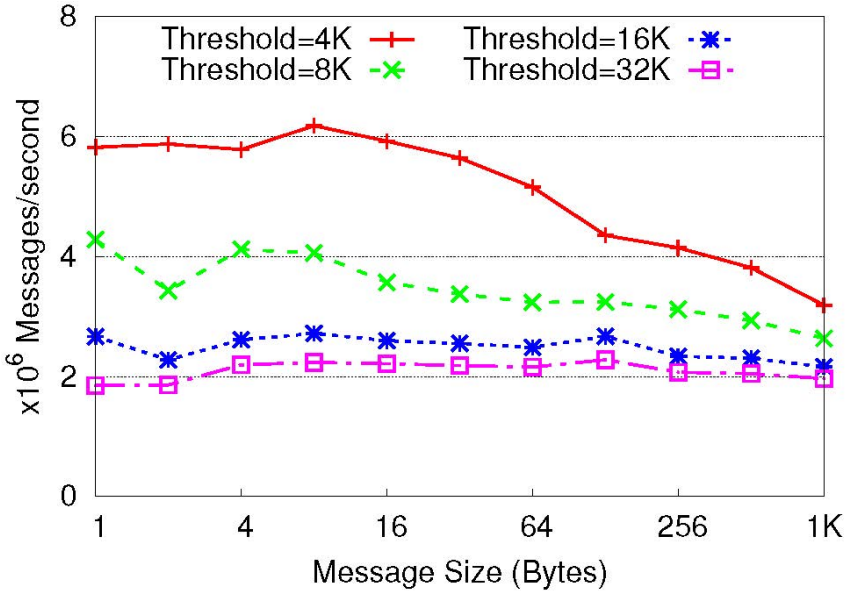
- Rendezvous
  - Synchronous protocol which requires an acknowledgement from a matching receive in order for the send operation to complete
  - Best communication performance for larger messages

- But what about overlap?

Analyzing Overlap Potential of Eager Protocol

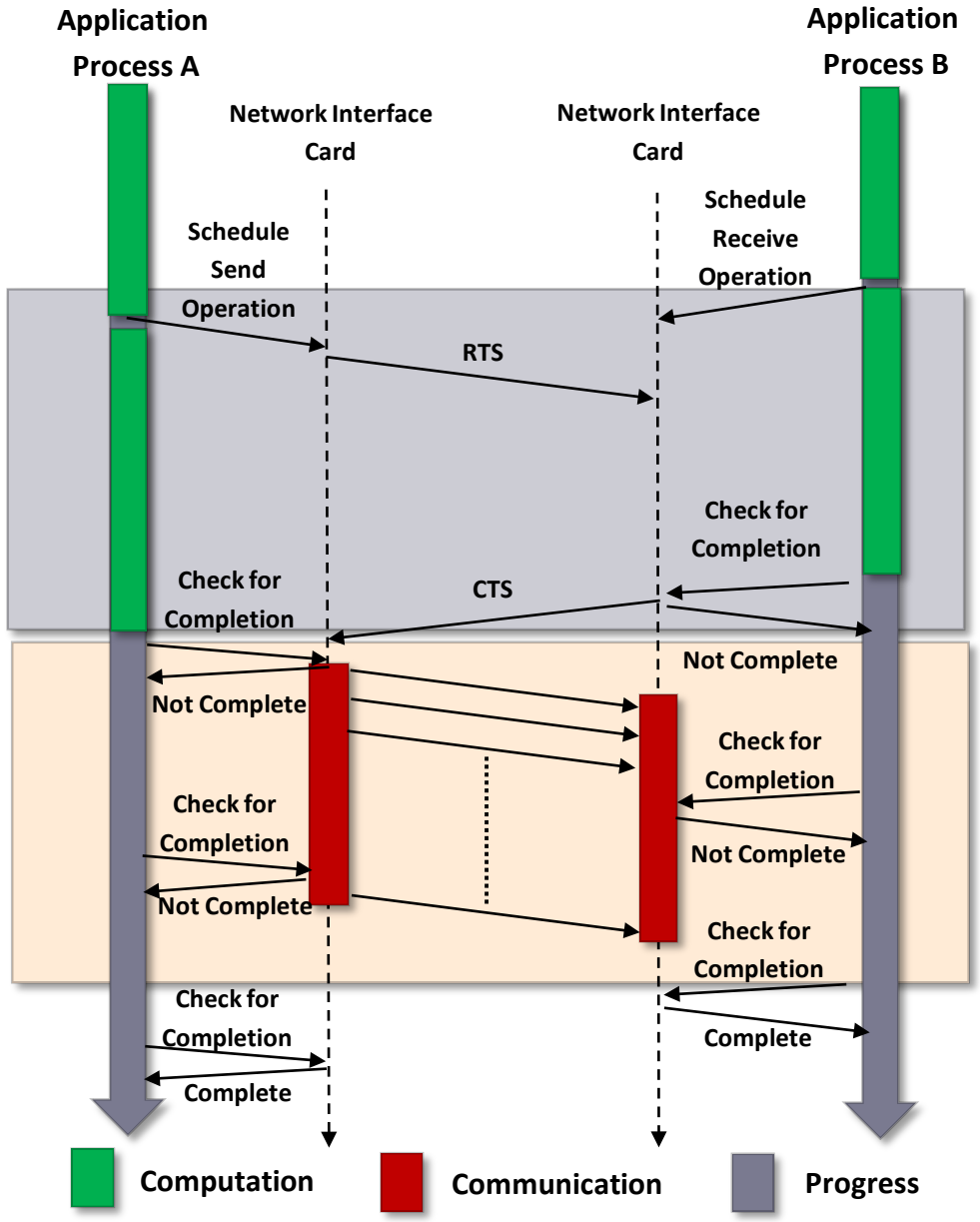


- Application processes schedule communication operation
- Network adapter progresses communication in the background
- Application process free to perform useful compute in the foreground
- **Overlap of computation and communication => Better Overall Application Performance**
- **Increased buffer requirement**
- **Poor communication performance if used for all types of communication operations**



Impact of changing Eager Threshold on performance of multi-pair message-rate benchmark with 32 processes on Stampede

Analyzing Overlap Potential of Rendezvous Protocol



- Application processes schedule communication operation
- Application process free to perform useful compute in the foreground
- Little communication progress in the background
- All communication takes place at final synchronization

- **Reduced buffer requirement**
- **Good communication performance if used for large message sizes and operations where communication library is progressed frequently**

- **Poor overlap of computation and communication => Poor Overall Application Performance**

Outline

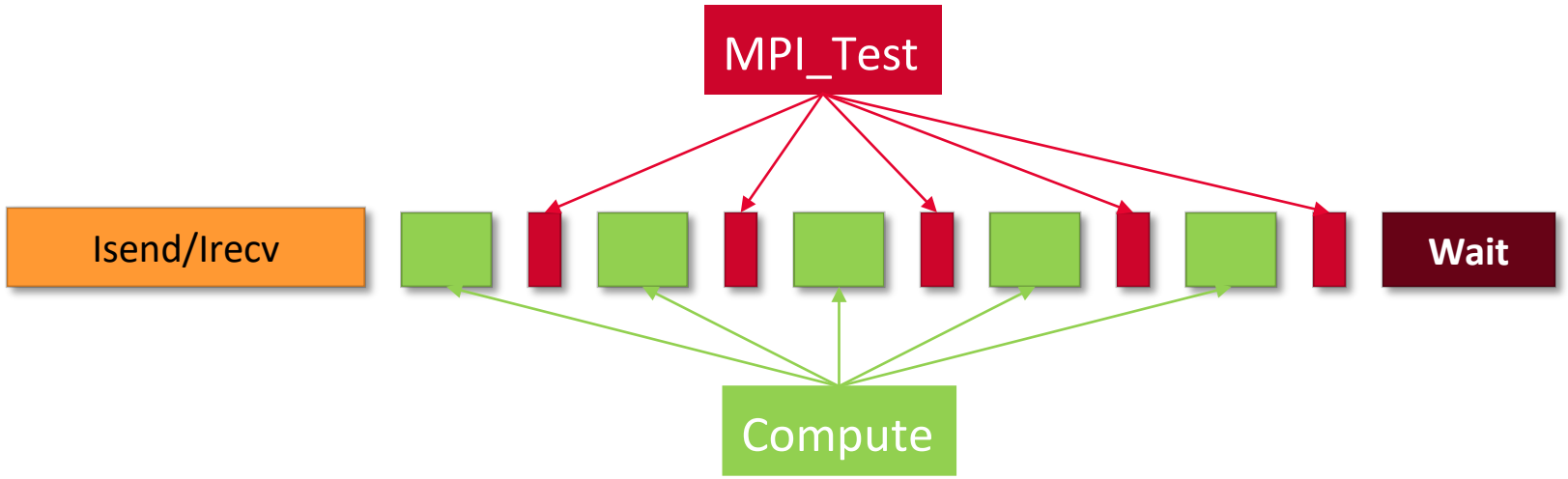
- Introduction
- Motivation
- Contributions
- Design Methodology
- Experimental Results
- Conclusions

Asynchronous Progress Methods

- *Hardware-based progression – Not generic*
- *Software-based progression*
  - Host application based (Manual progression)
  - Kernel assisted: Require root privileges
  - Thread/Process based

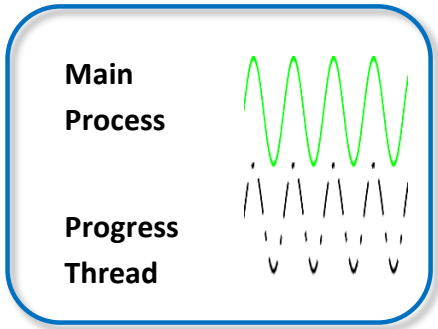
Asynchronous Progress: Host Application based

- MPI_Test() calls inserted between compute operations
- Difficult to identify where MPI_Test() to be inserted
- Require domain knowledge as application code has to be modified

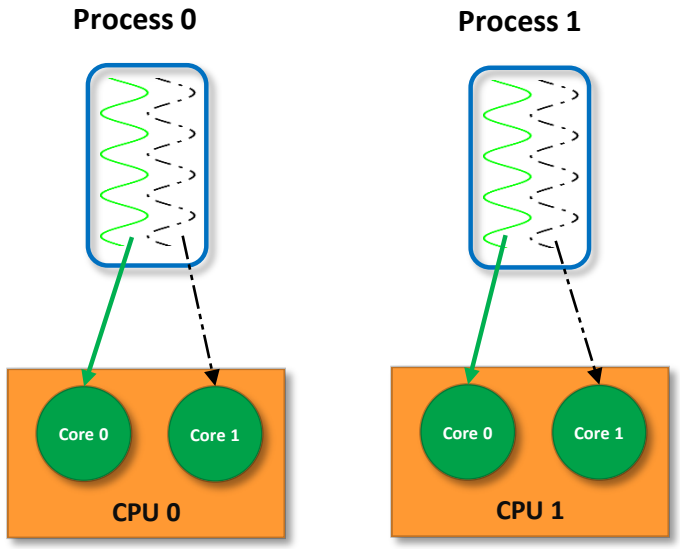


Methods of Asynchronous Progress : Thread/Process based

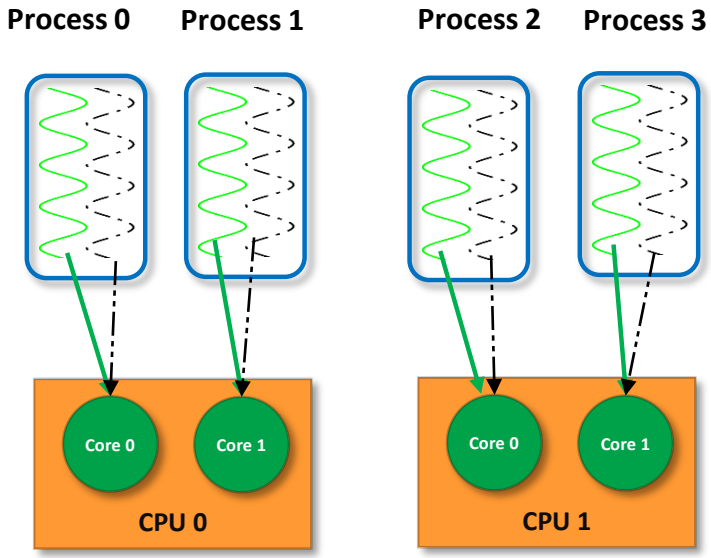
- Progress threads are created for non-blocking message communication
- Two approaches
  - Individual progress thread for each user process - 1:1
    - Partially Subscribed
    - Fully subscribed
  - Separate progress processes for a group of user processes - 1:N



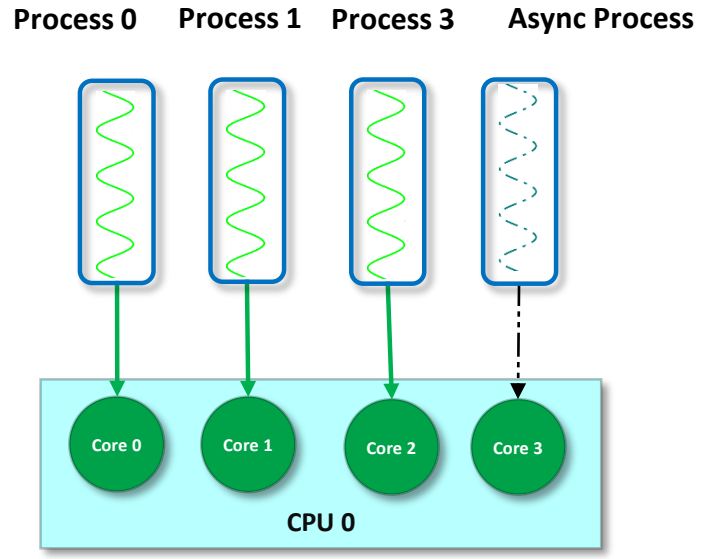
Partial Subscription



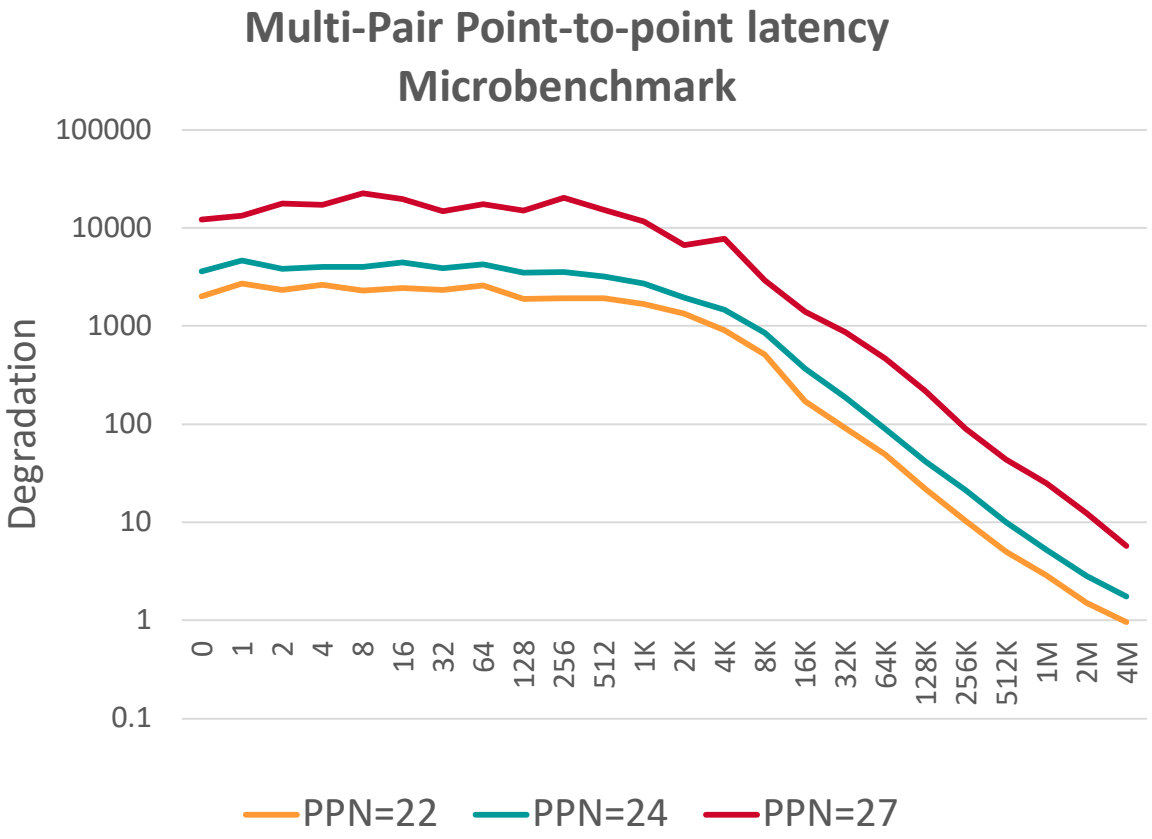
Full Subscription



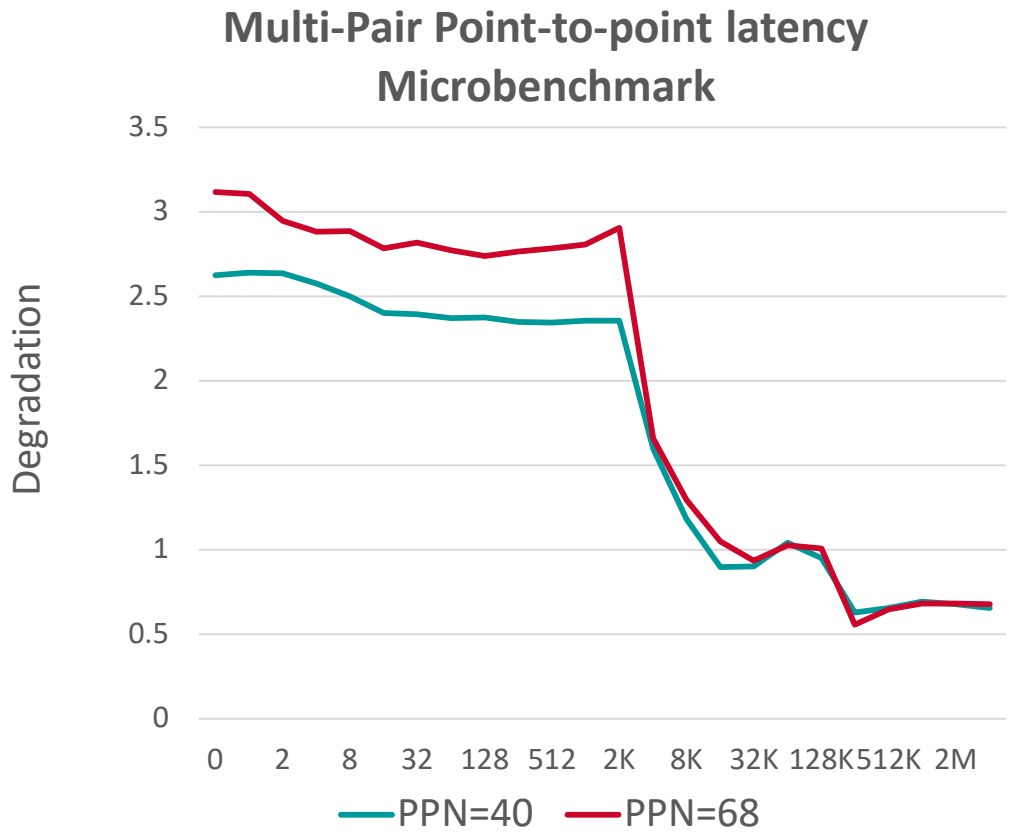
Separate progress processes



Impact of Thread-based Progress on Performance



Nodes=2; Max PPN=28; Hyperthreading=No; Arch=Broadwell



Nodes=2; Max PPN=68; Hyperthreading=Yes; Arch=KNL

Observation : Latency numbers grows more rapidly with increase in process per node (PPN)

P2P Communication

Eager

- Asynchronous protocol that allows to send data immediately irrespective of receiver state
- Send operation completes without acknowledgement from a matching receive
- Best communication performance for smaller messages

Rendezvous

- Synchronous protocol which requires an acknowledgement from a matching receive for the send operation to complete
- Best communication performance for larger messages

But what about overlap?

CHALLENGES

1. How can MPI library identify scenarios when asynchronous progress is required?

2. How can we minimize the CPU utilization of the asynchronous progress threads and maximize CPU availability for application's compute?

3. How can we reduce the number of context-switches and preemption between the main thread and asynchronous progress thread?

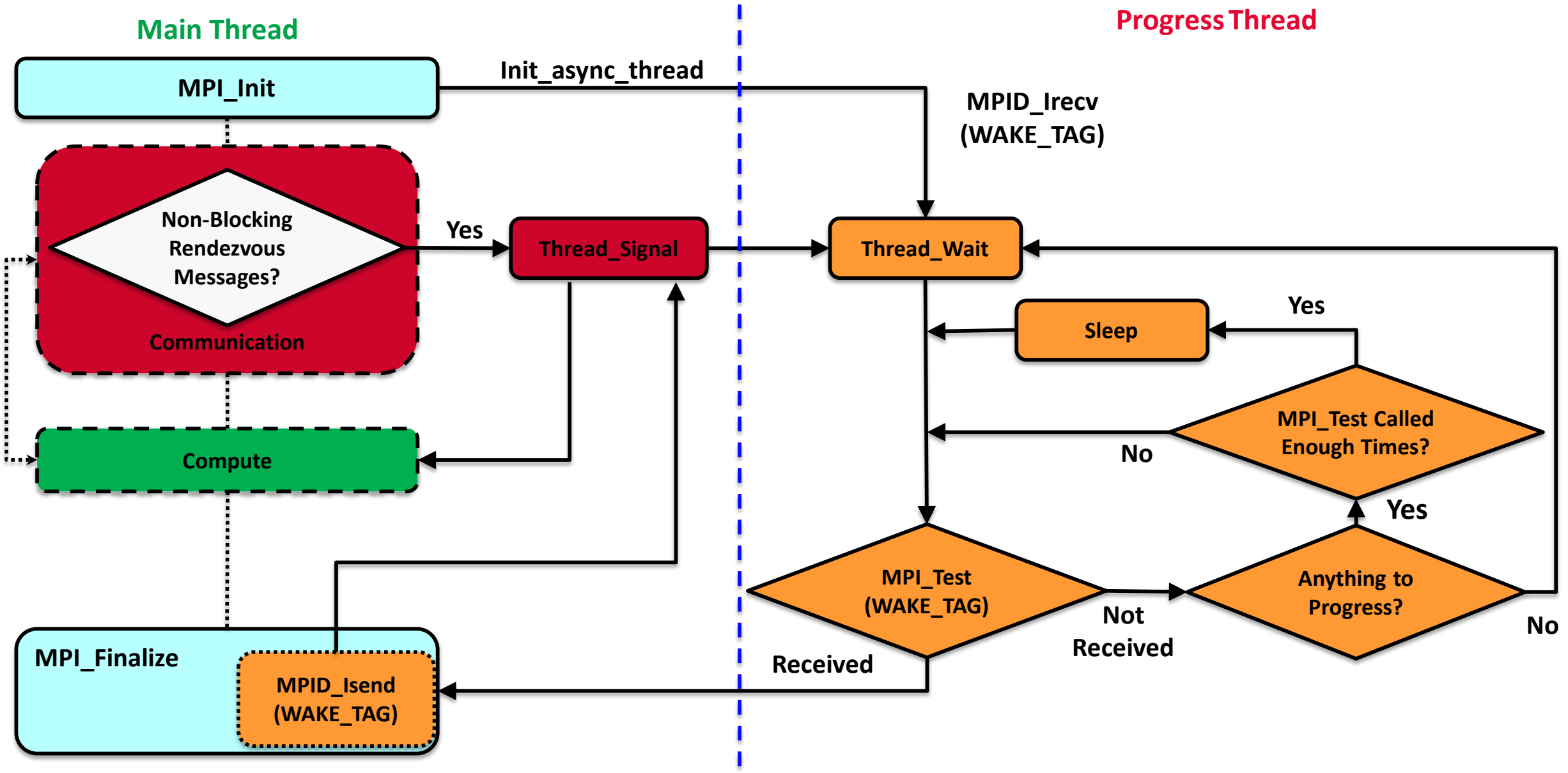
4. Can we avoid using specialized hardware or software resources?

CONTRIBUTIONS

Proposed a thread-based asynchronous progress design that

- does not require additional cores or offload hardware
- does not necessitate administrative privileges at remote cluster nodes
- does not require change in application code
- ensure fair usage of system resources among the main and progress threads

PROPOSED DESIGN



EXPERIMENTAL SETUP

Cluster	Processor	Memory	Interconnect
Broadwell + InfiniBand (No Hyperthreading)	2.4 GHz 14-core Xeon E5-2680v4 per socket, 2 sockets, 1 thread/core	512 GB RAM + 400GB PCIe SSD.	IB-EDR (100Gbps)
KNL+ Omni-Path (Hyperthreaded)	1.4 GHz 68-core Intel Xeon Phi 7250 per socket 1 socket, 4 hardware threads/core.	96GB DDR4 RAM + 16 GB MCDRAM	Omni-Path (100Gbps)
Skylake + Omni-Path (Hyperthreaded)	2.1 GHz 24-core Intel Xeon Platinum 8160 per socket, 2 sockets, 2 hardware threads/core.	192GB DDR4 RAM	Omni-Path (100Gbps)
Skylake + InfiniBand (Hyperthreaded)	2.4 GHz 20-core Intel Xeon Gold 6148 per socket, 2 threads/core	384GB DDR3 RAM	IB-EDR (100G)
OpenPOWER + InfiniBand (No Hyperthreading)	3.4 GHz 10-core Power-8 CPUs per socket 2 sockets, 8 threads per core	256GB DDR3 RAM	IB-EDR (100G)

EXPERIMENTAL SETUP

Mpi Library	Version	Configurations
MVAPICH2	MVAPICH2-X 2.3b	MPICH_ASYNC_PROGRESS=1 MV2_THREADS_PER_PROCESS=2 (MPICH Async)
MVAPICH2	MVAPICH2-X 2.3b	MV2_ASYNC_PROGRESS=1 MV2_OPTIMIZED_ASYNC_PROGRESS=1 (Proposed design)
Intel MPI	2018.1.163	I_MPI_ASYNC_PROGRESS=1
OpenMPI	3.0.1	Default (No support for async progress)

1. **Amit Ruhela**, Hari Subramoni, Sourav Chakraborty, Mohammadreza Bayatpour, Pooya Kousha, and D.K. Panda, "*Efficient Asynchronous Progress without Dedicated Resources*", **Parallel Computing 2019**

2. **Amit Ruhela**, Hari Subramoni, Sourav Chakraborty, Mohammadreza Bayatpour, Pooya Kousha, and D.K. Panda, "*Efficient Asynchronous Communication Progress for MPI without Dedicated Resources*", **EuroMPI 2018**

Overview of the MVAPICH2 Project

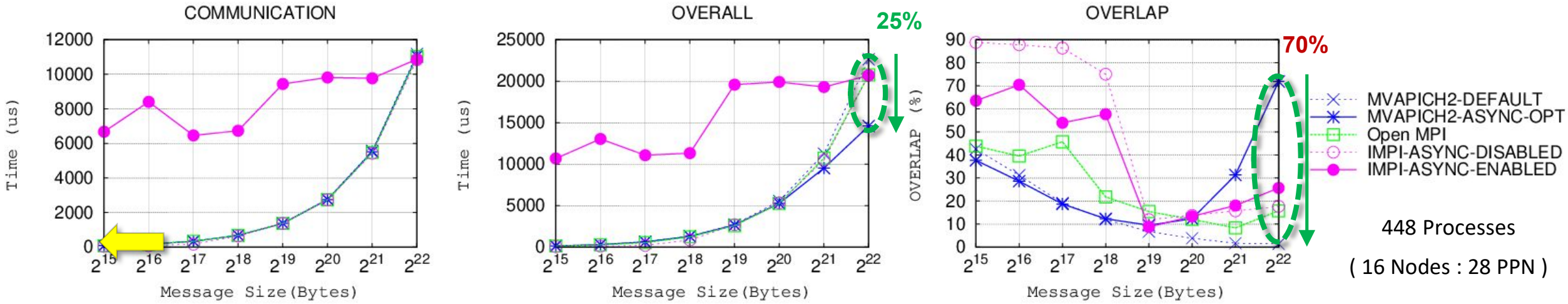
- High Performance open-source MPI Library for InfiniBand, Omni-Path, Ethernet/iWARP, and RDMA over Converged Ethernet (RoCE)
  - MVAPICH (MPI-1), MVAPICH2 (MPI-2.2 and MPI-3.1), Started in 2001, First version available in 2002
  - MVAPICH2-X (MPI + PGAS), Available since 2011
  - Support for GPGPUs (MVAPICH2-GDR) and MIC (MVAPICH2-MIC), Available since 2014
  - Support for Virtualization (MVAPICH2-Virt), Available since 2015
  - Support for Energy-Awareness (MVAPICH2-EA), Available since 2015
  - Support for InfiniBand Network Analysis and Monitoring (OSU INAM) since 2015
  - **Used by more than 3,050 organizations in 89 countries**
  - **More than 614,000 (> 0.6 million) downloads from the OSU site directly**
  - Empowering many TOP500 clusters (Nov '18 ranking)
    - 3rd, 10,649,600-core (Sunway TaihuLight) at National Supercomputing Center in Wuxi, China
    - 5th, 448, 448 cores (Frontera) at TACC
    - 8th, 391,680 cores (ABCI) in Japan
    - 15th, 570,020 cores (Neurion) in South Korea and many others
  - Available with software stacks of many vendors and Linux Distros (RedHat, SuSE, and OpenHPC)
  - **http://mvapich.cse.ohio-state.edu**

Partner in the TACC Frontera System

- Empowering Top500 systems for over a decade

IMPACT : MULTI-PAIR POINT-TO-POINT LATENCY

Broadwell + InfiniBand

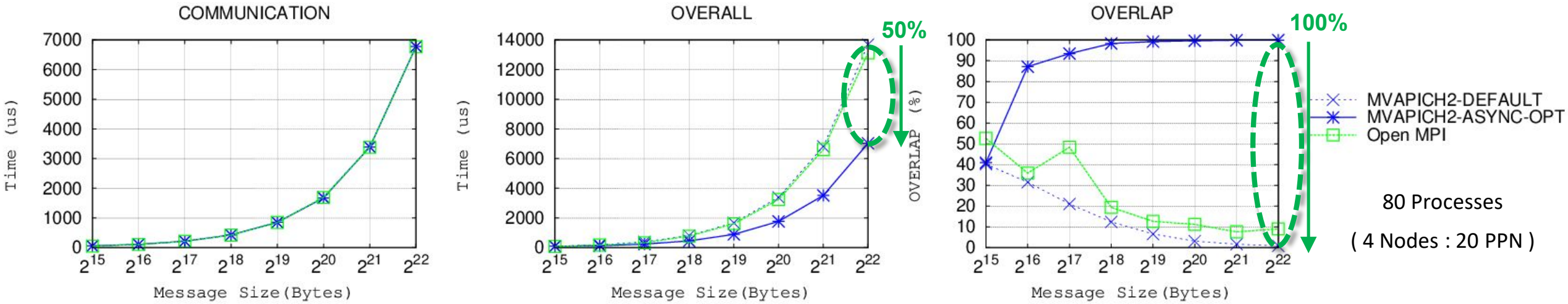


Observations :

1. Communication time is similar for all MPI libraries
2. Overlap percentage is highest with Optimized Asynchronous progress design beyond eager threshold
3. Shows 25% reduction in overall time for the latency benchmark

IMPACT : MULTI-PAIR POINT-TO-POINT LATENCY

OpenPOWER + InfiniBand

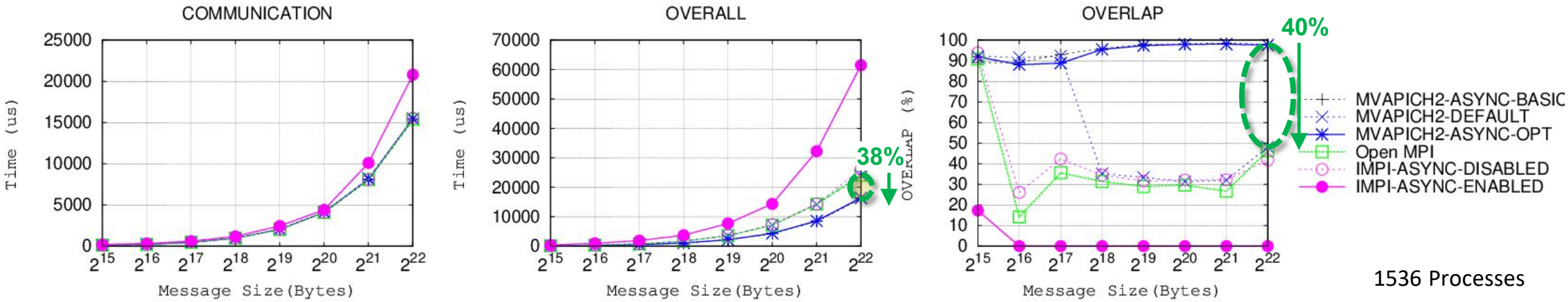


Observations :

1. Near 100% overlap between computation and communication
2. 50% reduction in latency numbers by optimized async design

IMPACT : MULTI-PAIR POINT-TO-POINT LATENCY

Skylake + Omni-Path



1536 Processes

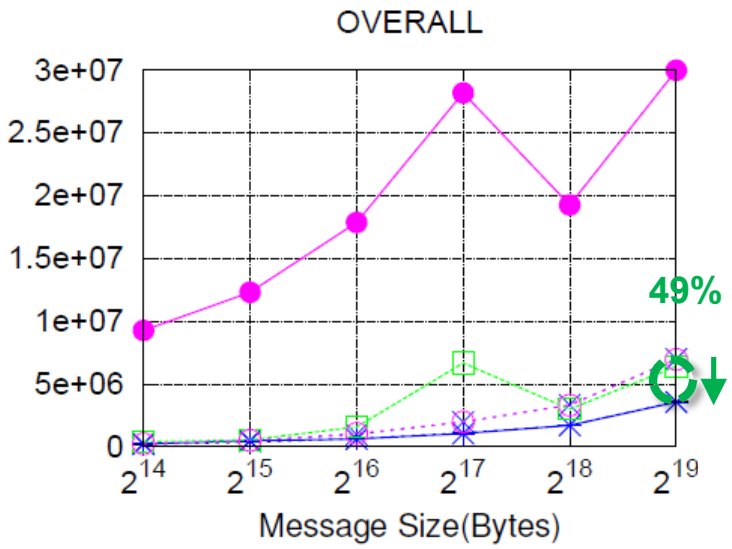
(32 Nodes : 48 PPN)

Observations :

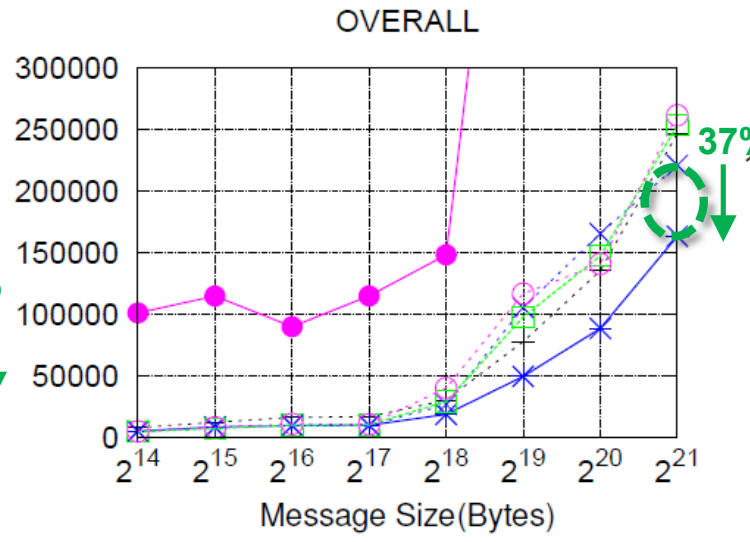
1. Up to 38% reduction in latency numbers by optimized async progress design on Skylake + Omni-Path architecture
2. The trend in performance numbers on **KNL + Omni-Path** architecture follows similar trend
   - Reduction in latency numbers by optimized async progress design by up to 34% with 4352 processes

IMPACT : Igatherv, Iscatterv, and Ialltoallv

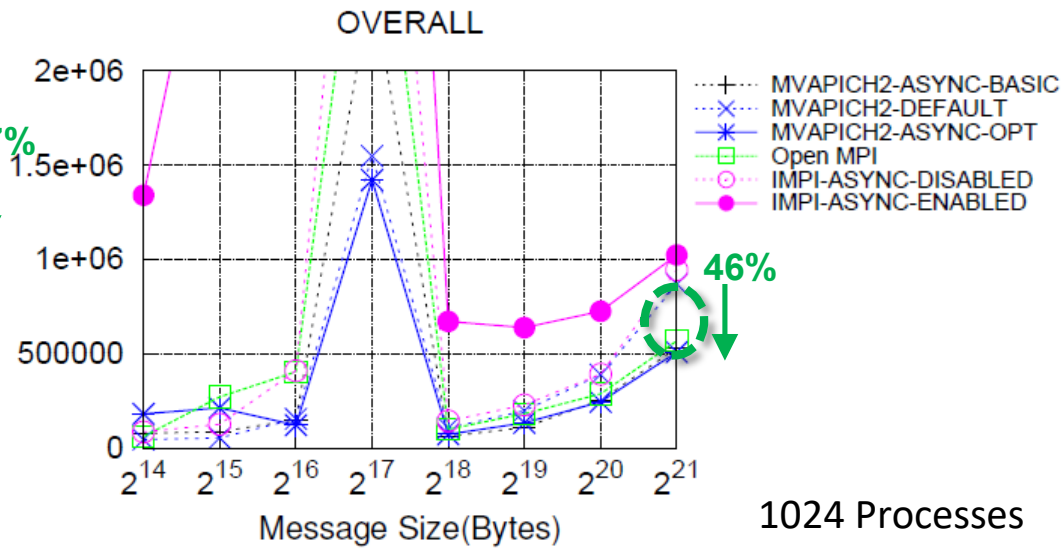
KNL + Omni-Path



MPI_Ialltoallv



MPI_Iscatterv



MPI_Igatherv

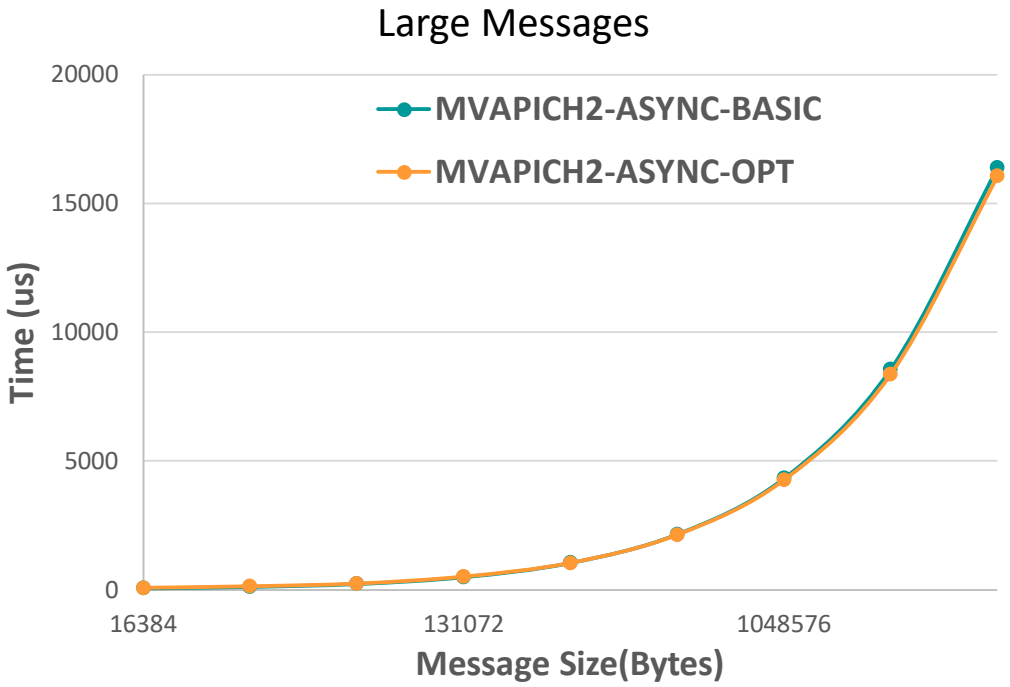
1024 Processes

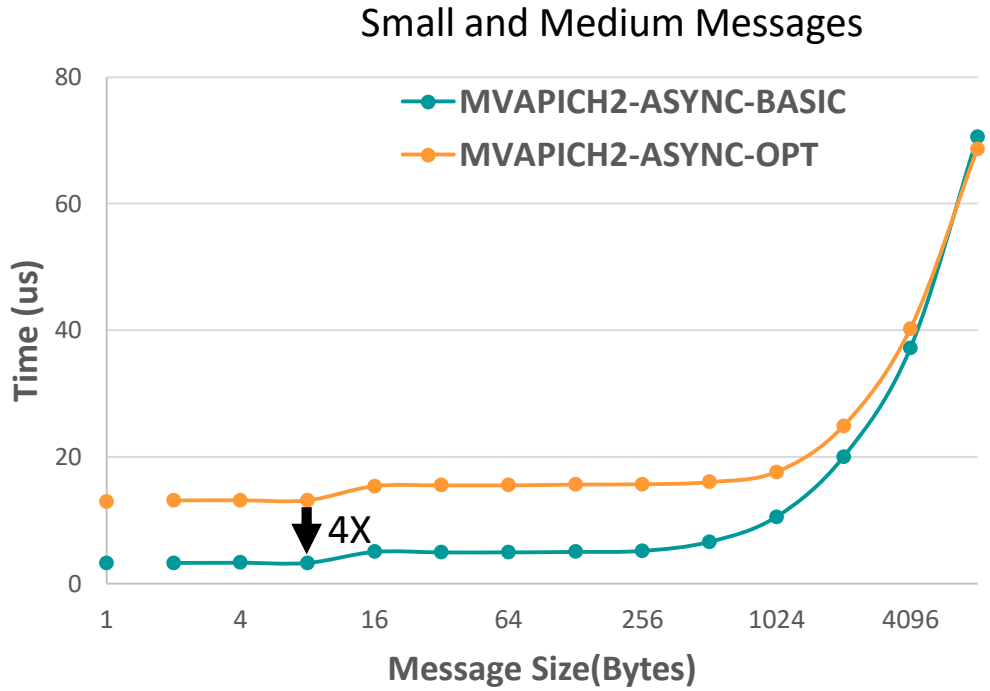
(16 Nodes : 64 PPN)

Observations :

1. Consistent trend of optimized async progress design outperforming on all four hardware platforms.
   - Up to 49% reduction in numbers for MPI_Ialltoallv
   - Up to 37% reduction in numbers for MPI_Iscatterv
   - Up to 46% reduction in latency numbers for MPI_Igatherv
2. 15-20% less overheads than default MPICH async design at small messages

Experimental Results : Does Hyperthreading help?





Skylake + Omni-Path

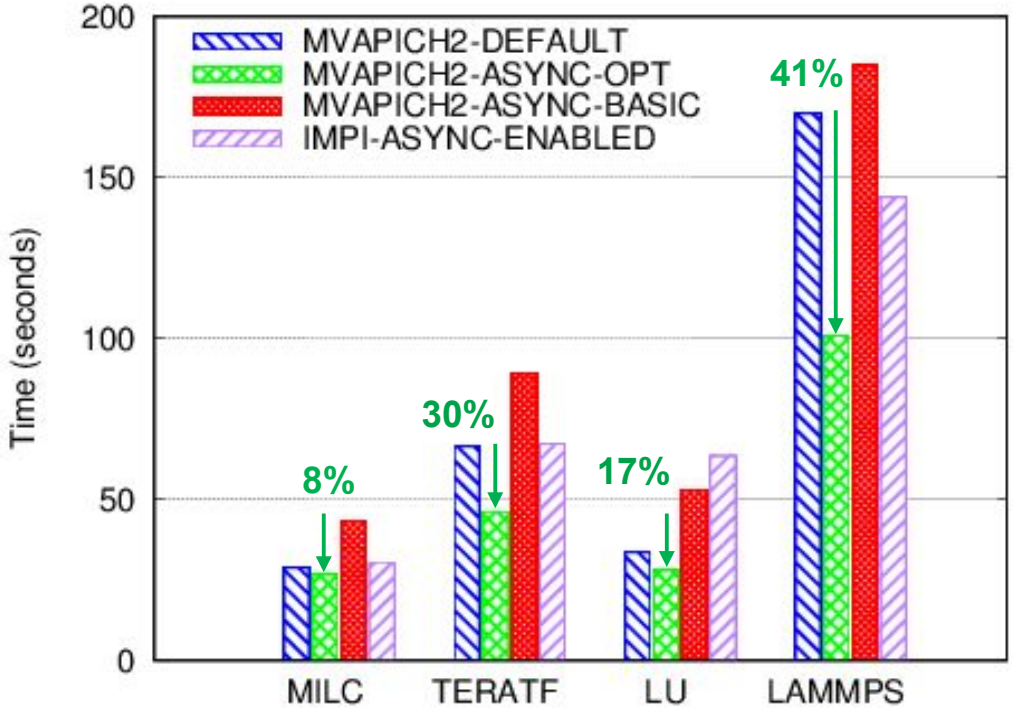
Observations :

1. ***Default MPICH design runs on Skylake and KNL nodes at full subscription because of supported hyper-threading***
2. ***Performance of default MPICH design similar in performance to optimized async design at large messages but incurs up to 4 X overheads for small and medium messages***

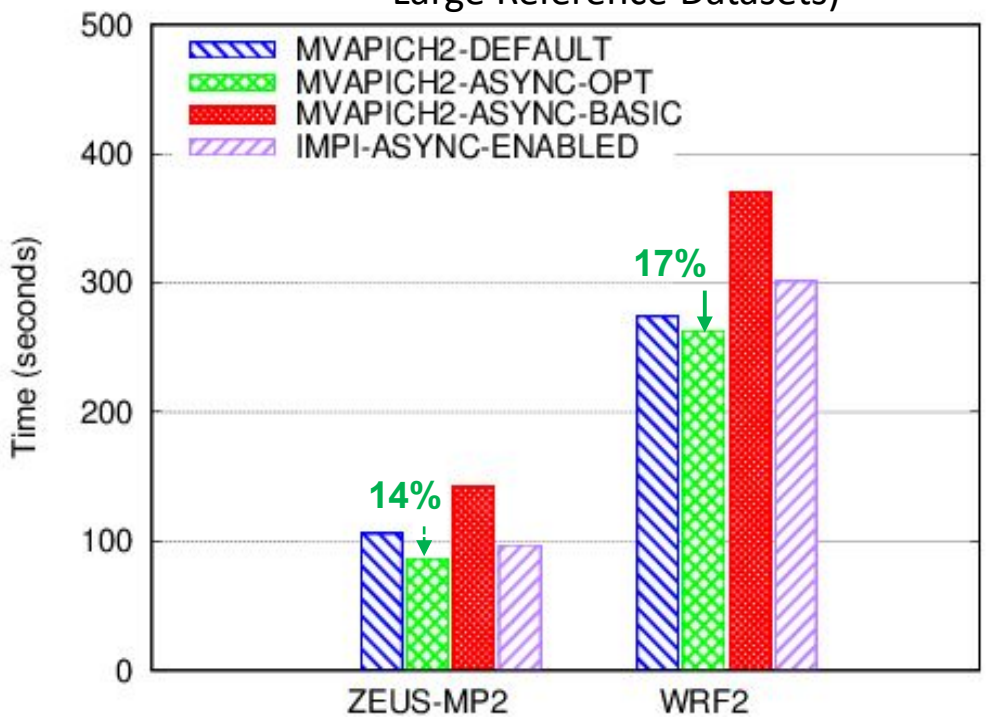
IMPACT ON APPLICATIONS: SPEC MPI

Skylake + Omni-Path

Medium Reference Datasets



Large Reference Datasets)



576 Processes (12 Nodes : 48 PPN)

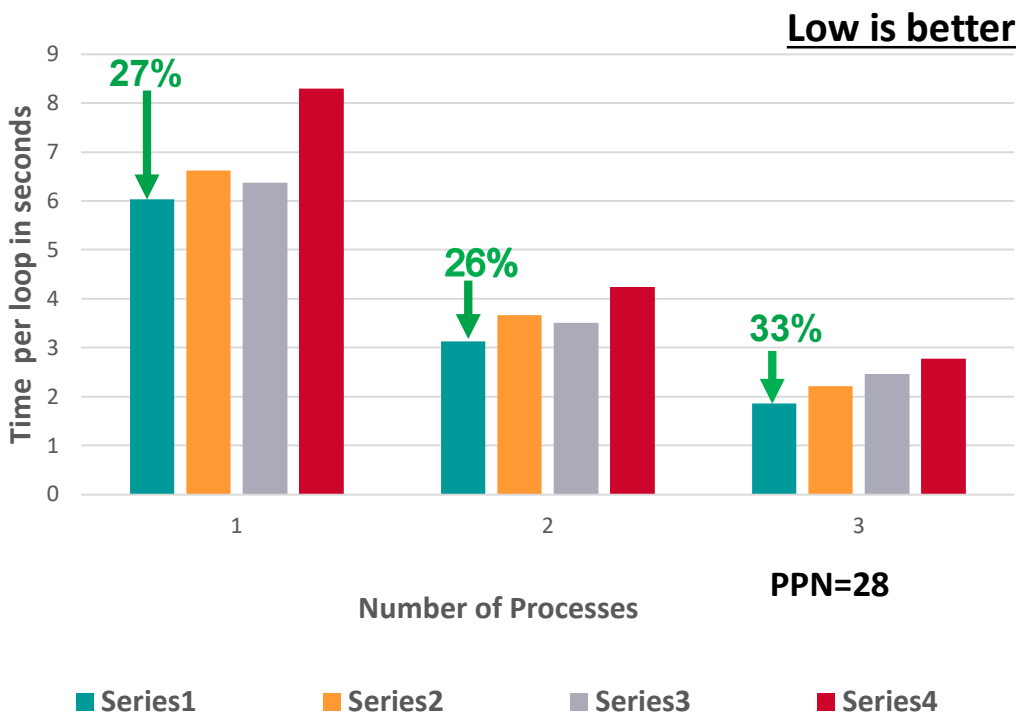
Observations :

1. Up to 41% improved performance for SPECMPI applications with Skylake + Omni-Path
2. Up to 18% improved performance for SPECMPI applications with KNL + Omni-Path *

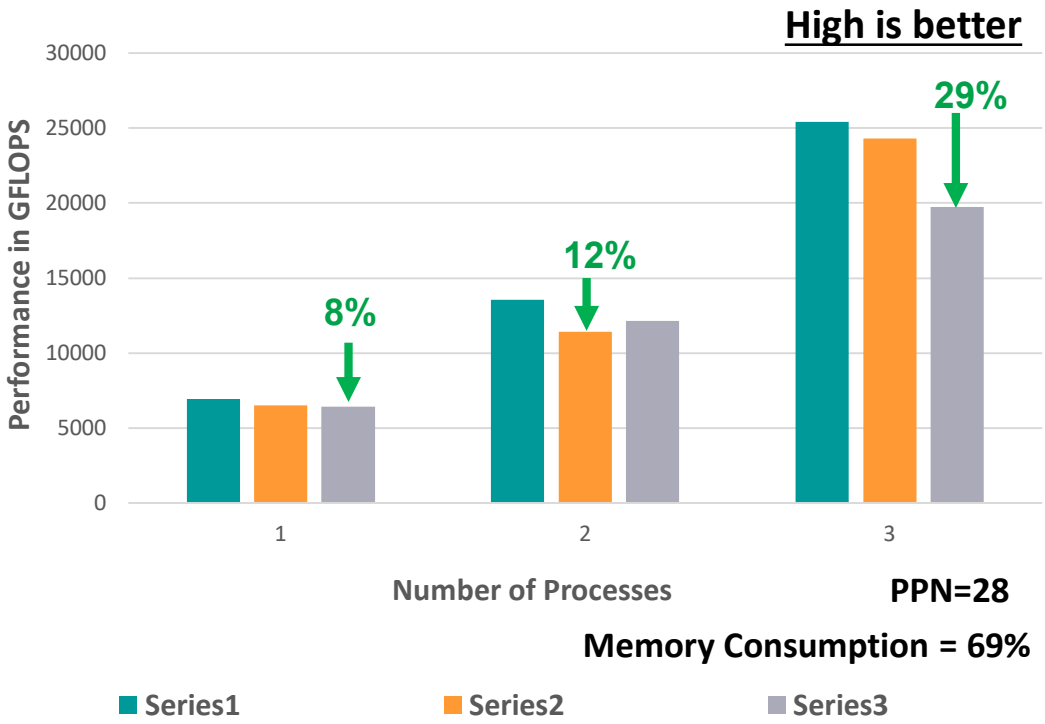
IMPACT ON APPLICATIONS: P3DFFT AND HPL

Broadwell + InfiniBand

P3DFFT



High Performance Linpack (HPL)



Observations :

1. Up to 33% performance improvement in P3DFFT application with 448 processes
2. Up to 29% performance improvement in HPL application with 896 processes

CONCLUSIONS

- Proposed scalable asynchronous progress design that requires
  - No additional software or hardware resources
  - No change in host application code
  - No require administrative privileges

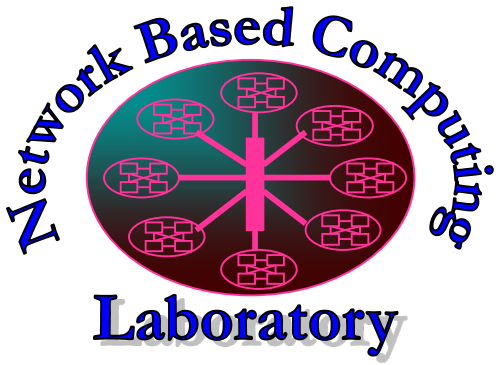
- Improved performance of benchmarks and application by up to 50%

- The async design is available in MVAPICH2-X library since v2.3rc1

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

Thank You!

{ruhela.2, subramoni.1, chakraborty.52 , bayatpour.1, kousha.2, panda.2} @ osu.edu



Network-Based Computing Laboratory
http://nowlab.cse.ohio-state.edu/

MVAPICH Web Page
http://mvapich.cse.ohio-state.edu/