MVAPICH
MPI, PGAS and Hybrid MPI+PGAS Library

How to Boost the Performance of Your MPI and PGAS Applications with MVAPICH2 Libraries

A Tutorial at

MVAPICH User Group 2017

by

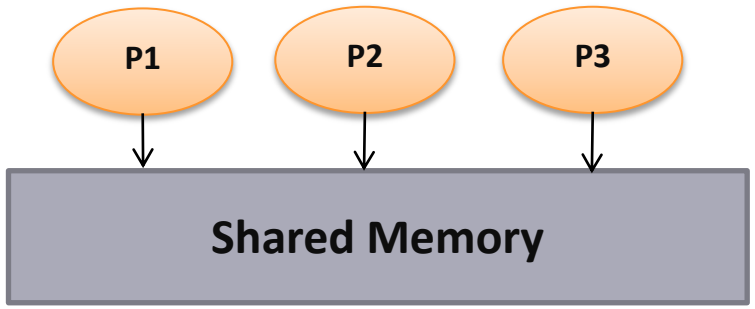
The MVAPICH Team

The Ohio State University

E-mail: panda@cse.ohio-state.edu

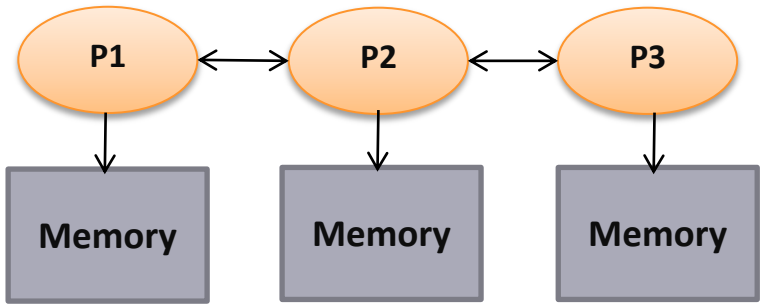
http://www.cse.ohio-state.edu/~panda

Parallel Programming Models Overview



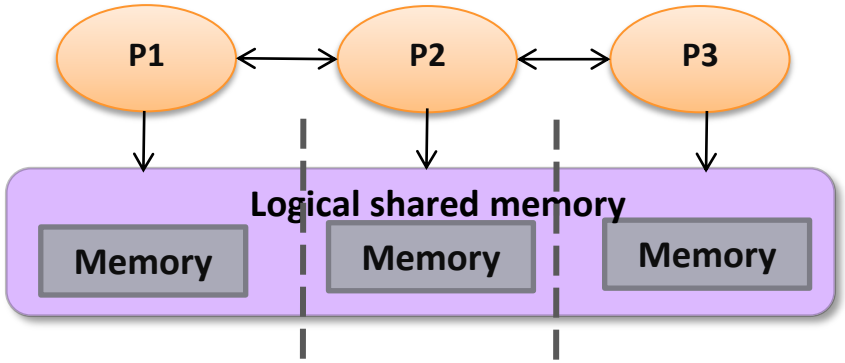
Shared Memory Model

SHMEM, DSM



Distributed Memory Model

MPI (Message Passing Interface)



Partitioned Global Address Space (PGAS)

Global Arrays, UPC, Chapel, X10, 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.
- PGAS models and Hybrid MPI+PGAS models are gradually receiving importance

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

Application Kernels/Applications

Middleware

Programming Models

MPI, PGAS (UPC, Global Arrays, OpenSHMEM), CUDA, OpenMP, OpenACC, Cilk, Hadoop (MapReduce), Spark (RDD, DAG), etc.

Communication Library or Runtime for Programming Models

- Point-to-point Communication
- Collective Communication
- Energy-Awareness
- Synchronization and Locks
- I/O and File Systems
- Fault Tolerance

Networking Technologies
(InfiniBand, 40/100GigE, Aries, and Omni-Path)

Multi-/Many-core Architectures

Accelerators (GPU and MIC)

Co-Design Opportunities and Challenges across Various Layers

Performance

Scalability

Resilience

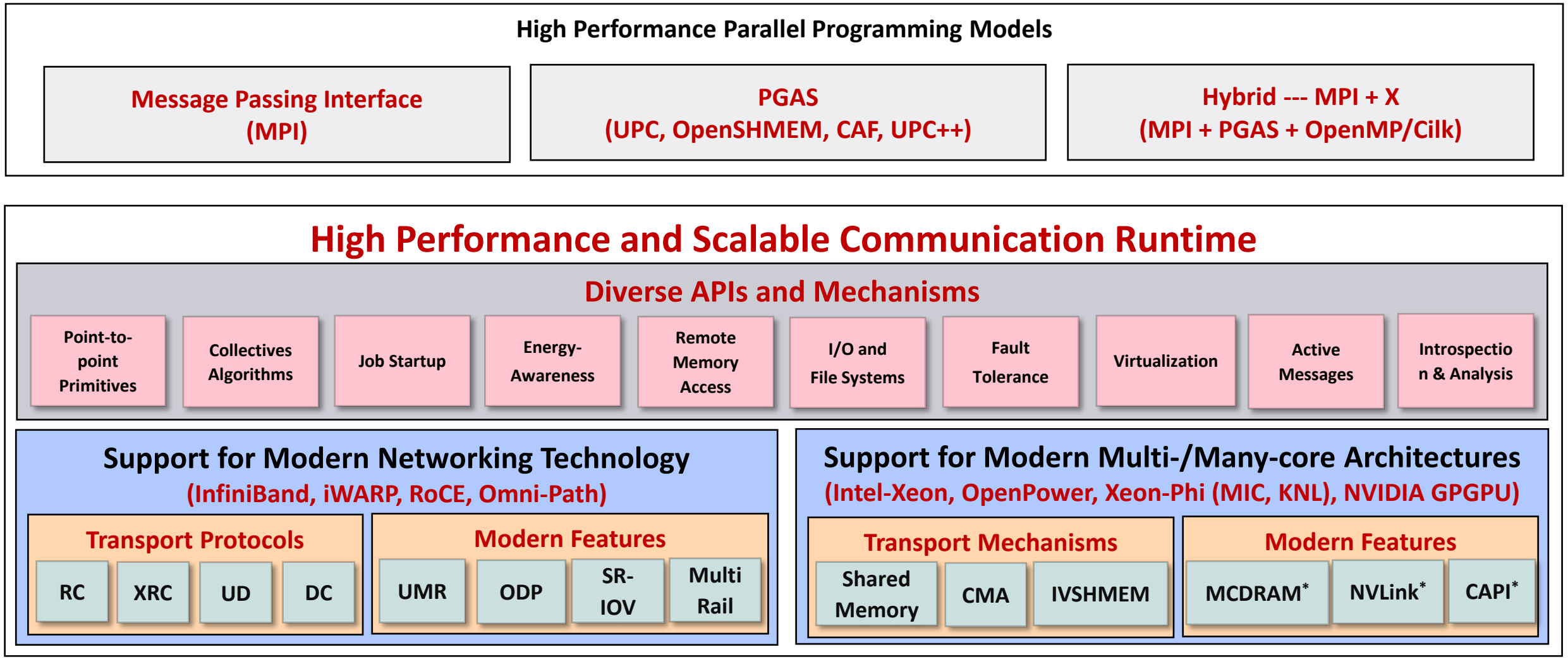
Designing (MPI+X) for Exascale

- Scalability for million to billion processors
  - Support for highly-efficient inter-node and intra-node communication (both two-sided and one-sided)
- Scalable Collective communication
  - Offloaded
  - Non-blocking
  - Topology-aware
- Balancing intra-node and inter-node communication for next generation multi-/many-core (128-1024 cores/node)
  - Multiple end-points per node
- Support for efficient multi-threading
- Integrated Support for GPGPUs and Accelerators
- Fault-tolerance/resiliency
- QoS support for communication and I/O
- Support for Hybrid MPI+PGAS programming
  - MPI + OpenMP, MPI + UPC, MPI + OpenSHMEM, CAF, MPI + UPC++…
- Virtualization
- Energy-Awareness

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-3.0), 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 2,800 organizations in 85 countries**
  - **More than 424,000 (> 0.4 million) downloads from the OSU site directly**
  - Empowering many TOP500 clusters (June '17 ranking)
    - 1st, 10,649,600-core (Sunway TaihuLight) at National Supercomputing Center in Wuxi, China
    - 15th, 241,108-core (Pleiades) at NASA
    - 20th, 462,462-core (Stampede) at TACC
    - 44th, 74,520-core (Tsubame 2.5) at Tokyo Institute of Technology
  - Available with software stacks of many vendors and Linux Distros (RedHat and SuSE)
  - http://mvapich.cse.ohio-state.edu
- Empowering Top500 systems for over a decade
  - System-X from Virginia Tech (3rd in Nov 2003, 2,200 processors, 12.25 TFlops) ->
  - Stampede at TACC (12th in Jun'16, 462,462 cores, 5.168 Plops)

Architecture of MVAPICH2 Software Family



* Upcoming

Strong Procedure for Design, Development and Release

- Research is done for exploring new designs
- Designs are first presented to conference/journal publications
- Best performing designs are incorporated into the codebase
- Rigorous Q&A procedure before making a release
  - Exhaustive unit testing
  - Various test procedures on diverse range of platforms and interconnects
  - Performance tuning
  - Applications-based evaluation
  - Evaluation on large-scale systems
- Even alpha and beta versions go through the above testing

MVAPICH2 Software Family

Requirements	Library
MPI with IB, iWARP and RoCE	MVAPICH2
Advanced MPI, OSU INAM, PGAS and MPI+PGAS with IB and RoCE	MVAPICH2-X
MPI with IB & GPU	MVAPICH2-GDR
MPI with IB & MIC	MVAPICH2-MIC
HPC Cloud with MPI & IB	MVAPICH2-Virt
Energy-aware MPI with IB, iWARP and RoCE	MVAPICH2-EA
MPI Energy Monitoring Tool	OEMT
InfiniBand Network Analysis and Monitoring	OSU INAM
Microbenchmarks for Measuring MPI and PGAS Performance	OMB

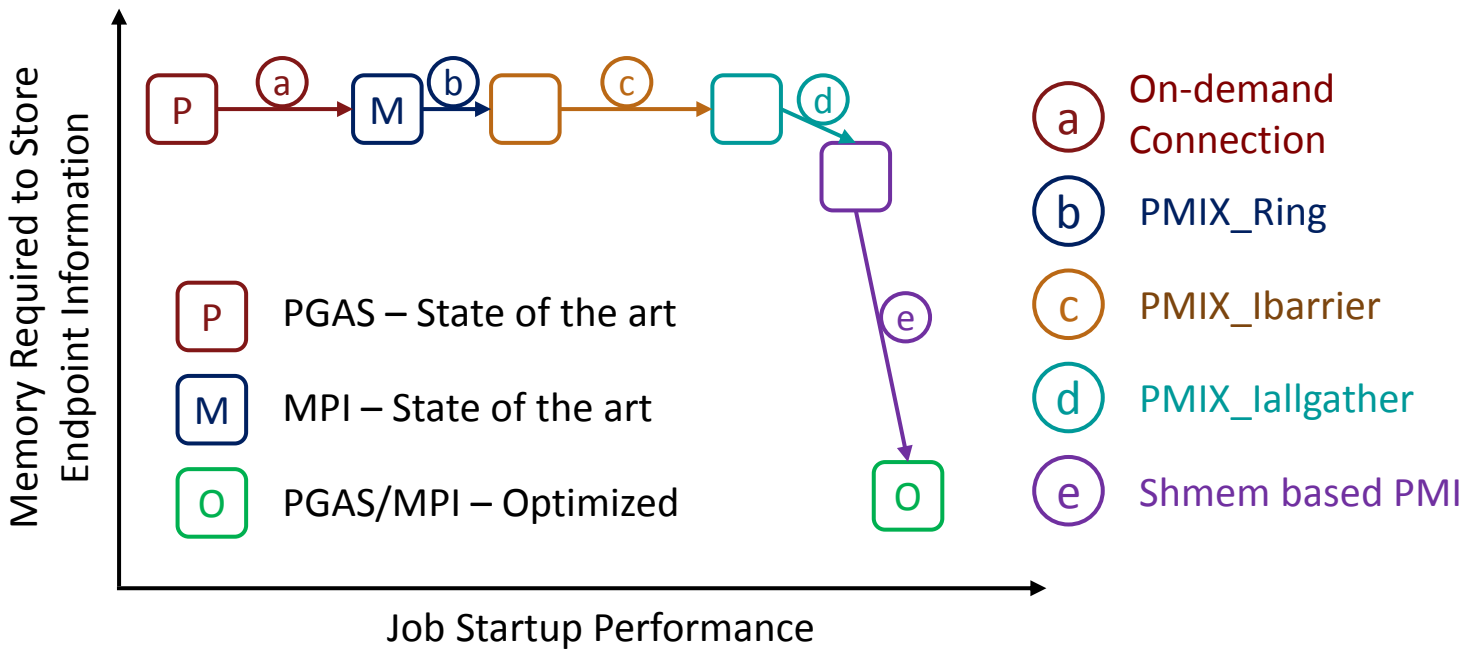
MVAPICH2 2.3b

- Released on 08/10/2017
- Major Features and Enhancements
  - Based on MPICH-3.2
  - Enhance performance of point-to-point operations for CH3-Gen2 (InfiniBand), CH3-PSM, and CH3-PSM2 (Omni-Path) channels
  - Improve performance for MPI-3 RMA operations
  - Introduce support for Cavium ARM (ThunderX) systems
  - Improve support for process to core mapping on many-core systems
    - New environment variable MV2_THREADS_BINDING_POLICY for multi-threaded MPI and MPI+OpenMP applications
    - Support `linear' and `compact' placement of threads
    - Warn user if over-subscription of core is detected
  - Improve launch time for large-scale jobs with mpirun_rsh
  - Add support for non-blocking Allreduce using Mellanox SHARP
  - Efficient support for different Intel Knight's Landing (KNL) models
  - Improve performance for Intra- and Inter-node communication for OpenPOWER architecture
  - Improve support for large processes per node and huge pages on SMP systems
  - Enhance collective tuning for many architectures/systems
  - Enhance support for MPI_T PVARs and CVARs

Presentation Overview

- **Job start-up**
- Point-to-point Inter-node Protocol
- Transport Type Selection
- Multi-rail and QoS
- Process Mapping and Point-to-point Intra-node Protocols
- Collectives
- MPI_T Support

Towards High Performance and Scalable Startup at Exascale



- Near-constant MPI and OpenSHMEM initialization time at any process count
- 10x and 30x improvement in startup time of MPI and OpenSHMEM respectively at 16,384 processes
- Memory consumption reduced for remote endpoint information by O(processes per node)
- 1GB Memory saved per node with 1M processes and 16 processes per node

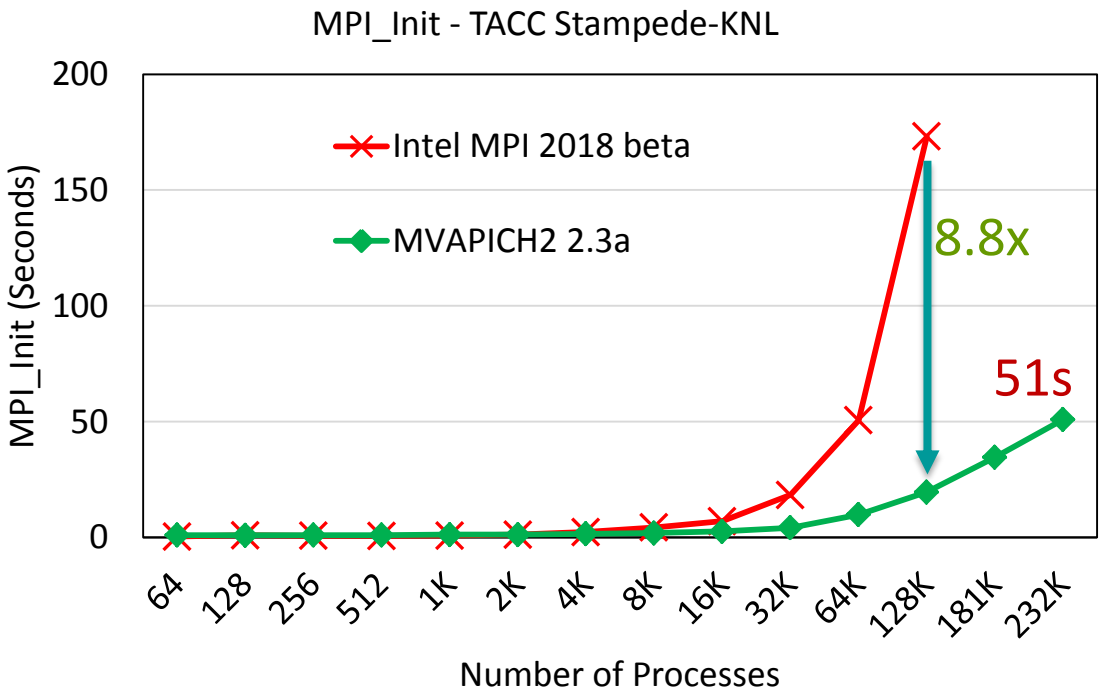
(a) **On-demand Connection Management for OpenSHMEM and OpenSHMEM+MPI.** S. Chakraborty, H. Subramoni, J. Perkins, A. A. Awan, and D K Panda, 20th International Workshop on High-level Parallel Programming Models and Supportive Environments (HIPS '15)

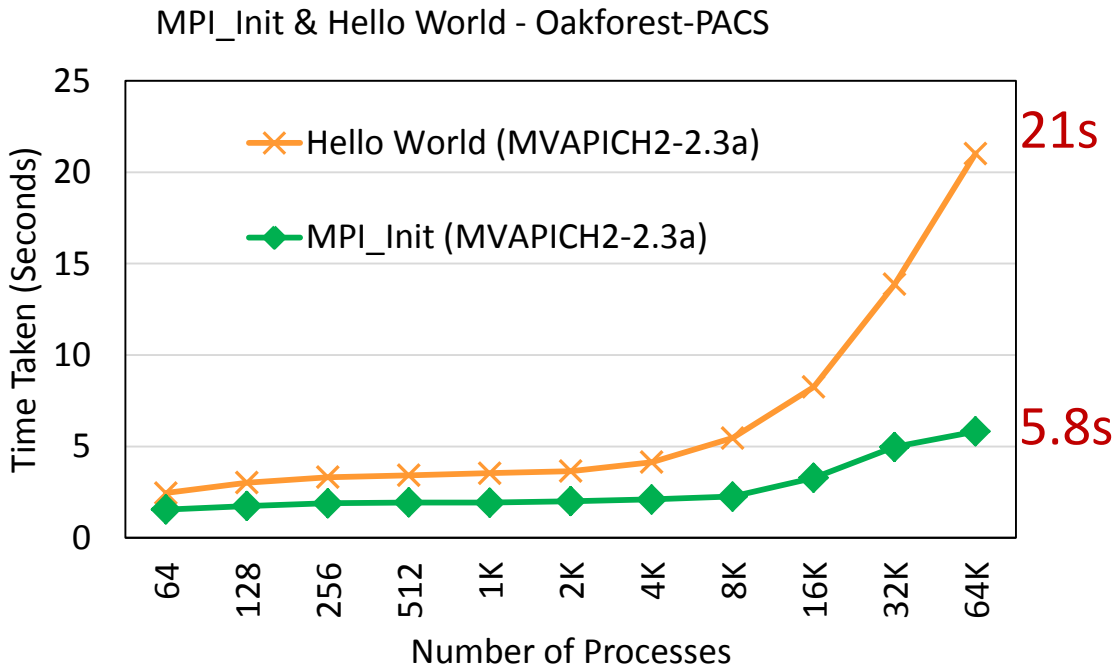
(b) **PMI Extensions for Scalable MPI Startup.** S. Chakraborty, H. Subramoni, A. Moody, J. Perkins, M. Arnold, and D K Panda, Proceedings of the 21st European MPI Users' Group Meeting (EuroMPI/Asia '14)

(c) (d) **Non-blocking PMI Extensions for Fast MPI Startup.** S. Chakraborty, H. Subramoni, A. Moody, A. Venkatesh, J. Perkins, and D K Panda, 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid '15)

(e) **SHMEMPMI – Shared Memory based PMI for Improved Performance and Scalability.** S. Chakraborty, H. Subramoni, J. Perkins, and D K Panda, 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid '16)

Startup Performance on KNL + Omni-Path



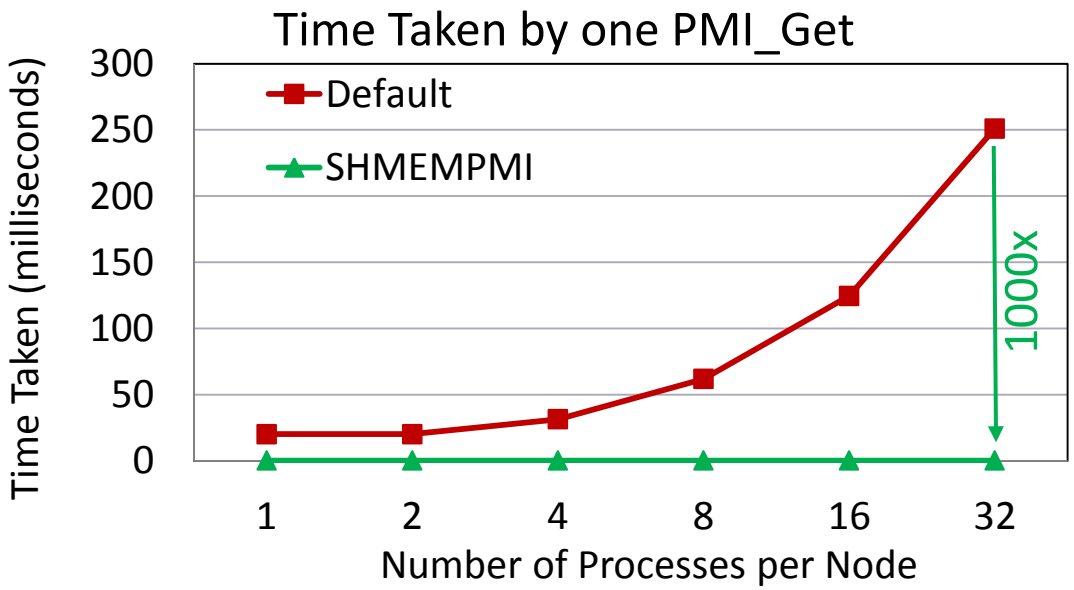


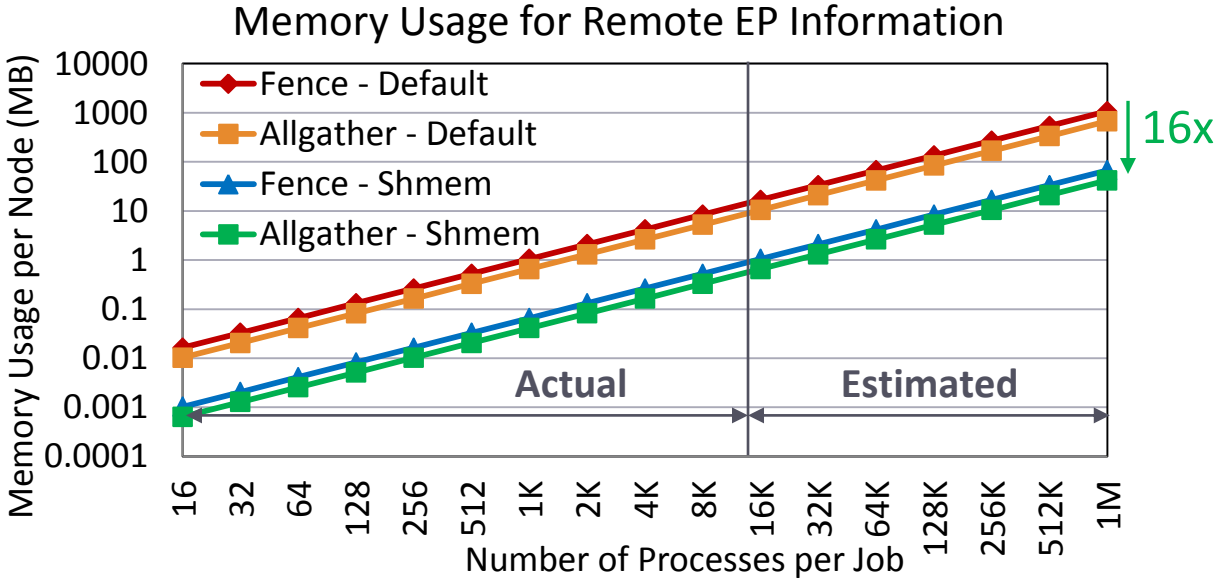
- MPI_Init takes 51 seconds on 231,956 processes on 3,624 KNL nodes (Stampede – Full scale)
- 8.8 times faster than Intel MPI at 128K processes (Courtesy: TACC)
- At 64K processes, MPI_Init and Hello World takes 5.8s and 21s respectively (Oakforest-PACS)
- All numbers reported with 64 processes per node

New designs available in MVAPICH2-2.3a and as patch for SLURM-15.08.8 and SLURM-16.05.1

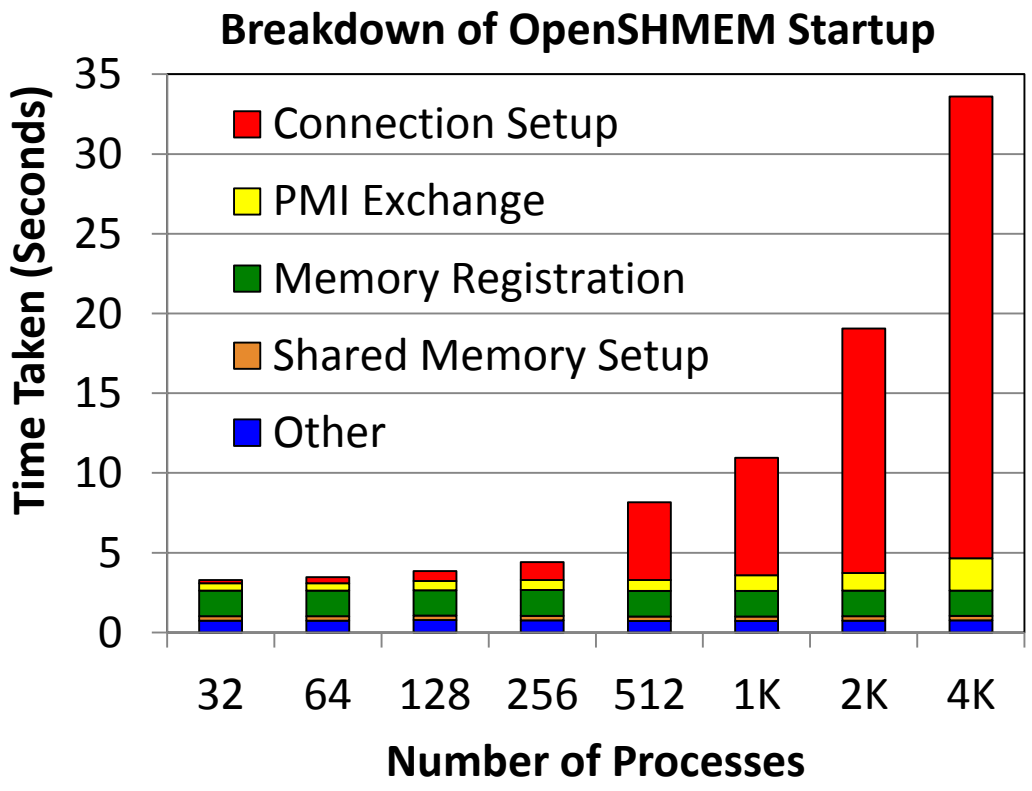
Process Management Interface (PMI) over Shared Memory (SHMEMPMI)

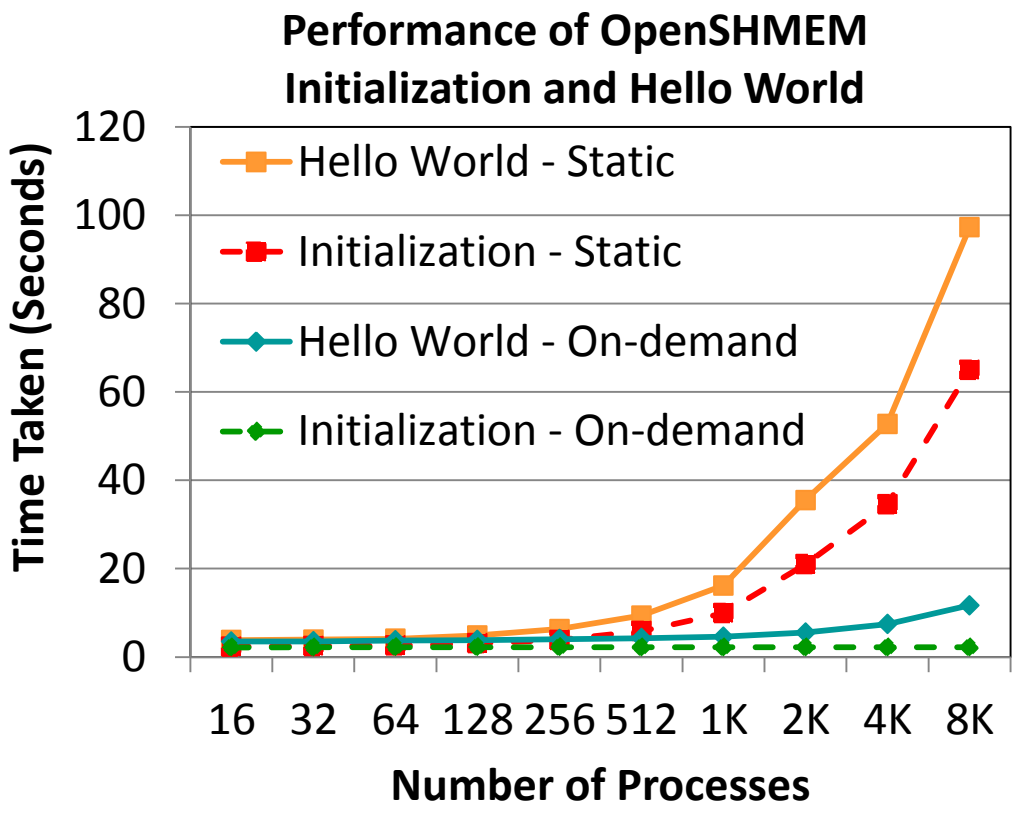
- SHMEMPMI allows MPI processes to directly read remote endpoint (EP) information from the process manager through shared memory segments
- Only a single copy per node - O(processes per node) reduction in memory usage
- Estimated savings of 1GB per node with 1 million processes and 16 processes per node
- Up to 1,000 times faster PMI Gets compared to default design
- **Available for MVAPICH2 2.2rc1 and SLURM-15.08.8**





On-demand Connection Management for OpenSHMEM+MPI





- Static connection establishment wastes memory and takes a lot of time
- On-demand connection management improves OpenSHMEM initialization time by 29.6 times
- Time taken for Hello World reduced by 8.31 times at 8,192 processes
- **Available since MVAPICH2-X 2.1rc1**

How to Get the Best Startup Performance with MVAPICH2?

- **MV2_HOMOGENEOUS_CLUSTER=1** //Set for homogenous clusters
- **MV2_ON_DEMAND_UD_INFO_EXCHANGE=1** //Enable UD based address exchange

Using SLURM as launcher

- **Use PMI2**
  - ./configure --with-pm=slurm --with-pmi=pmi2
  - srun --mpi=pmi2 ./a.out
- **Use PMI Extensions**
  - Patch for SLURM available at http://mvapich.cse.ohio-state.edu/download/
  - Patches available for SLURM 15, 16, and 17
  - PMI Extensions are automatically detected by MVAPICH2

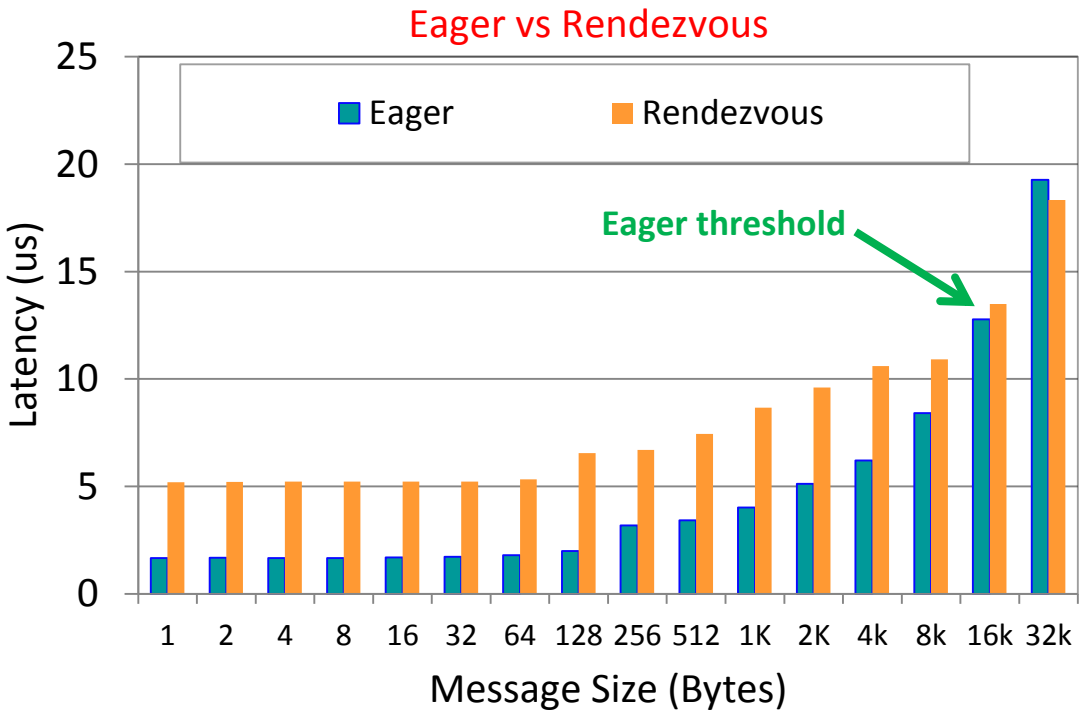
Using mpirun_rsh as launcher

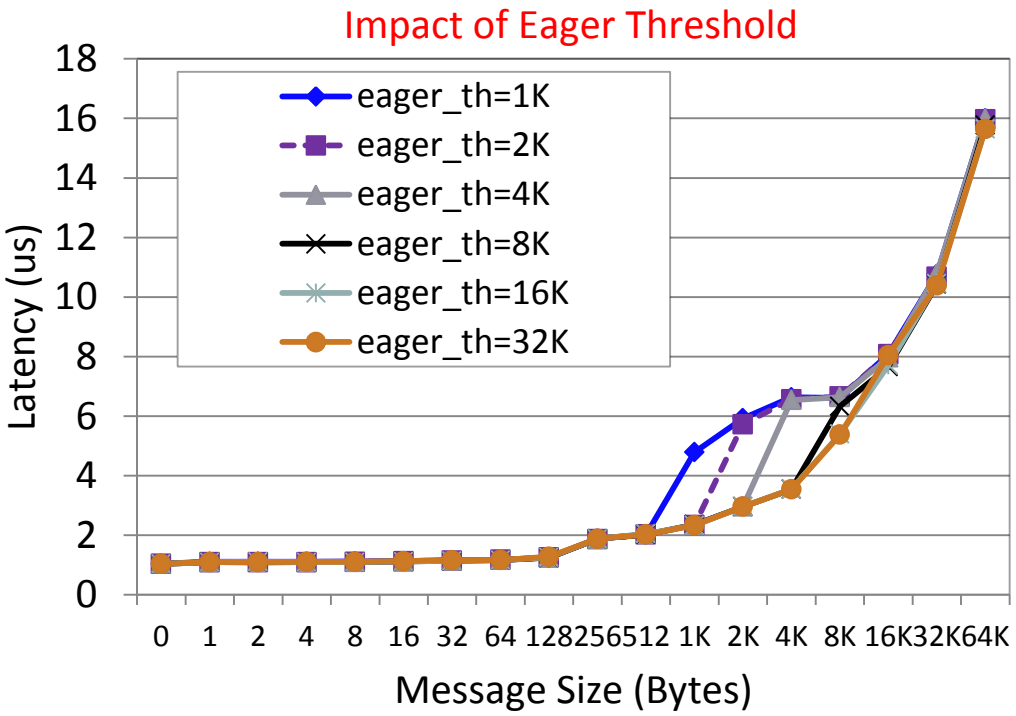
- **MV2_MT_DEGREE**
  - degree of the hierarchical tree used by mpirun_rsh
- **MV2_FASTSSH_THRESHOLD**
  - #nodes beyond which hierarchical-ssh scheme is used
- **MV2_NPROCS_THRESHOLD**
  - #nodes beyond which file-based communication is used for hierarchical-ssh during start up

Presentation Overview

- Job start-up
- **Point-to-point Inter-node Protocol**
- Transport Type Selection
- Multi-rail and QoS
- Process Mapping and Point-to-point Intra-node Protocols
- Collectives
- MPI_T Support

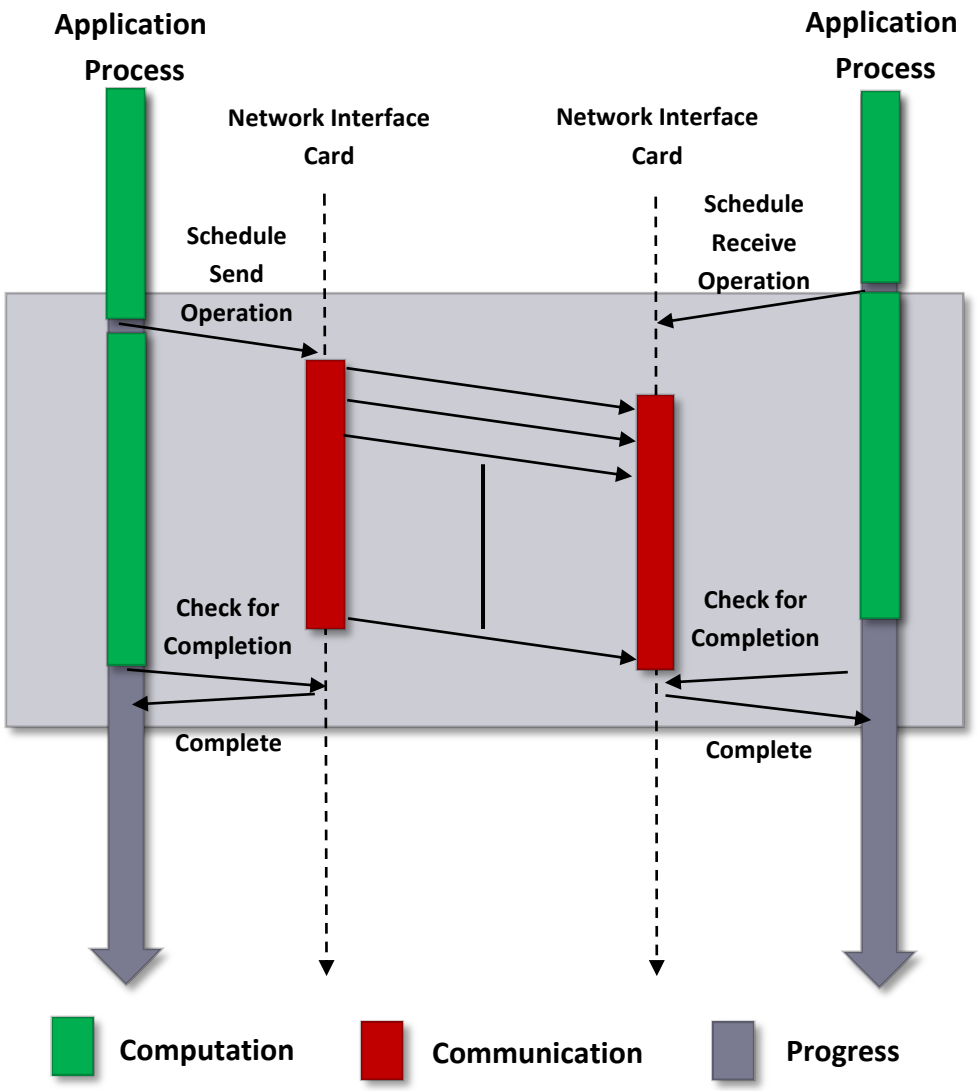
Inter-node Point-to-Point Tuning: Eager Thresholds



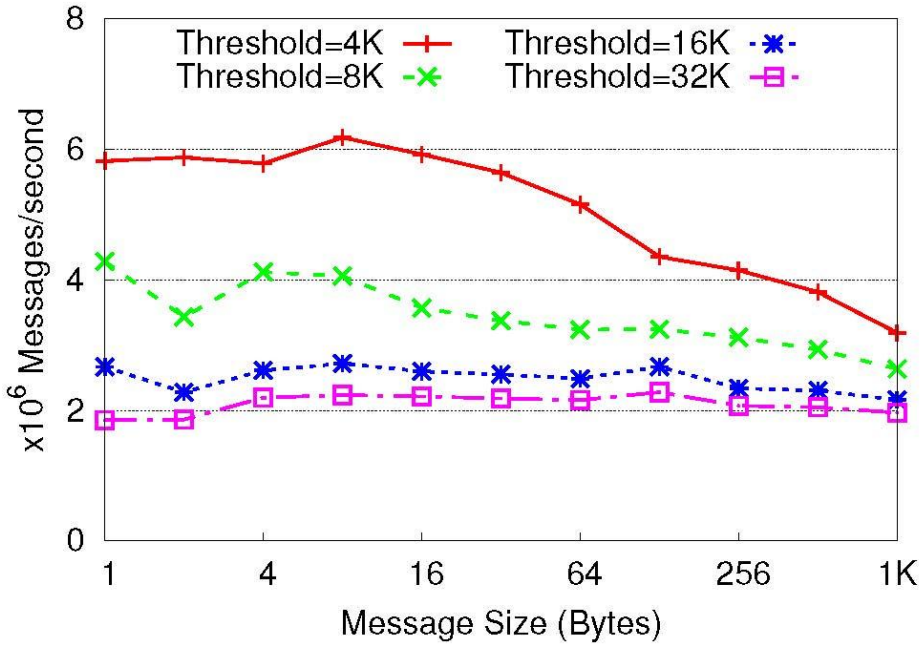


- Switching Eager to Rendezvous transfer
  - Default: Architecture dependent on common platforms, in order to achieve both best performance and memory footprint
- Threshold can be modified by users to get smooth performance across message sizes
  - mpirun_rsh –np 2 –hostfile hostfile MV2_IBA_EAGER_THRESHOLD=32K a.out
  - Memory footprint can increase along with eager threshold

Analyzing Overlap Potential of Eager Protocol

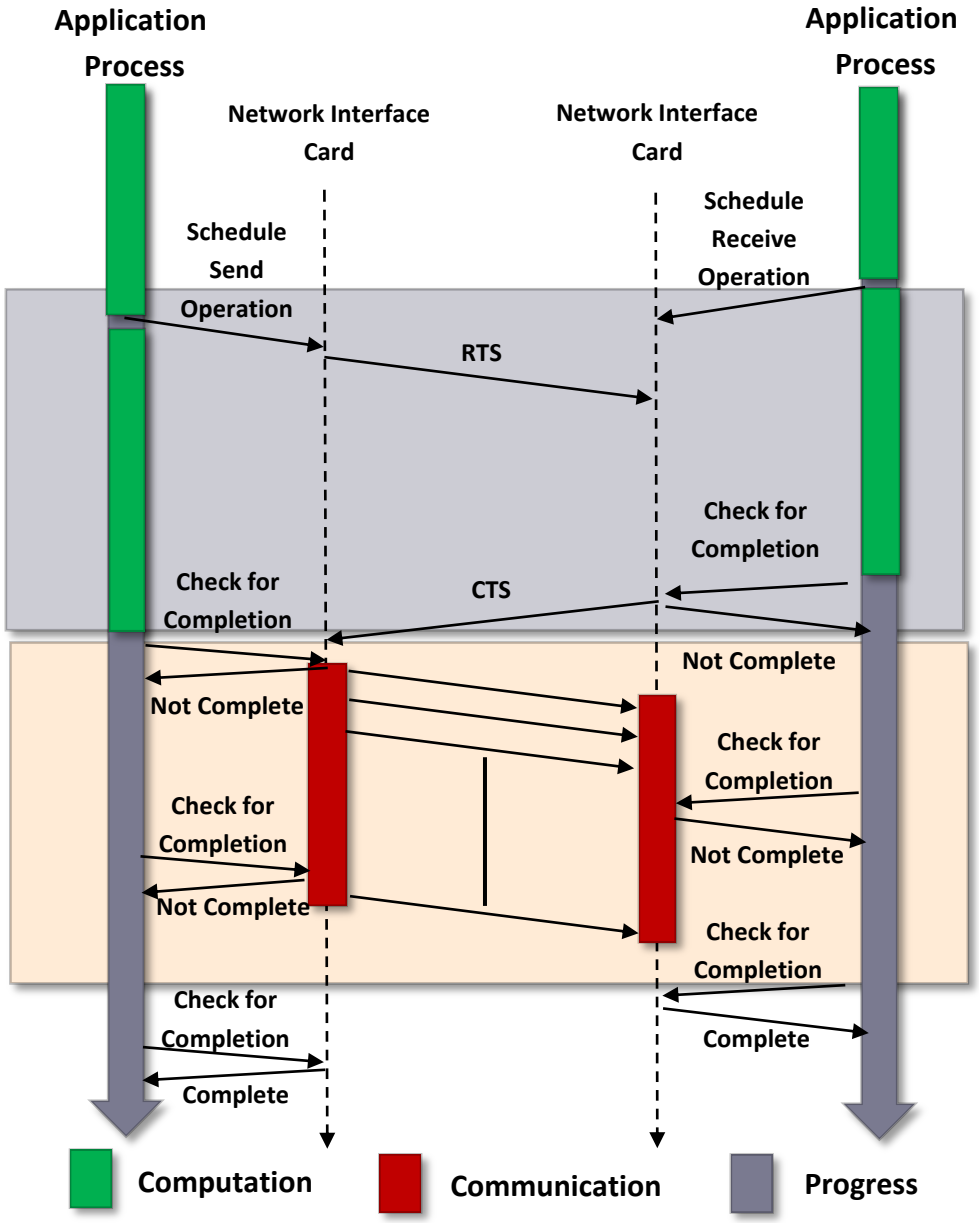


- Application processes schedule communication operation
- Network adapter progresses communication in the background
- Application process free to perform useful compute in the foreground
- **Overlap of computation and communication => Better Overall Application Performance**
- **Increased buffer requirement**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

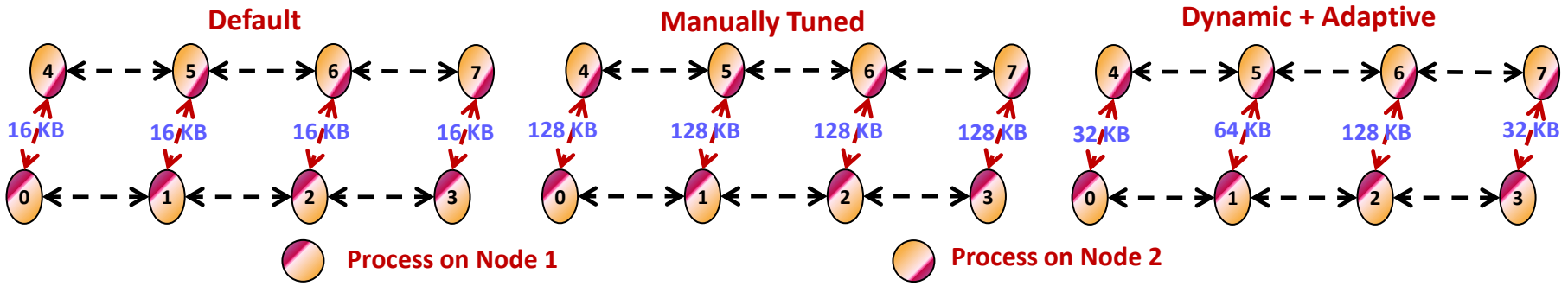
- **Poor overlap of computation and communication => Poor Overall Application Performance**

Dynamic and Adaptive MPI Point-to-point Communication Protocols

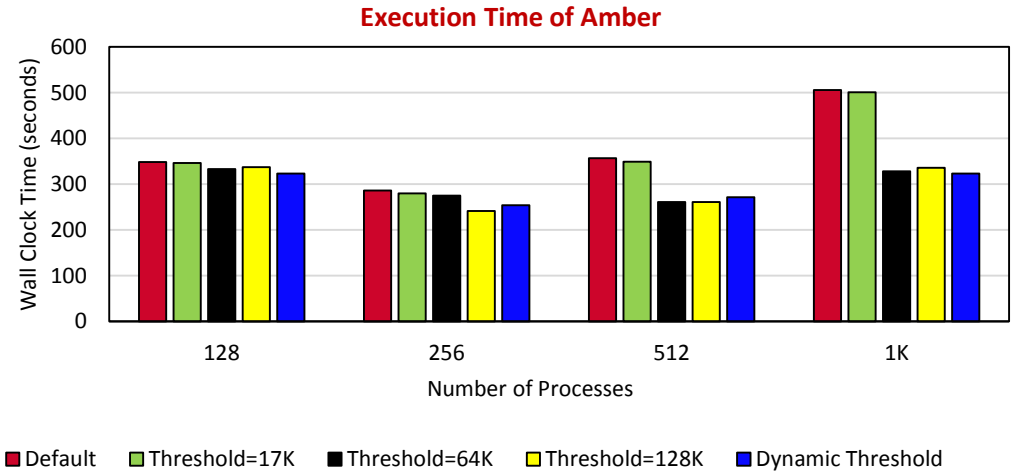
Desired Eager Threshold

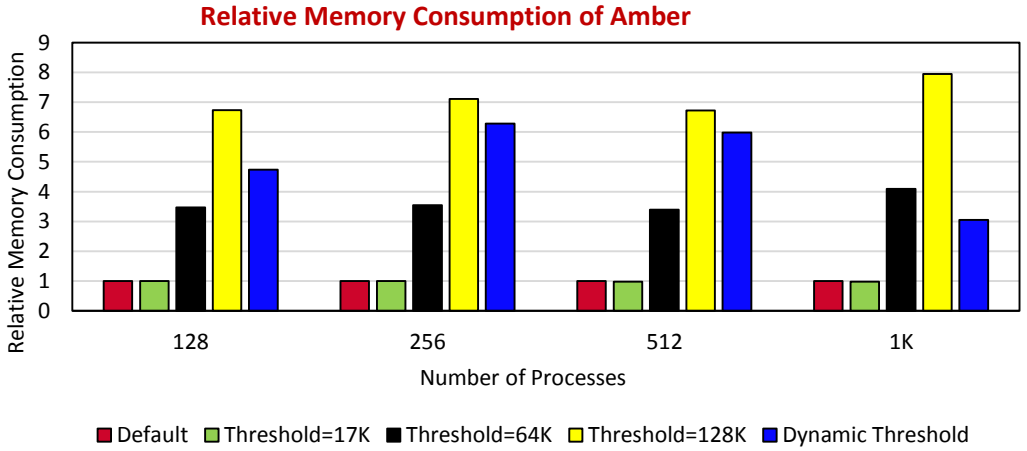
Process Pair	Eager Threshold (KB)
0 – 4	32
1 – 5	64
2 – 6	128
3 – 7	32

Eager Threshold for Example Communication Pattern with Different Designs



Default	Poor overlap; Low memory requirement	Low Performance; High Productivity
Manually Tuned	Good overlap; High memory requirement	High Performance; Low Productivity
Dynamic + Adaptive	Good overlap; Optimal memory requirement	High Performance; High Productivity





H. Subramoni, S. Chakraborty, D. K. Panda, Designing Dynamic & Adaptive MPI Point-to-Point Communication Protocols for Efficient Overlap of Computation & Communication, ISC'17 - **Best Paper**

Dynamic and Adaptive Tag Matching

Challenge

Tag matching is a significant overhead for receivers

Existing Solutions are

- Static and do not adapt dynamically to communication pattern
- Do not consider memory overhead

Solution

A new tag matching design

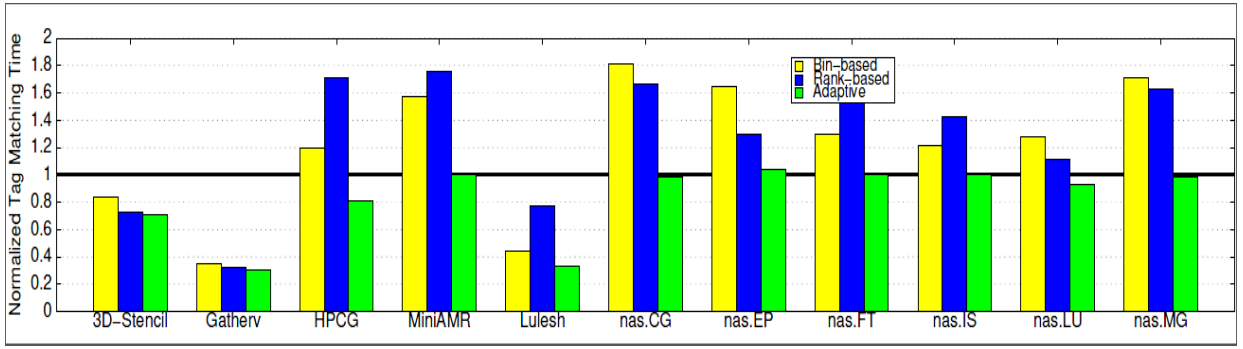
- Dynamically adapt to communication patterns
- Use different strategies for different ranks
- Decisions are based on the number of request object that must be traversed before hitting on the required one

Results

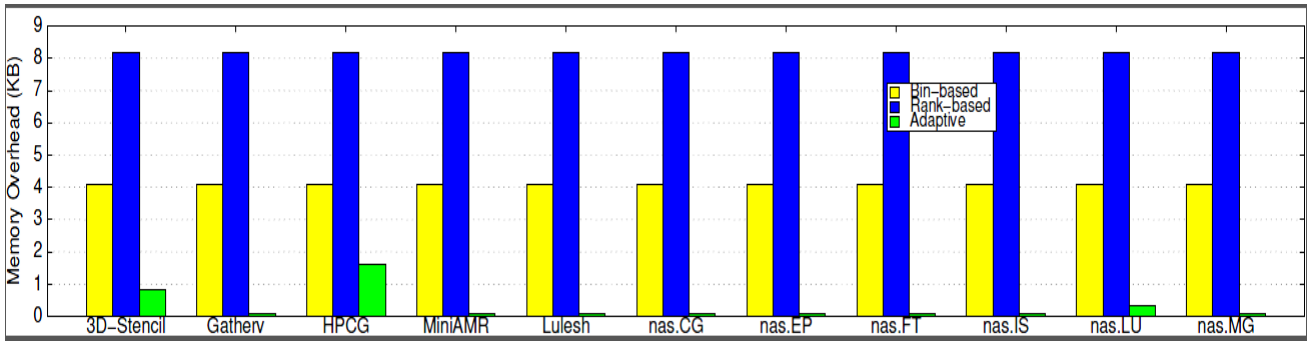
Better performance than other state-of-the art tag-matching schemes

Minimum memory consumption

Will be available in future MVAPICH2 releases



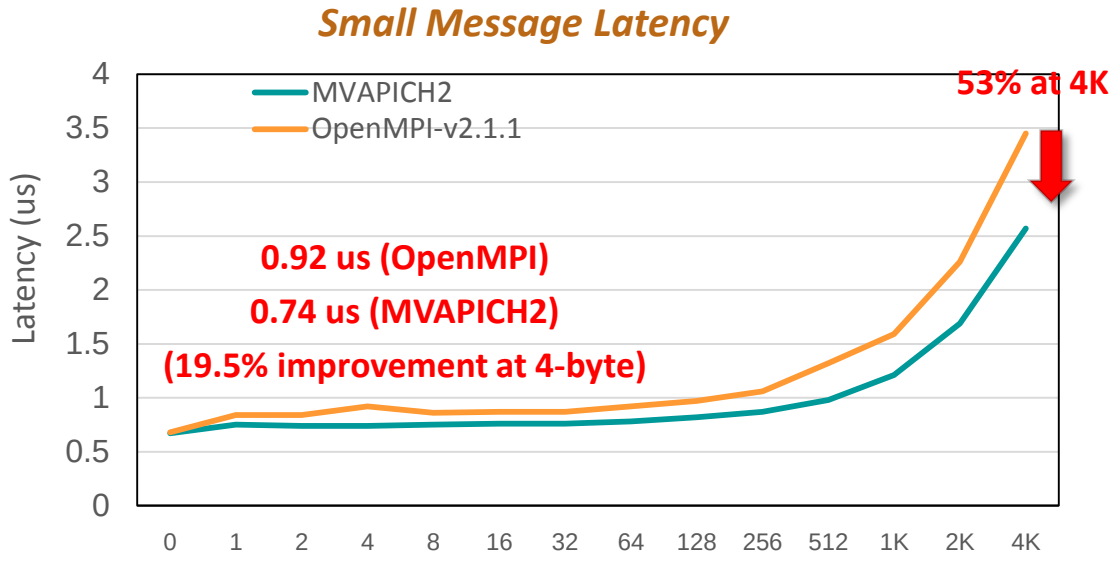
Normalized Total Tag Matching Time at 512 Processes Normalized to Default (Lower is Better)

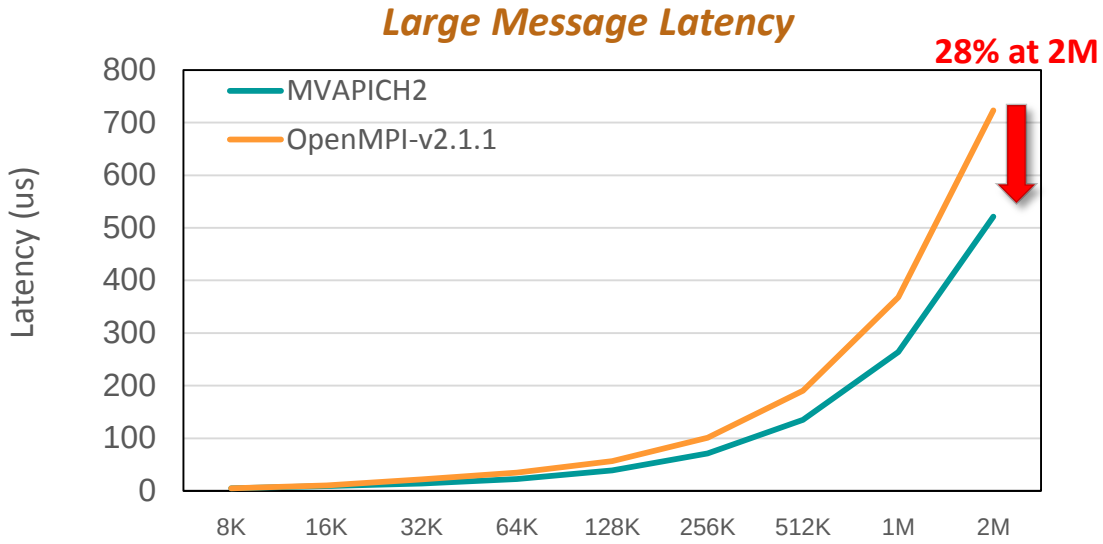


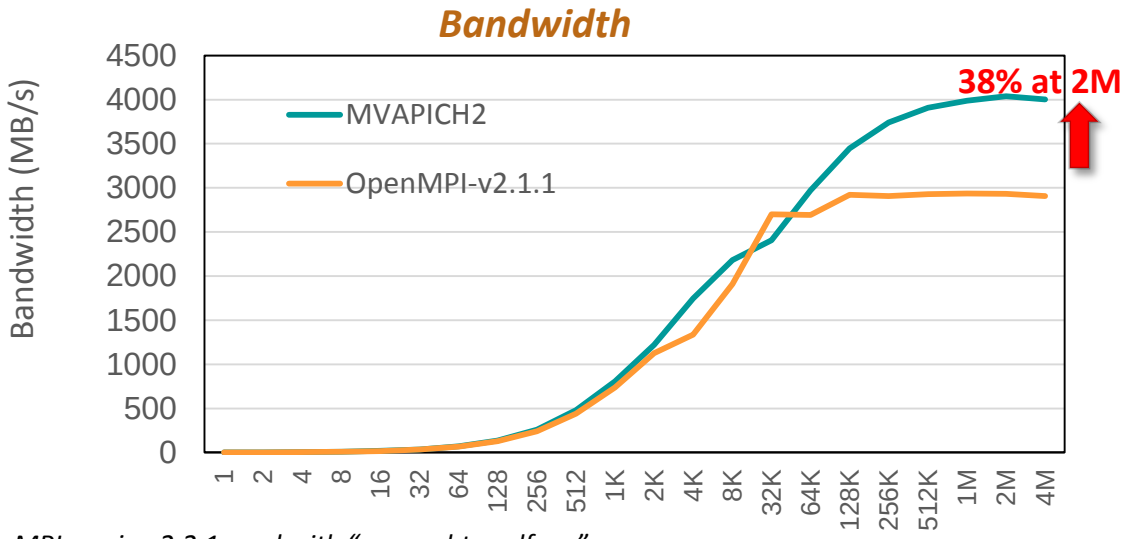
Normalized Memory Overhead per Process at 512 Processes Compared to Default (Lower is Better)

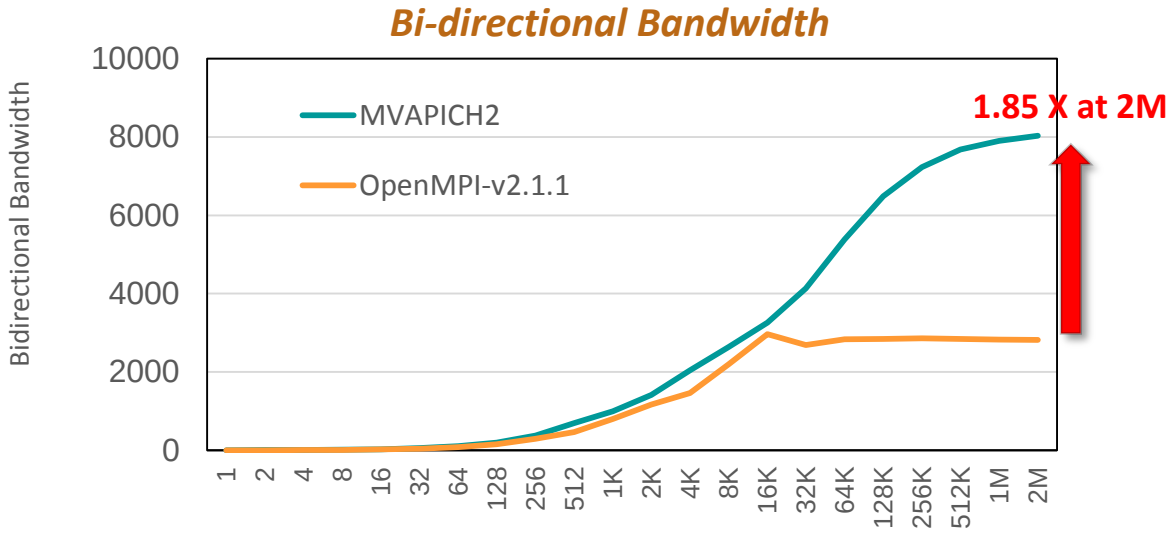
Adaptive and Dynamic Design for MPI Tag Matching; M. Bayatpour, H. Subramoni, S. Chakraborty, and D. K. Panda; IEEE Cluster 2016. [Best Paper Nominee]

Intra-node Point-to-point Performance on ARMv8









*** OpenMPI version 2.2.1 used with "—mca bta self,sm"*

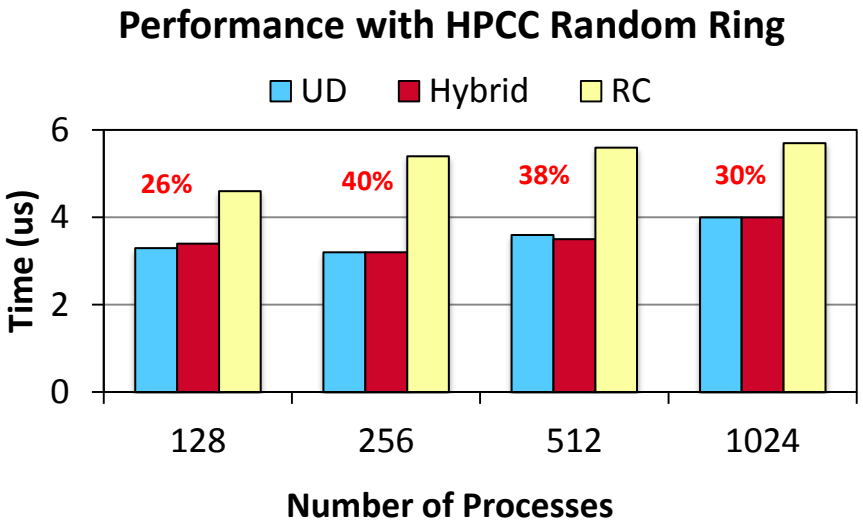
Platform: ARMv8 (aarch64) processor with 96 cores dual-socket CPU. Each socket contains 48 cores.

Available in MVAPICH2 2.3b

Presentation Overview

- Job start-up
- Point-to-point Inter-node Protocol
- **Transport Type Selection**
- Multi-rail and QoS
- Process Mapping and Point-to-point Intra-node Protocols
- Collectives
- MPI_T Support

Hybrid (UD/RC/XRC) Mode in MVAPICH2

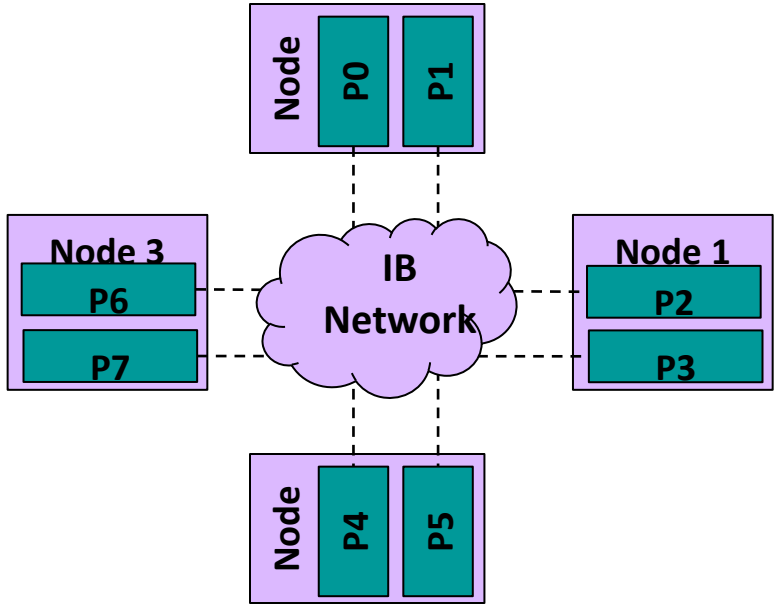


- Both UD and RC/XRC have benefits
  - Hybrid for the best of both
- Enabled by configuring MVAPICH2 with the –enable-hybrid
- Available since MVAPICH2 1.7 as integrated interface

Parameter	Significance	Default	Notes
MV2_USE_UD_HYBRID	• Enable / Disable use of UD transport in Hybrid mode	Enabled	• Always Enable
MV2_HYBRID_ENABLE_THRESHOLD_SIZE	• Job size in number of processes beyond which hybrid mode will be enabled	1024	• Uses RC/XRC connection until job size < threshold
MV2_HYBRID_MAX_RC_CONN	• Maximum number of RC or XRC connections created per process • Limits the amount of connection memory	64	• Prevents HCA QP cache thrashing

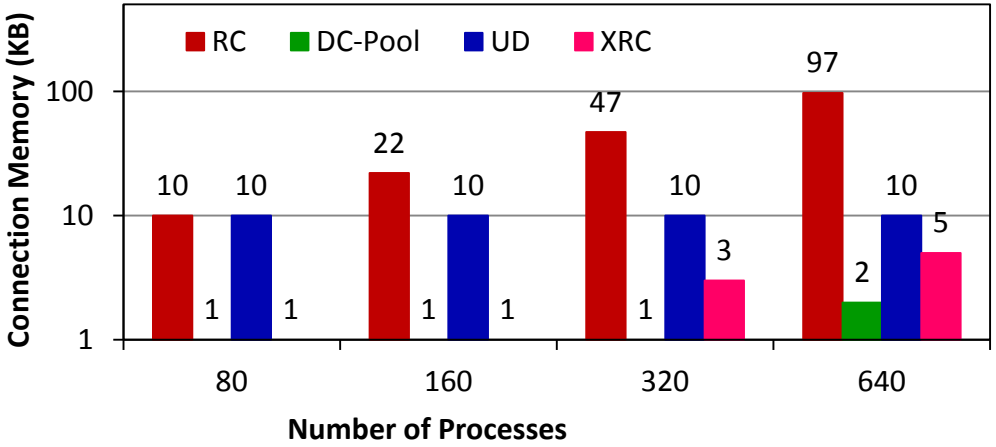
- **Refer to Running with Hybrid UD-RC/XRC section of MVAPICH2 user guide for more information**
- http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.3a-userguide.html#x1-690006.11

Minimizing Memory Footprint by Direct Connect (DC) Transport

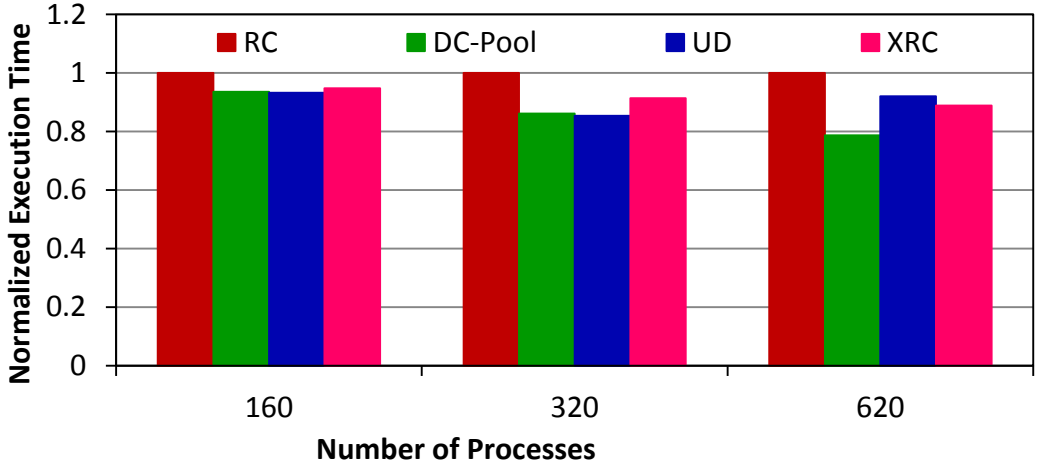


- Constant connection cost (*One QP for any peer*)
- Full Feature Set (RDMA, Atomics etc)
- Separate objects for send (DC Initiator) and receive (DC Target)
  - DC Target identified by "DCT Number"
  - Messages routed with (DCT Number, LID)
  - Requires same "DC Key" to enable communication
- Available since MVAPICH2-X 2.2a

Memory Footprint for Alltoall



NAMD - Apoa1: Large data set

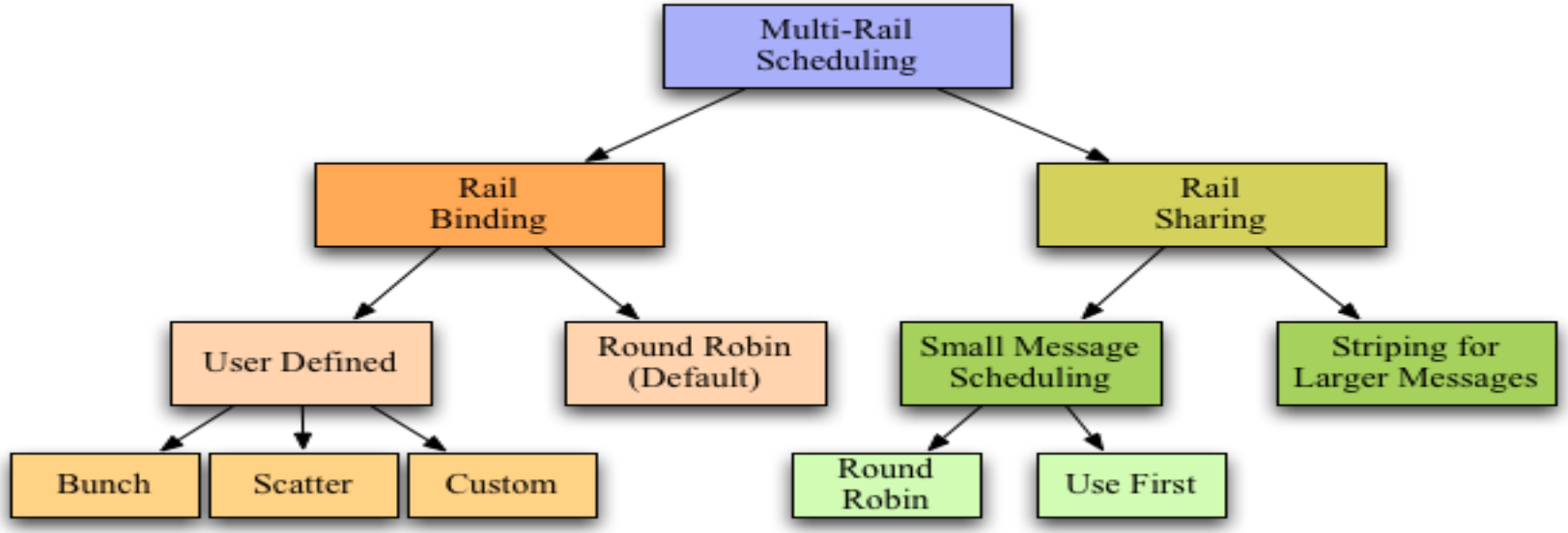


H. Subramoni, K. Hamidouche, A. Venkatesh, S. Chakraborty and D. K. Panda, Designing MPI Library with Dynamic Connected Transport (DCT) of InfiniBand : Early Experiences. IEEE International Supercomputing Conference (ISC '14)

Presentation Overview

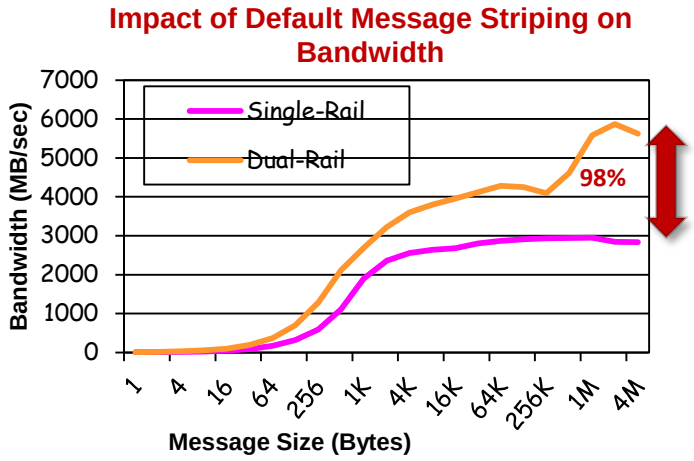
- Job start-up
- Point-to-point Inter-node Protocol
- Transport Type Selection
- **Multi-rail and QoS**
- Process Mapping and Point-to-point Intra-node Protocols
- Collectives
- MPI_T Support

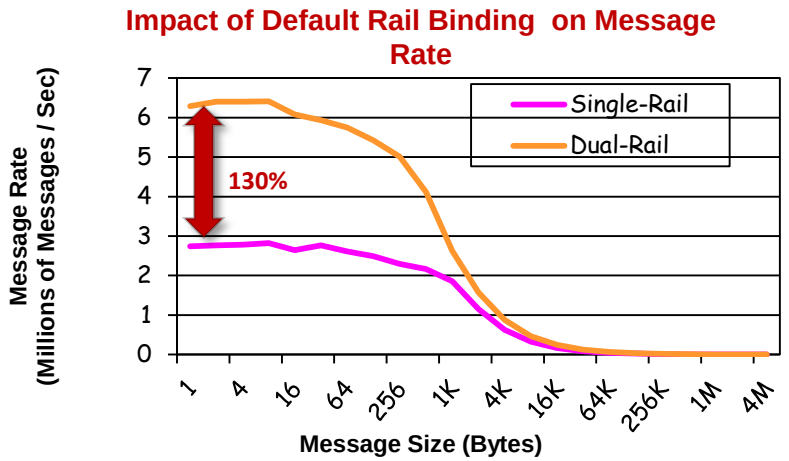
MVAPICH2 Multi-Rail Design

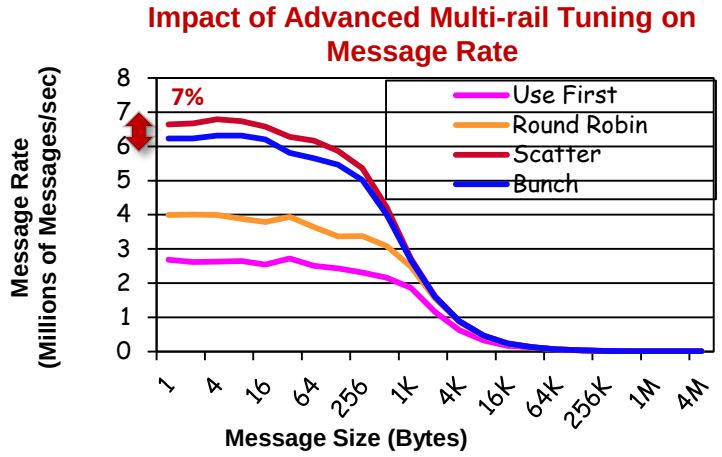


- What is a rail?
  - **HCA, Port, Queue Pair**
- Automatically detects and uses all active HCAs in a system
  - Automatically handles heterogeneity
- Supports multiple rail usage policies
  - Rail Sharing – Processes share all available rails
  - Rail Binding – Specific processes are bound to specific rails

Performance Tuning on Multi-Rail Clusters







Two 24-core Magny Cours nodes with two Mellanox ConnectX QDR adapters

Six pairs with OSU Multi-Pair bandwidth and messaging rate benchmark

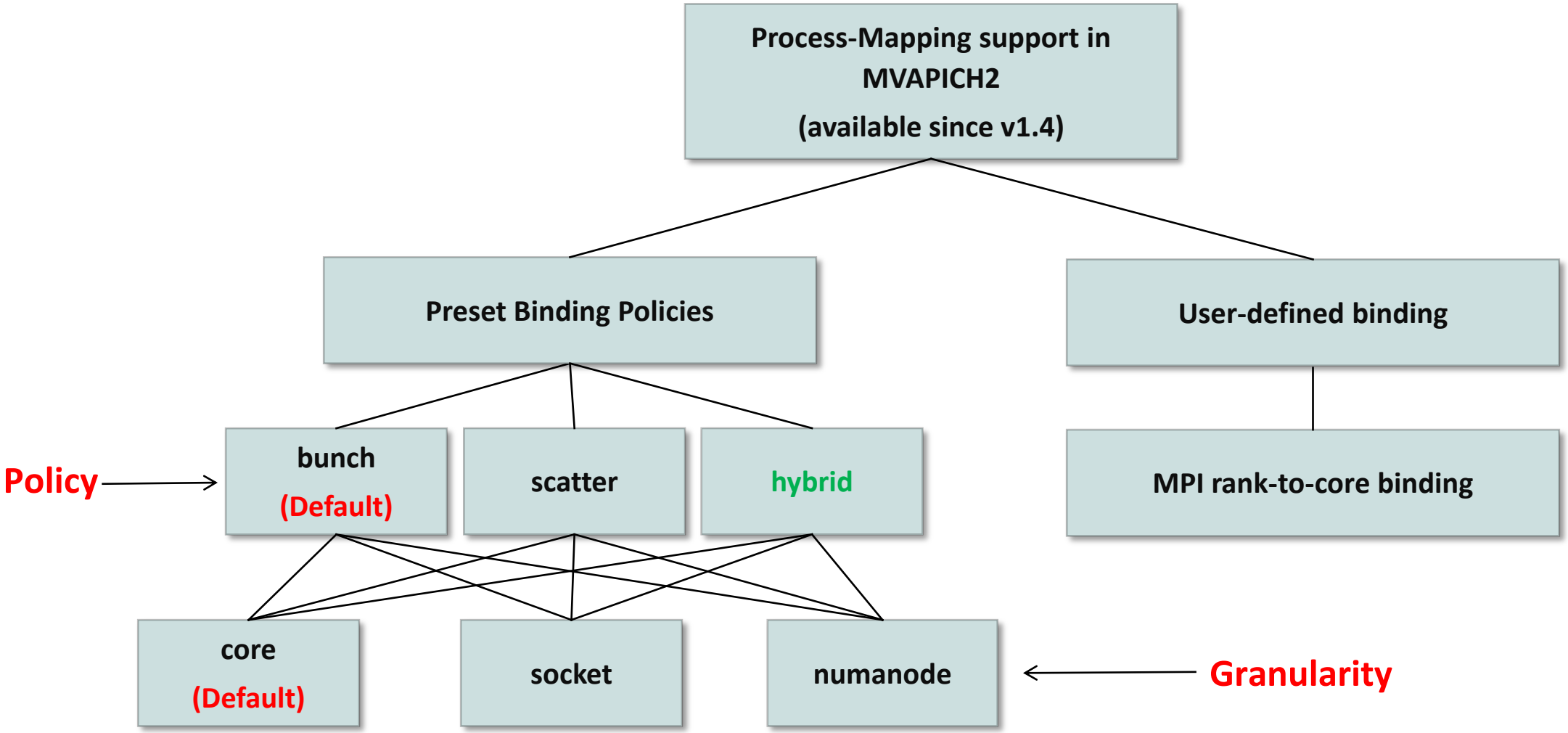
Parameter	Significance	Default	Notes
MV2_IBA_HCA	• Manually set the HCA to be used	Unset	• To get names of HCA ibstat | grep "^CA"
MV2_DEFAULT_PORT	• Select the port to use on a active multi port HCA	0	• Set to use different port
MV2_RAIL_SHARING_LARGE_MSG_THRESHOLD	• Threshold beyond which striping will take place	16 Kbyte	
MV2_RAIL_SHARING_POLICY	• Choose multi-rail rail sharing / binding policy • For Rail Sharing set to USE_FIRST or ROUND_ROBIN • Set to FIXED_MAPPING for advanced rail binding options	Rail Binding in Round Robin mode	• Advanced tuning can result in better performance
MV2_PROCESS_TO_RAIL_MAPPING	• Determines how HCAs will be mapped to the rails	BUNCH	• Options: SCATTER and custom list

- **Refer to Enhanced design for Multiple-Rail section of MVAPICH2 user guide for more information**
- **http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.3a-userguide.html#x1-700006.12**

Presentation Overview

- Job start-up
- Point-to-point Inter-node Protocol
- Transport Type Selection
- Multi-rail and QoS
- **Process Mapping and Point-to-point Intra-node Protocols**
- Collectives
- MPI_T Support

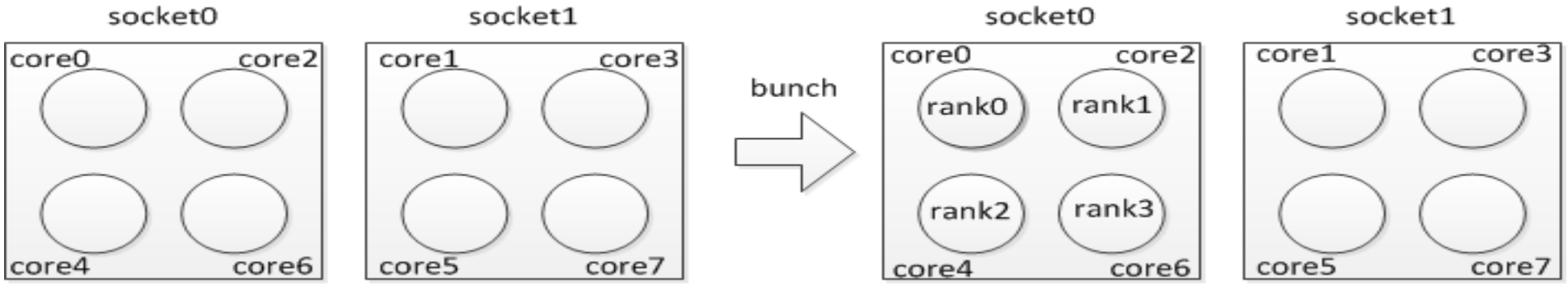
Process Mapping support in MVAPICH2



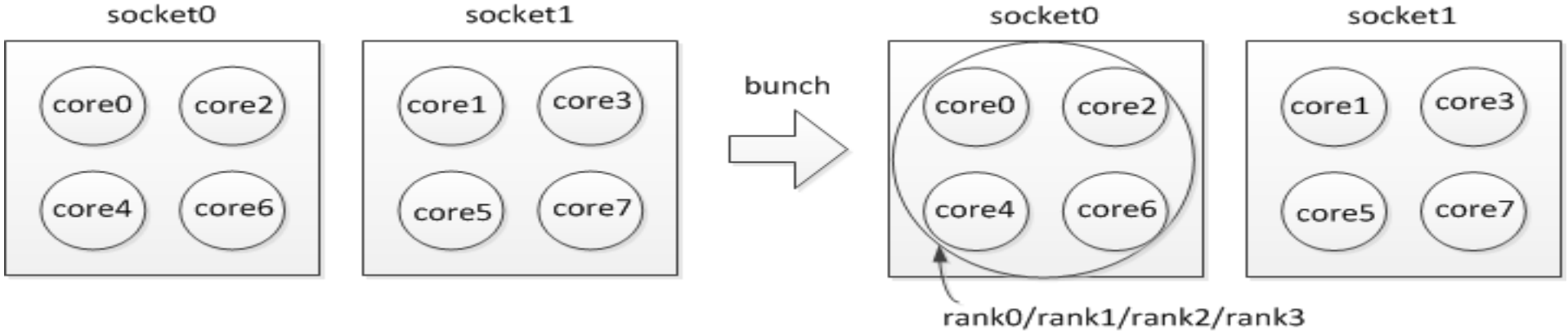
- MVAPICH2 detects processor architecture at job-launch

Preset Process-binding Policies – Bunch

- "Core" level "Bunch" mapping (Default)
  - MV2_CPU_BINDING_POLICY=bunch

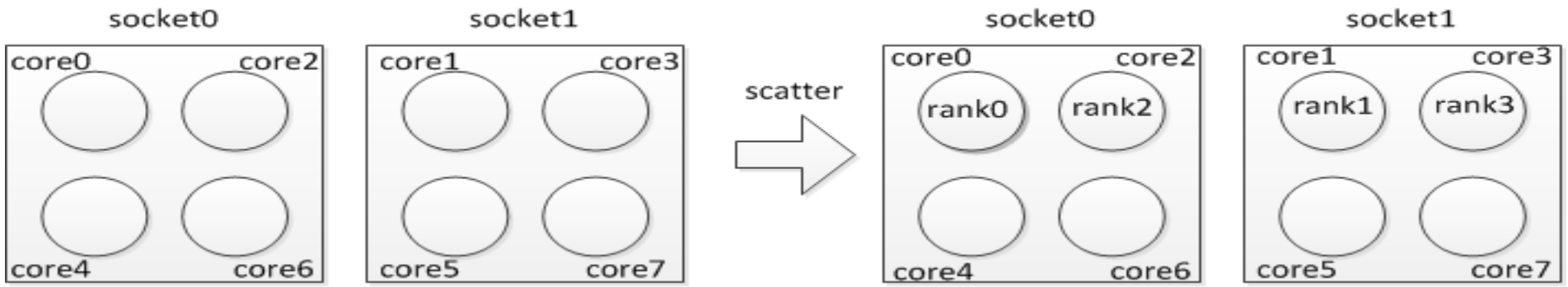


- "Socket/Numanode" level "Bunch" mapping
  - MV2_CPU_BINDING_LEVEL=socket MV2_CPU_BINDING_POLICY=bunch

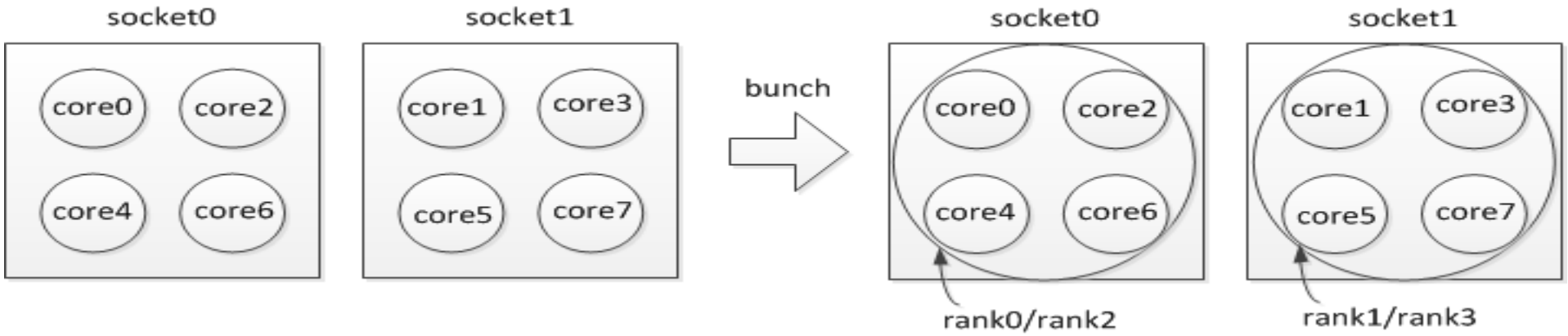


Preset Process-binding Policies – Scatter

- "Core" level "Scatter" mapping
  - MV2_CPU_BINDING_POLICY=scatter



- "Socket/Numanode" level "Scatter" mapping
  - MV2_CPU_BINDING_LEVEL=socket MV2_CPU_BINDING_POLICY=scatter

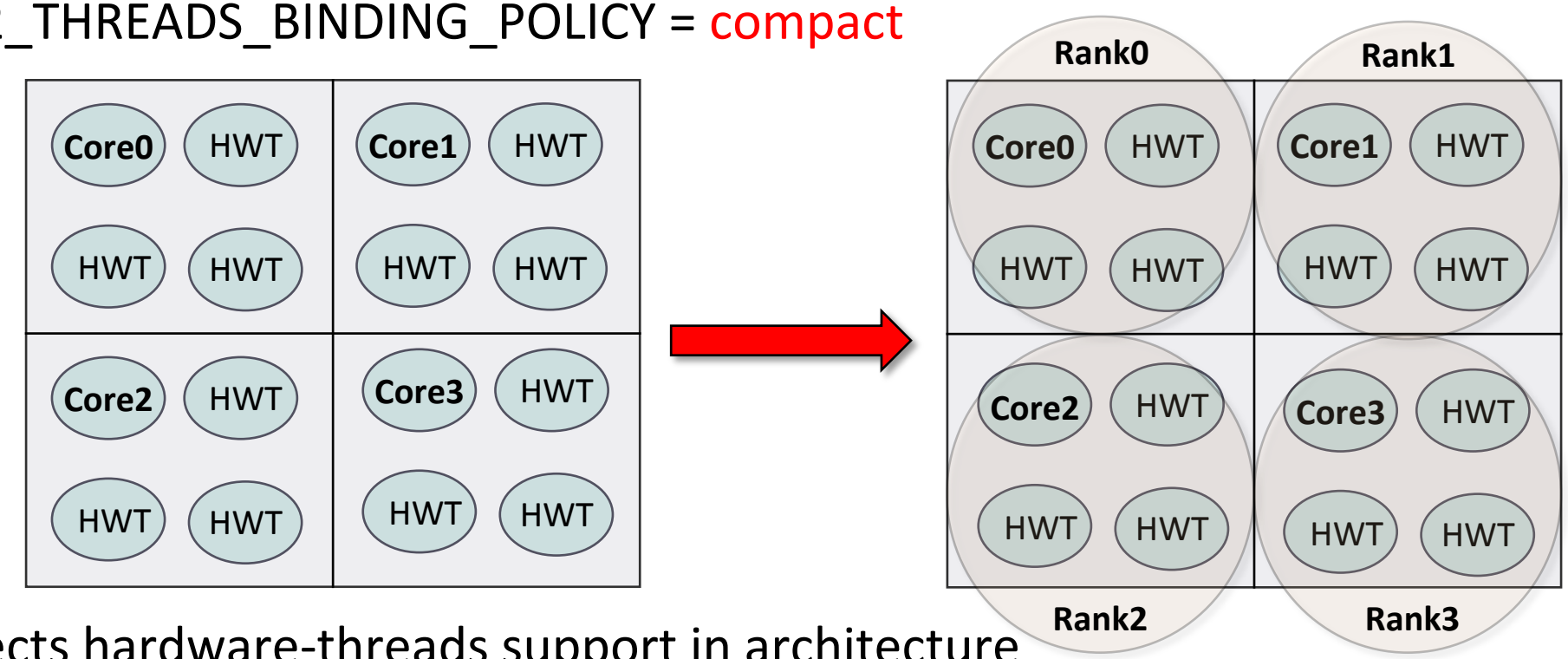


Process and thread binding policies in hybrid MPI+Threads

- A new process binding policy – "hybrid"
  - MV2_CPU_BINDING_POLICY = hybrid
- A new environment variable for co-locating Threads with MPI Processes
  - MV2_THREADS_PER_PROCESS = k
  - Automatically set to OMP_NUM_THREADS if OpenMP is being used
  - Provides a hint to the MPI runtime to spare resources for application threads.
- New variable for threads bindings with respect to parent process and architecture
  - MV2_THREADS_BINDING_POLICY = {linear | compact}
    - Linear – binds MPI ranks and OpenMP threads sequentially (one after the other)
      - Recommended to be used on non-hyperthreaded systems with MPI+OpenMP
    - Compact – binds MPI rank to physical-core and locates respective OpenMP threads on hardware threads
      - Recommended to be used on multi-/many-cores e.g., KNL, POWER8, and hyper-threaded Xeon, etc.

Binding Example in Hybrid (MPI+Threads)

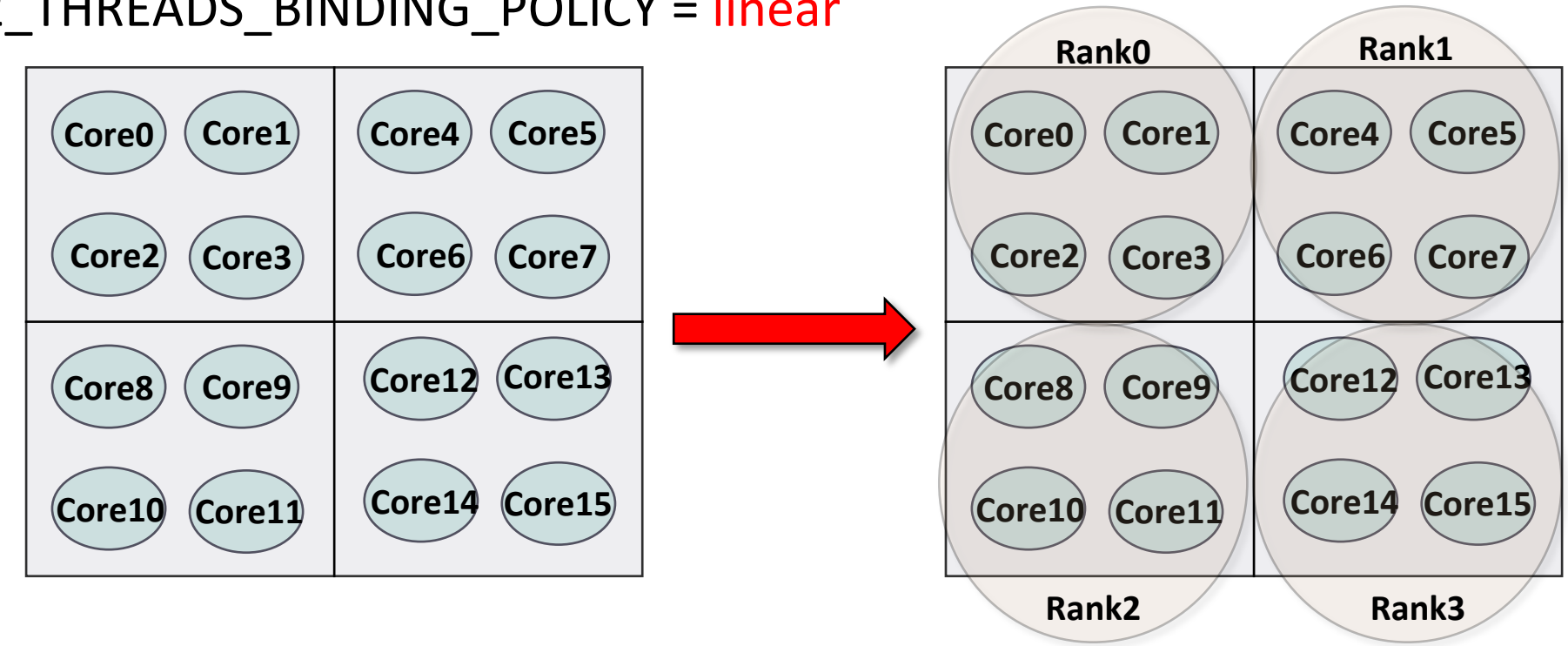
- MPI Processes = 4, OpenMP Threads per Process = 4
- MV2_CPU_BINDING_POLICY = hybrid
- MV2_THREADS_PER_PROCESS = 4
- MV2_THREADS_BINDING_POLICY = compact



- Detects hardware-threads support in architecture
- Assigns MPI ranks to physical cores and respective OpenMP Threads to HW threads

Binding Example in Hybrid (MPI+Threads) ---- Cont'd

- MPI Processes = 4, OpenMP Threads per Process = 4
- MV2_CPU_BINDING_POLICY = hybrid
- MV2_THREADS_PER_PROCESS = 4
- MV2_THREADS_BINDING_POLICY = linear



- MPI Rank-0 with its 4-OpenMP threads gets bound on Core-0 through Core-3, and so on

User-Defined Process Mapping

- User has complete-control over process-mapping
- To run 4 processes on cores 0, 1, 4, 5:
  - $ mpirun_rsh -np 4 -hostfile hosts MV2_CPU_MAPPING=0:1:4:5 ./a.out
- Use ',' or '-' to bind to a set of cores:
  - $mpirun_rsh -np 64 -hostfile hosts MV2_CPU_MAPPING=0,2-4:1:5:6 ./a.out
- Is process binding working as expected?
  - **MV2_SHOW_CPU_BINDING=1**
    - Display CPU binding information
    - Launcher independent
    - Example
      - MV2_SHOW_CPU_BINDING=1 MV2_CPU_BINDING_POLICY=scatter

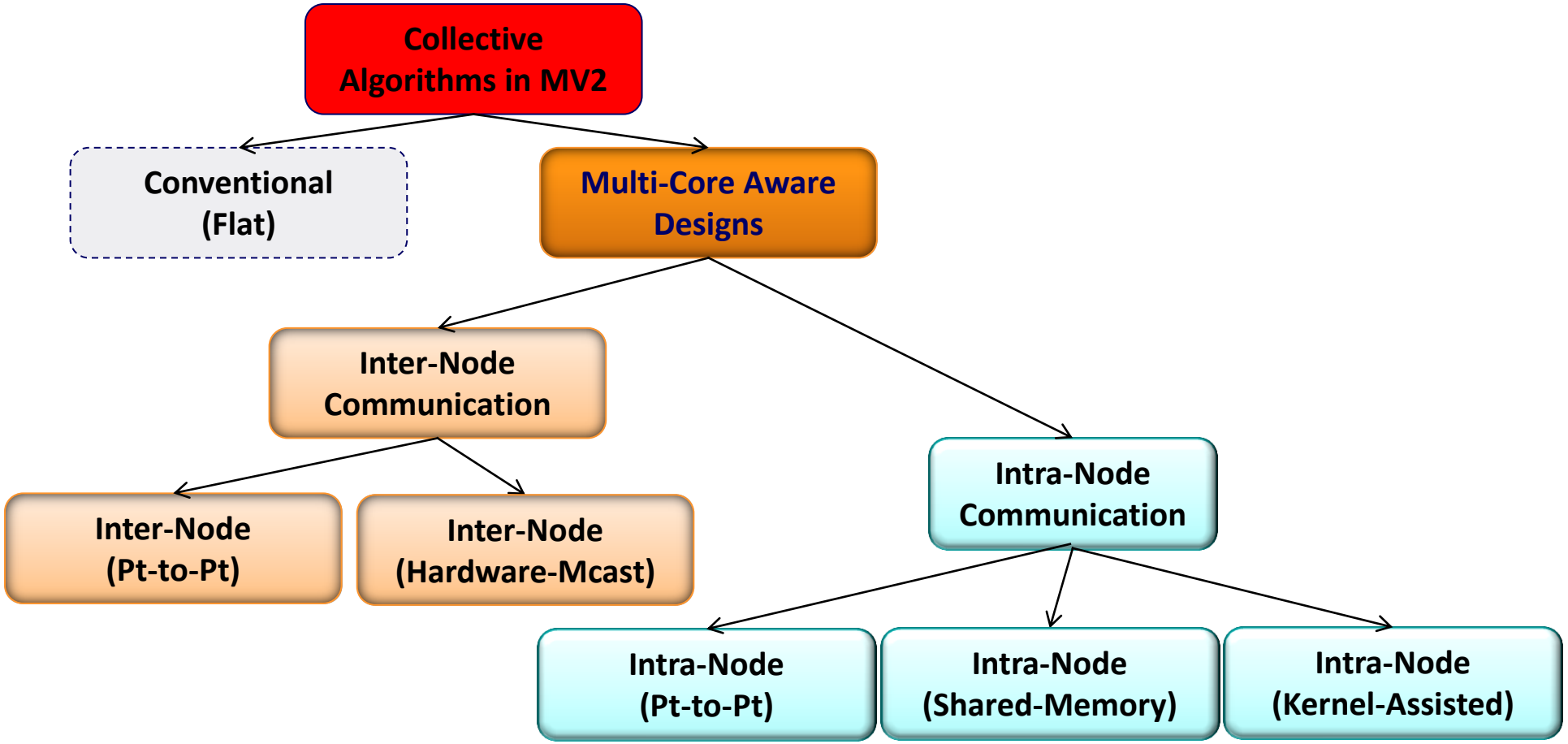
```
-------------CPU AFFINITY-------------
        RANK:0  CPU_SET:   0
        RANK:1  CPU_SET:   8
```

- **Refer to Running with Efficient CPU (Core) Mapping section of MVAPICH2 user guide for more information**
- http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.3a-userguide.html#x1-600006.5

Presentation Overview

- Job start-up
- Point-to-point Inter-node Protocol
- Transport Type Selection
- Multi-rail and QoS
- Process Mapping and Point-to-point Intra-node Protocols
- **Collectives**
- MPI_T Support

Collective Communication in MVAPICH2

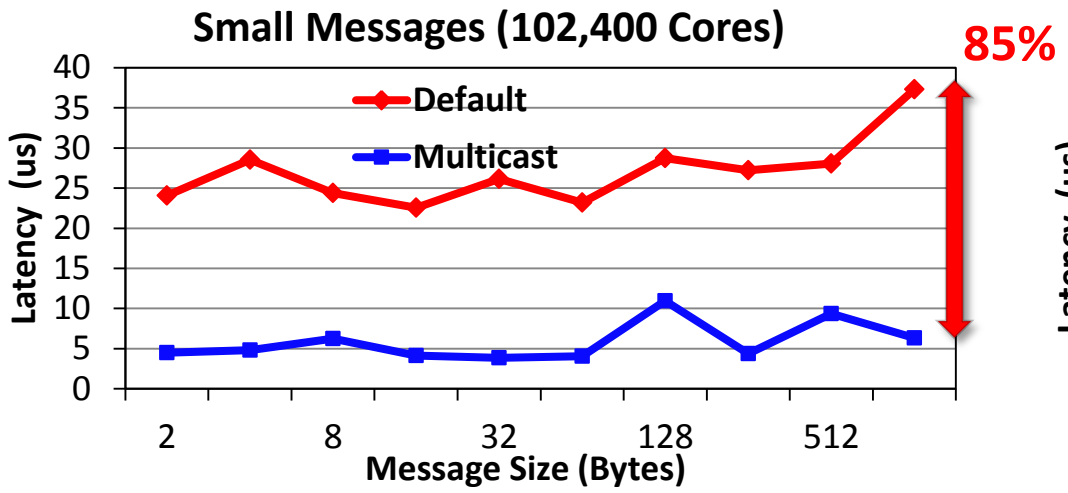


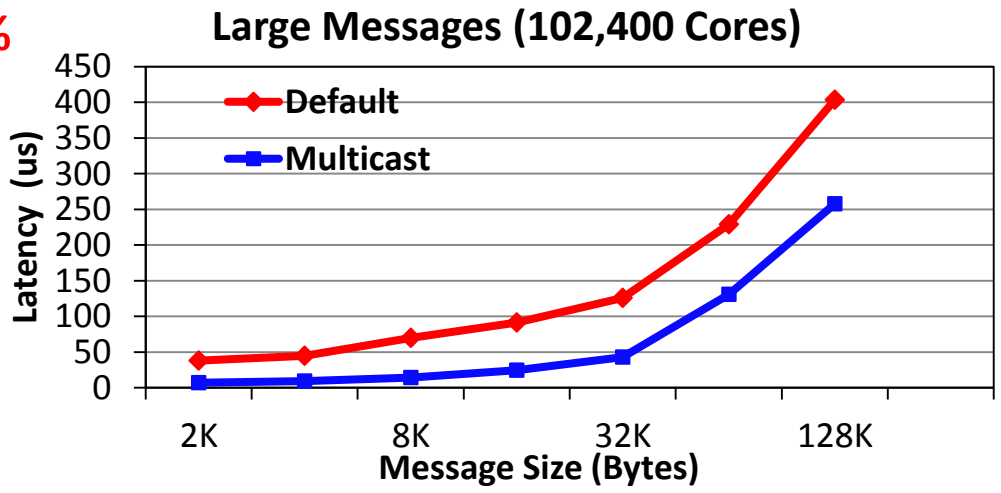
Run-time flags:

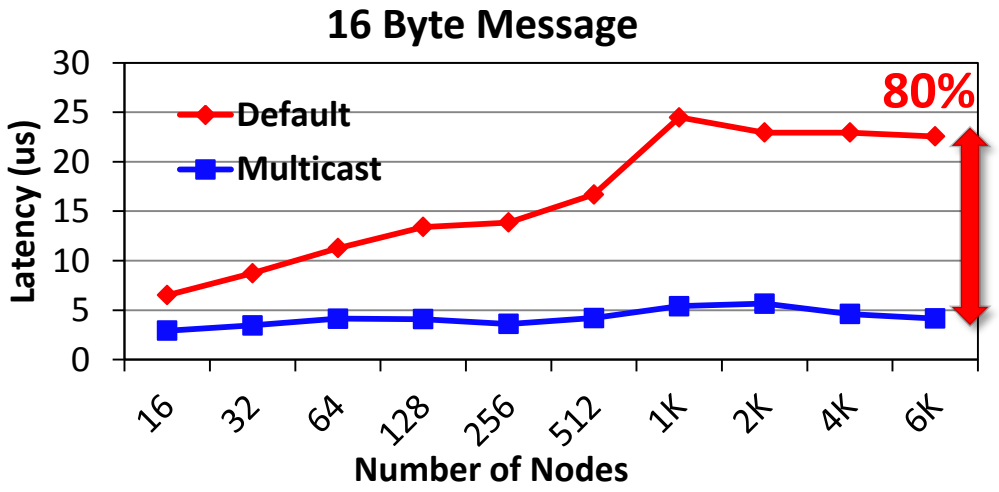
All shared-memory based collectives : MV2_USE_SHMEM_COLL (Default: ON)

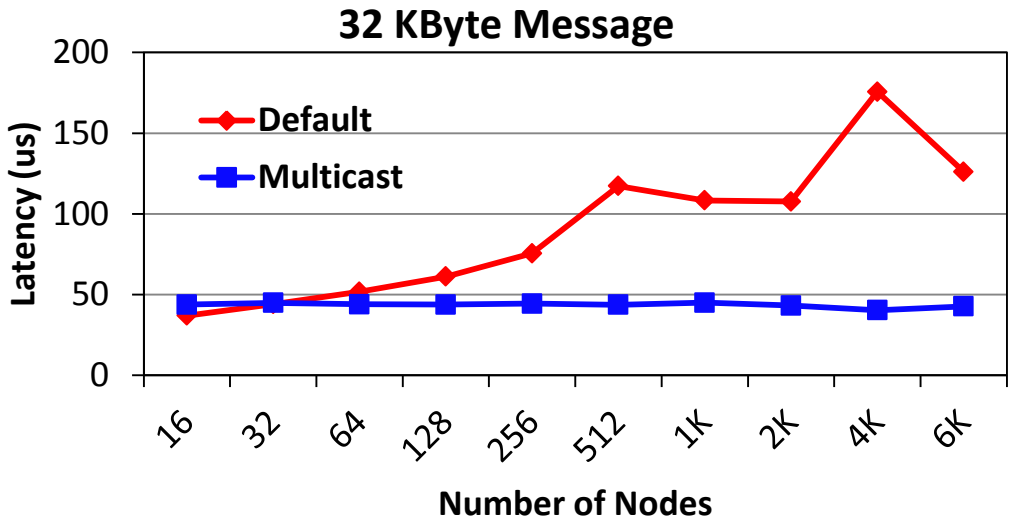
Hardware Mcast-based collectives : MV2_USE_MCAST (Default : OFF)

Hardware Multicast-aware MPI_Bcast on TACC Stampede



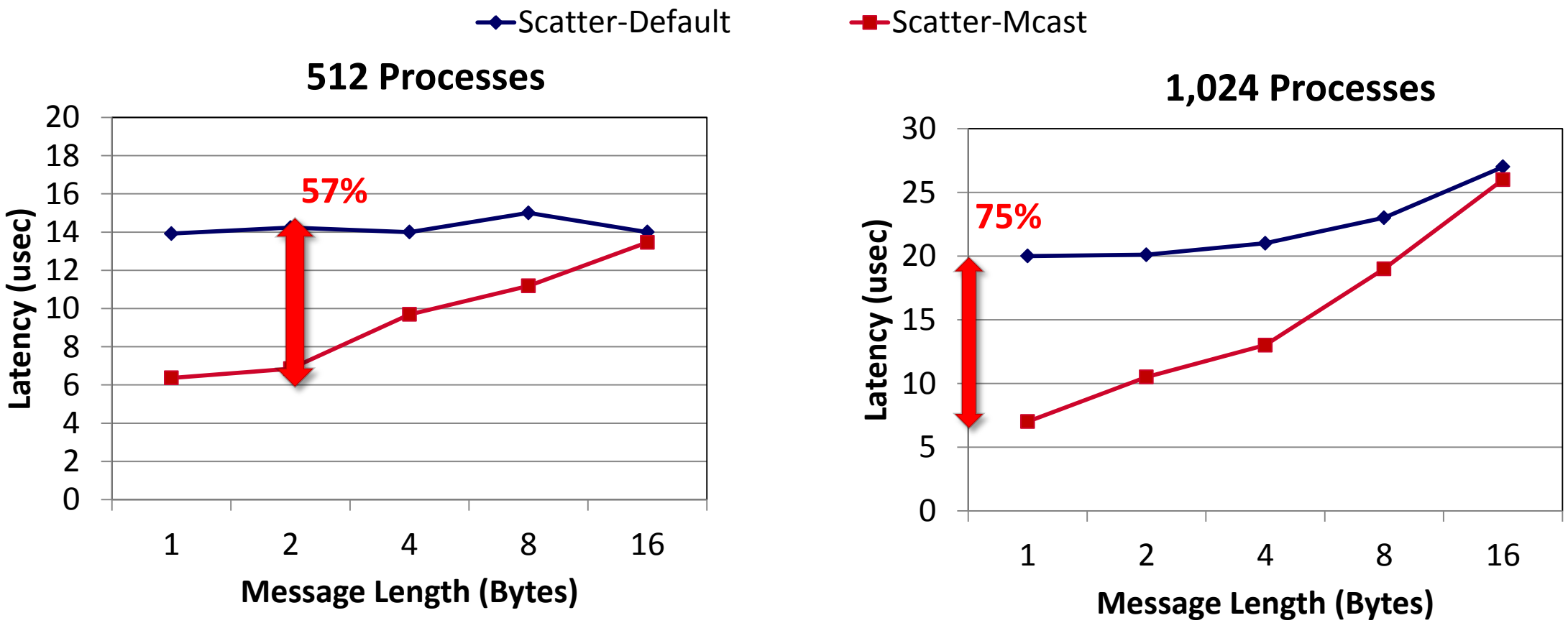






- MCAST-based designs improve latency of MPI_Bcast by up to **85%**
- Use MV2_USE_MCAST=1 to enable MCAST-based designs

MPI_Scatter - Benefits of using Hardware-Mcast

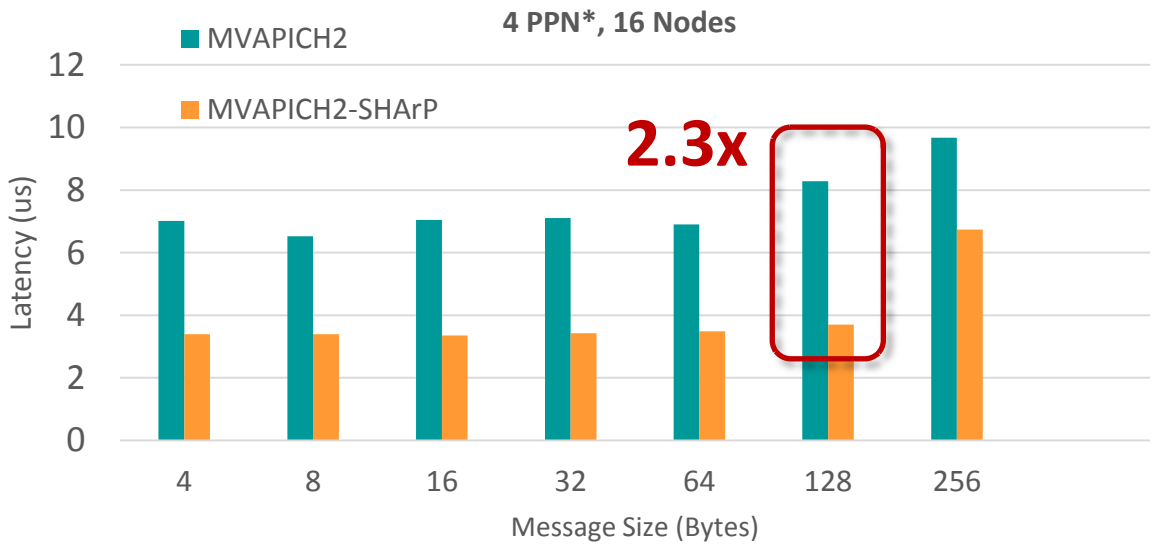


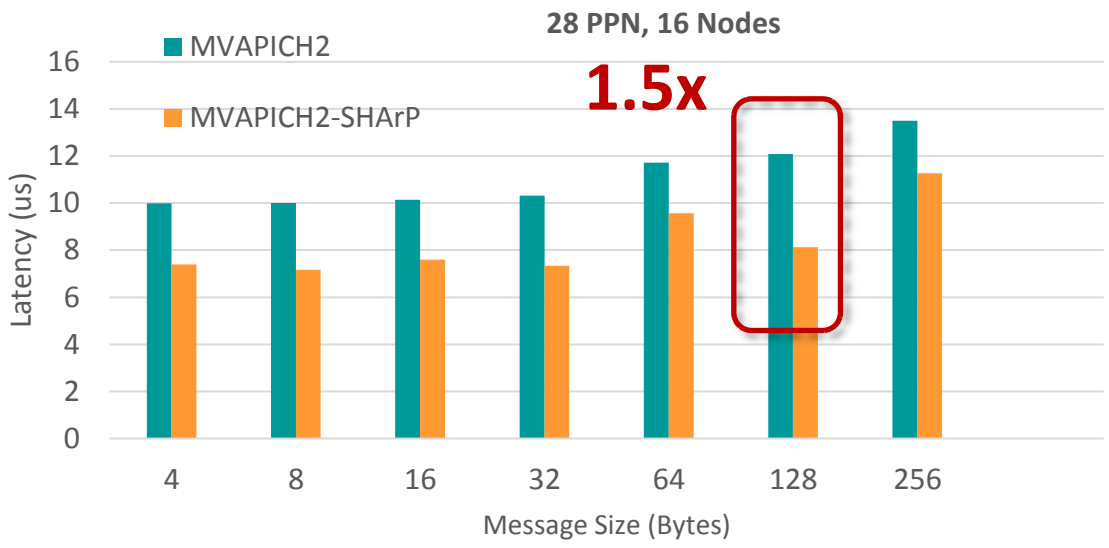
- Enabling MCAST-based designs for MPI_Scatter improves small message up to **75%**

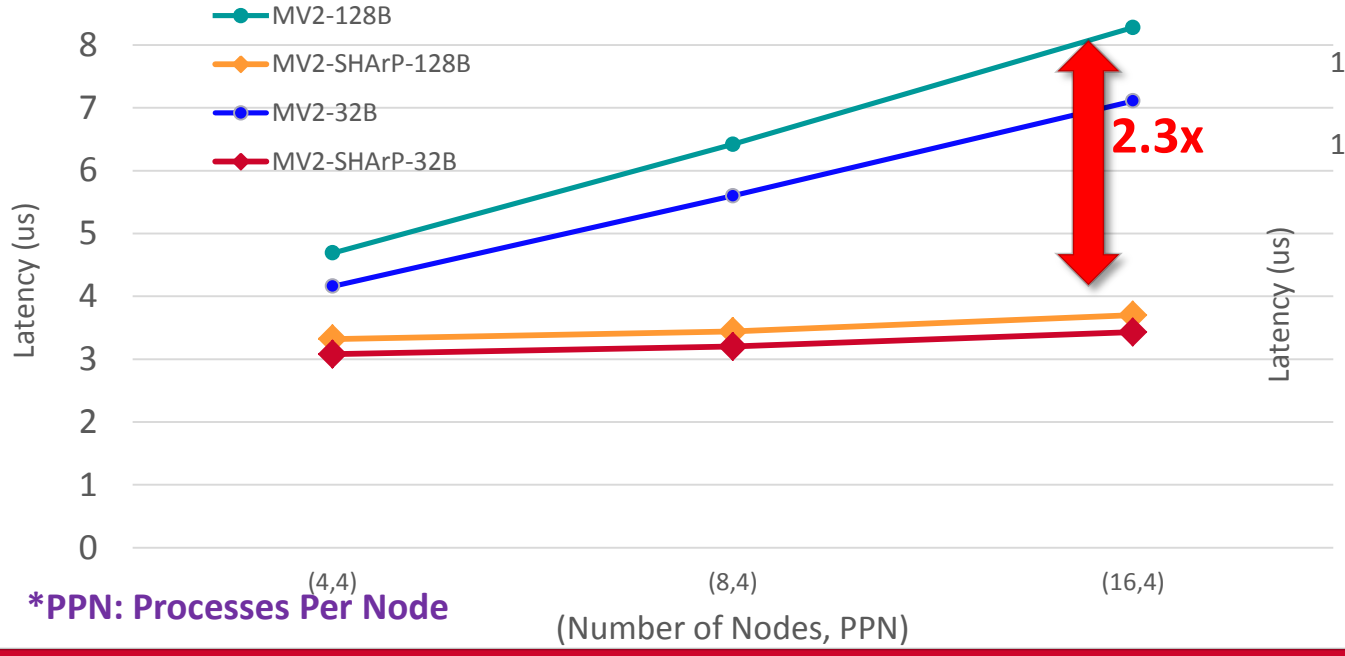
Parameter	Description	Default
MV2_USE_MCAST = 1	Enables hardware Multicast features	Disabled
--enable-mcast	Configure flag to enable	Enabled

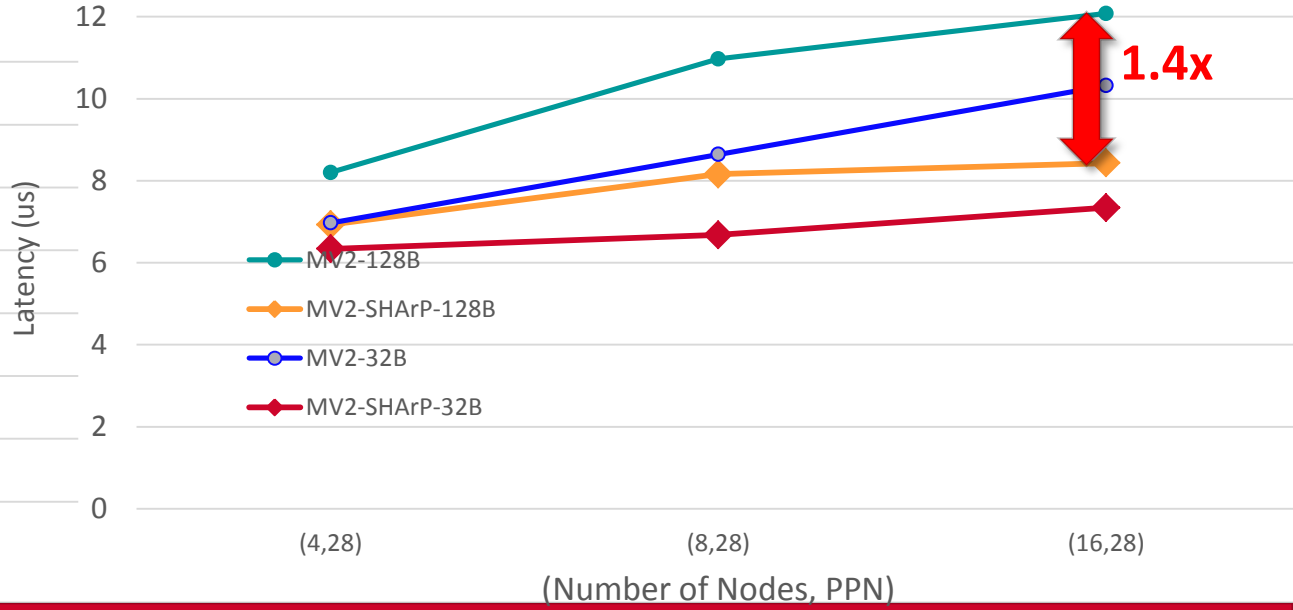
Advanced Allreduce Collective Designs Using SHArP

osu_allreduce (OSU Micro Benchmark) using MVAPICH2 2.3b



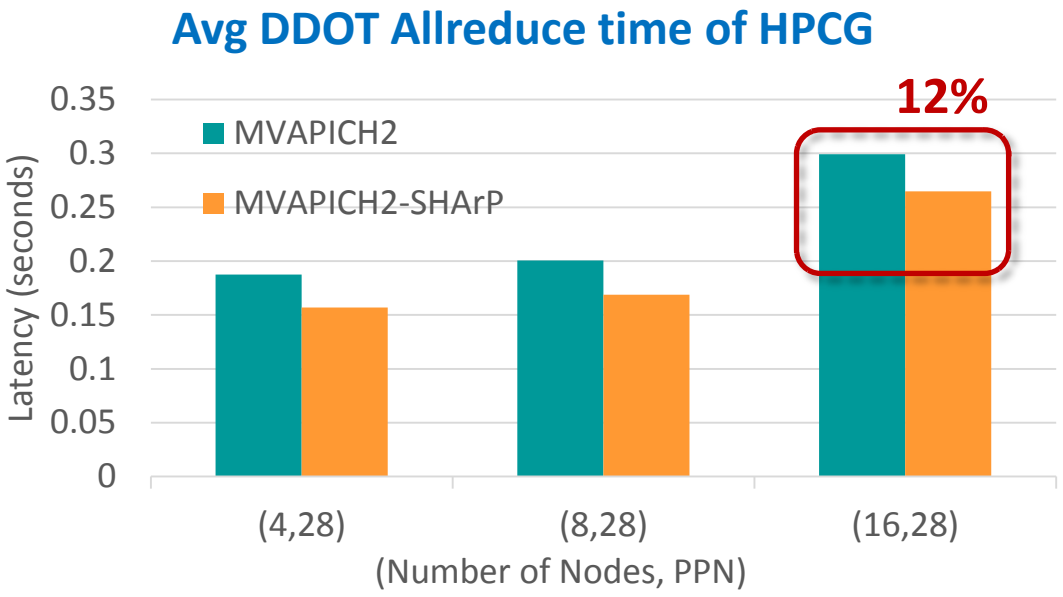


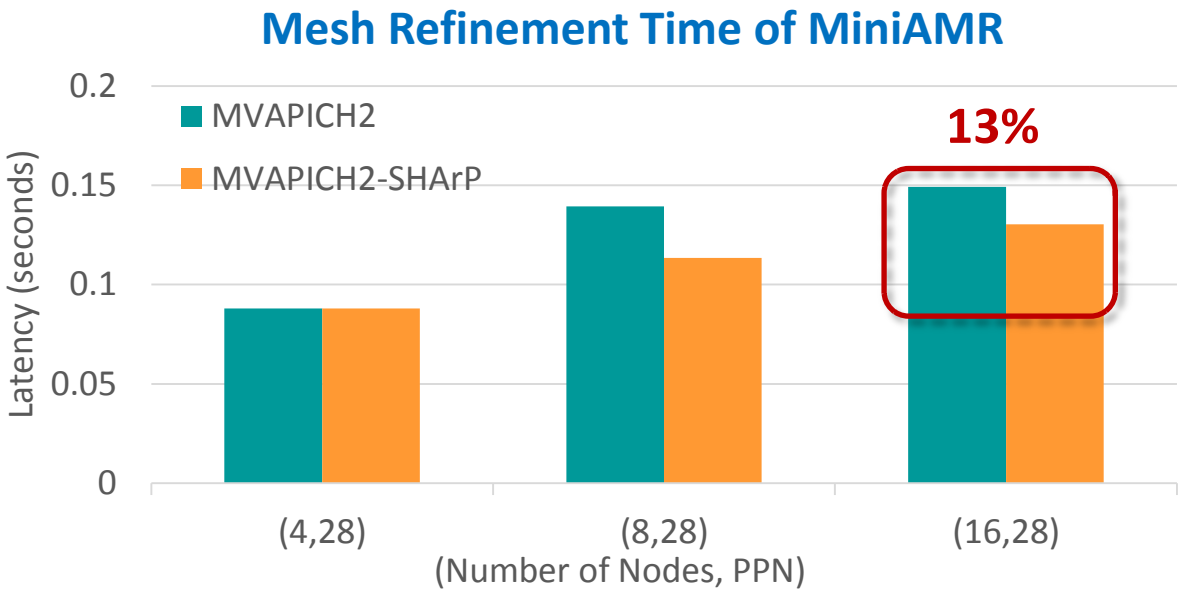




*PPN: Processes Per Node

Benefits of SHARP at Application Level



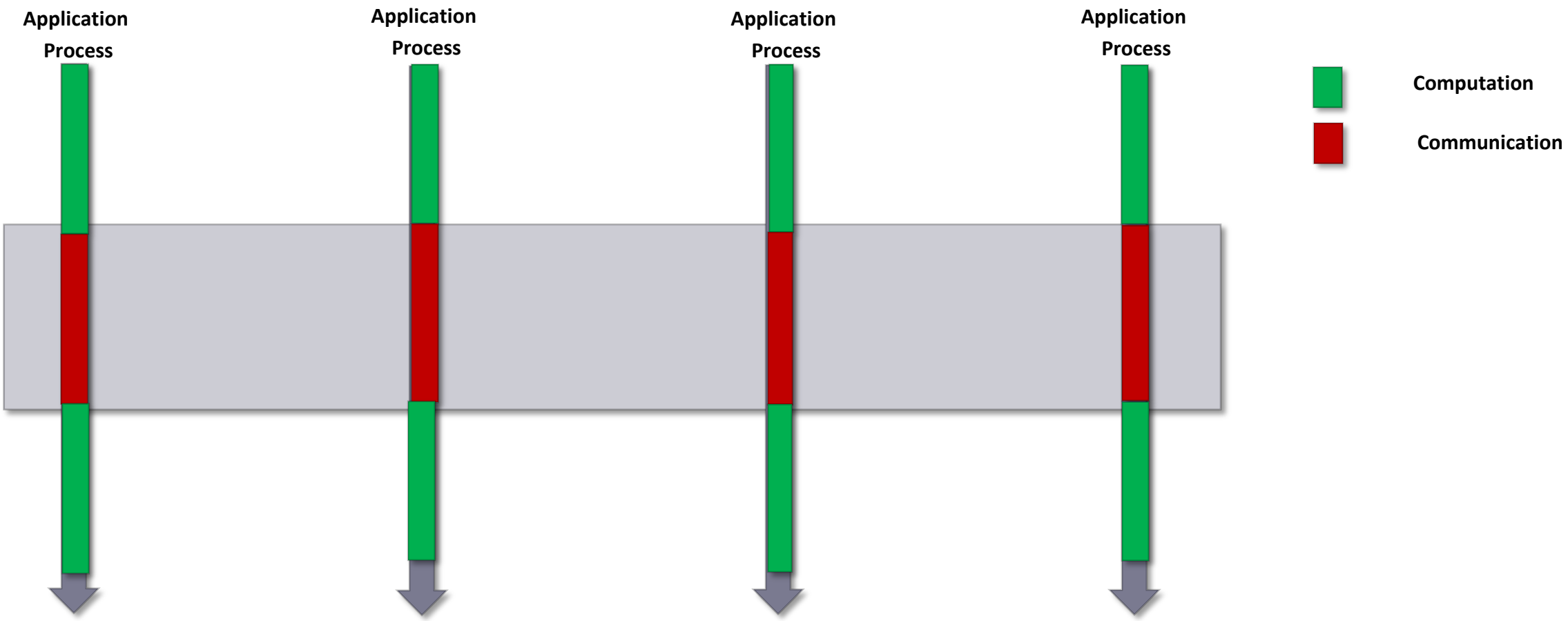


SHARP support available since MVAPICH2 2.3a

Parameter	Description	Default
MV2_ENABLE_SHARP=1	Enables SHARP-based collectives	Disabled
--enable-sharp	Configure flag to enable SHARP	Disabled

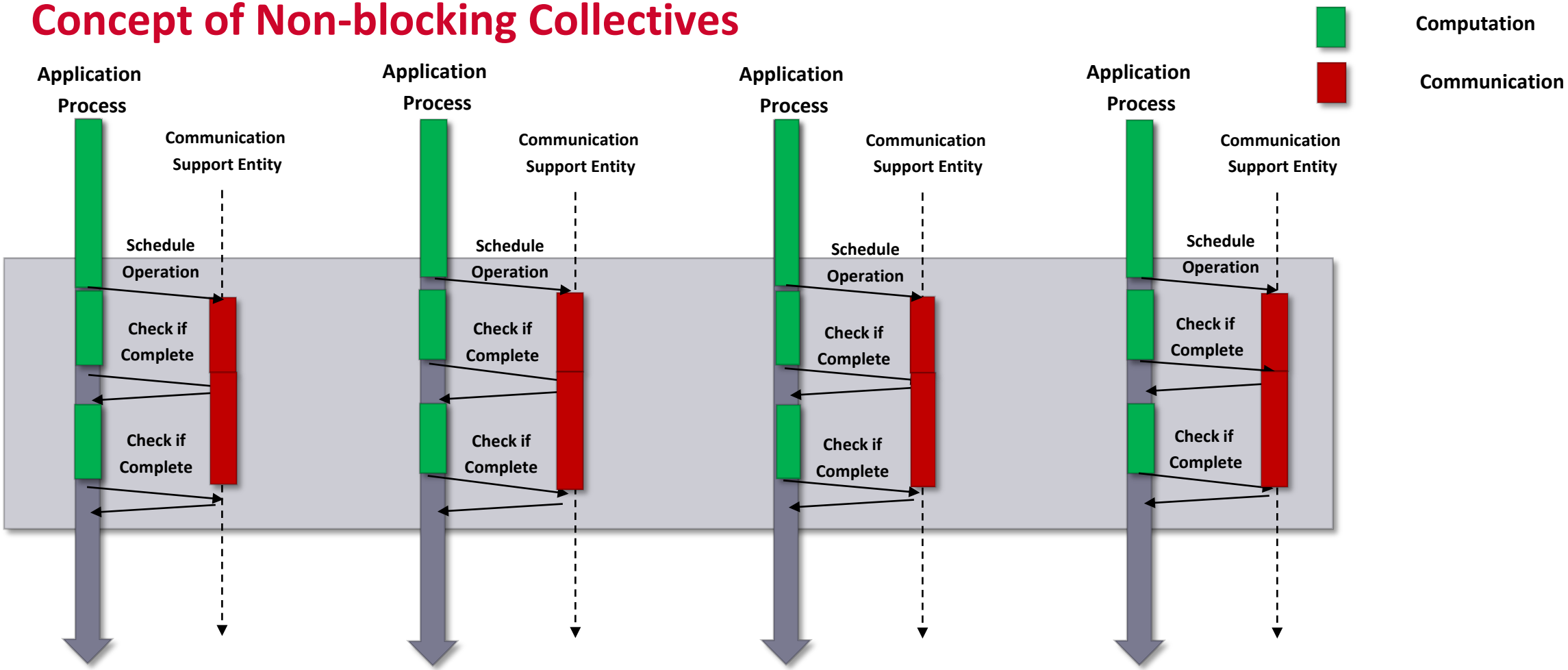
- **Refer to Running Collectives with Hardware based SHArP support section of MVAPICH2 user guide for more information**
- http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.3b-userguide.html#x1-990006.26

Problems with Blocking Collective Operations



- Communication time cannot be used for compute
  - No overlap of computation and communication
  - Inefficient

Concept of Non-blocking Collectives



- Application processes schedule collective operation
- Check periodically if operation is complete
- **Overlap of computation and communication => Better Performance**
- *Catch: Who will progress communication*

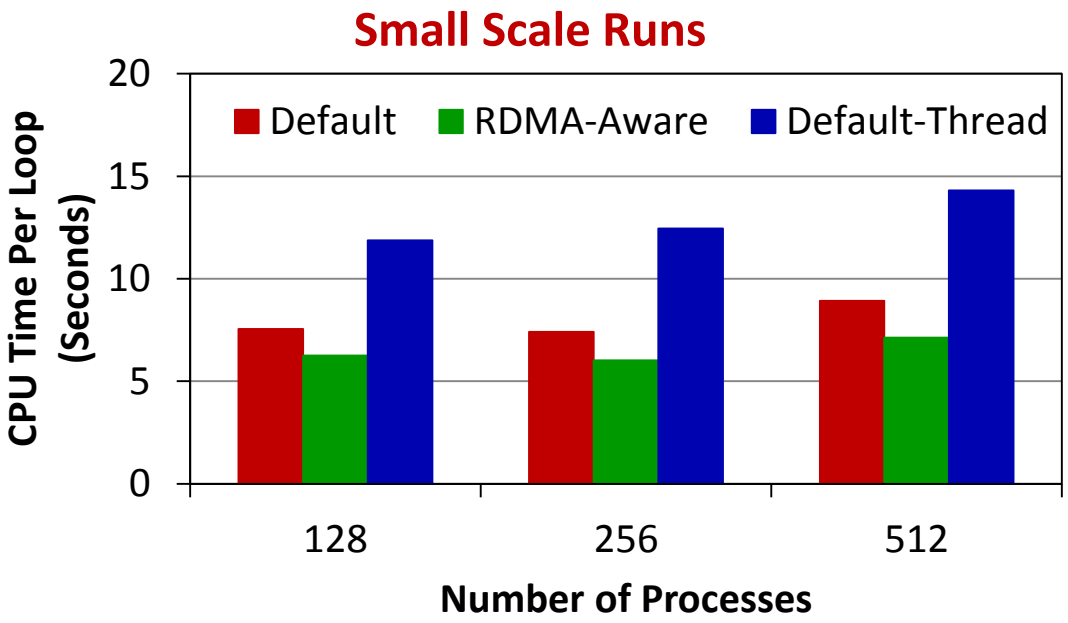
Non-blocking Collective (NBC) Operations

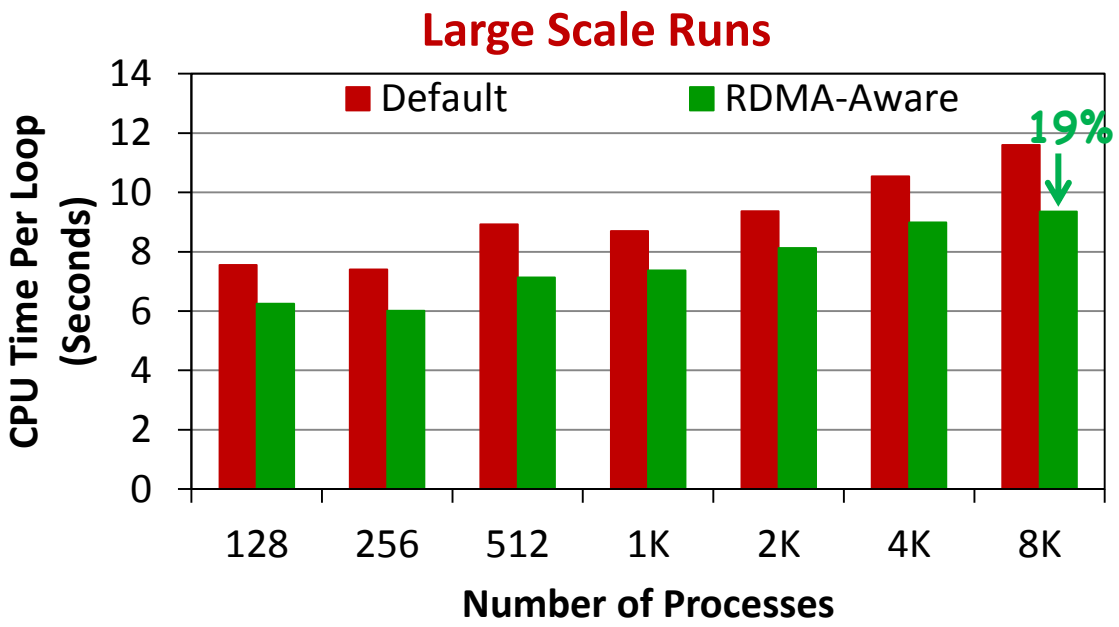
- Enables overlap of computation with communication
- Non-blocking calls do not match blocking collective calls
  - MPI may use different algorithms for blocking and non-blocking collectives
  - Blocking collectives: Optimized for latency
  - Non-blocking collectives: Optimized for overlap
- A process calling a NBC operation
  - Schedules collective operation and immediately returns
  - Executes application computation code
  - Waits for the end of the collective
- The communication progress by
  - Application code through MPI_Test
  - Network adapter (HCA) with hardware support
  - Dedicated processes / thread in MPI library
- There is a non-blocking equivalent for each blocking operation
  - Has an "I" in the name
    - MPI_Bcast -> MPI_Ibcast; MPI_Reduce -> MPI_Ireduce

How do I write applications with NBC?

```
void main()
{
    MPI_Init()
    …..
    MPI_Ialltoall(…)
    Computation that does not depend on result of Alltoall
    MPI_Test(for Ialltoall) /* Check if complete (non-blocking) */
    Computation that does not depend on result of Alltoall
    MPI_Wait(for Ialltoall) /* Wait till complete (Blocking) */
    …
    MPI_Finalize()
}
```

Performance of P3DFFT Kernel





- Weak scaling experiments; problem size increases with job size
- RDMA-Aware delivers 19% improvement over Default @ 8,192 processes
- Default-Thread exhibits worst performance
  - Possibly because threads steal CPU cycles from P3DFFT
  - Do not consider for large scale experiments
- Do not consider Default-Iprobe as it requires re-design of P3DFFT

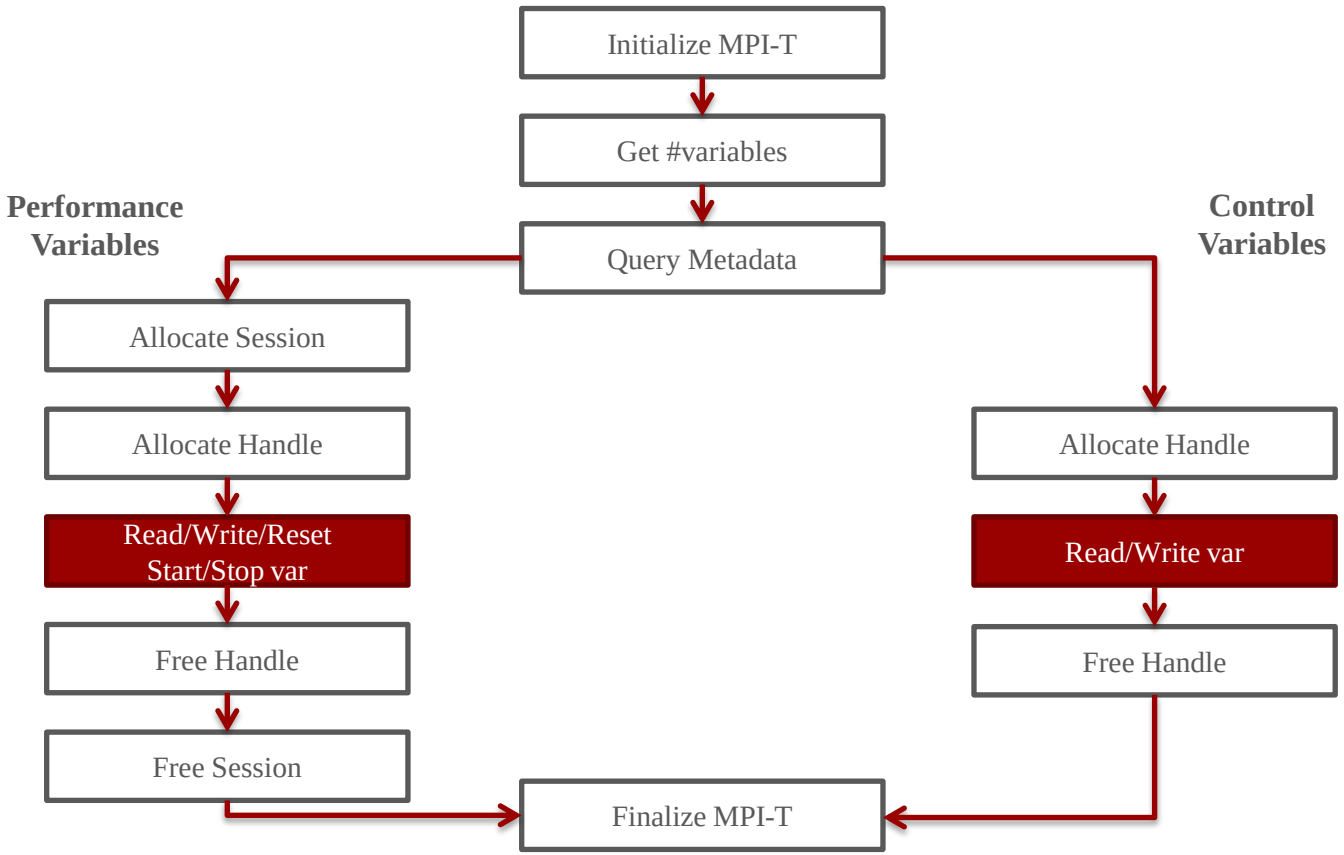
Presentation Overview

- Job start-up
- Point-to-point Inter-node Protocol
- Transport Type Selection
- Multi-rail and QoS
- Process Mapping and Point-to-point Intra-node Protocols
- Collectives
- **MPI_T Support**

MPI Tools Information Interface (MPI_T)

- Introduced in MPI 3.0 standard to expose internals of MPI to tools and applications
- Generalized interface – no defined variables in the standard
- Variables can differ between
  - MPI implementations
  - Compilations of same MPI library (production vs debug)
  - Executions of the same application/MPI library
  - There could be no variables provided
- Control Variables (CVARS) and Performance Variables (PVARS)
- More about the interface: mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf

MPI_T usage semantics




```
int MPI_T_pvar_start(MPI_T_pvar_session session, MPI_T_pvar_handle handle);
int MPI_T_pvar_read(MPI_T_pvar_session session, MPI_T_pvar_handle handle, void *buf);
int MPI_T_pvar_reset(MPI_T_pvar_session session, MPI_T_pvar_handle handle);
```

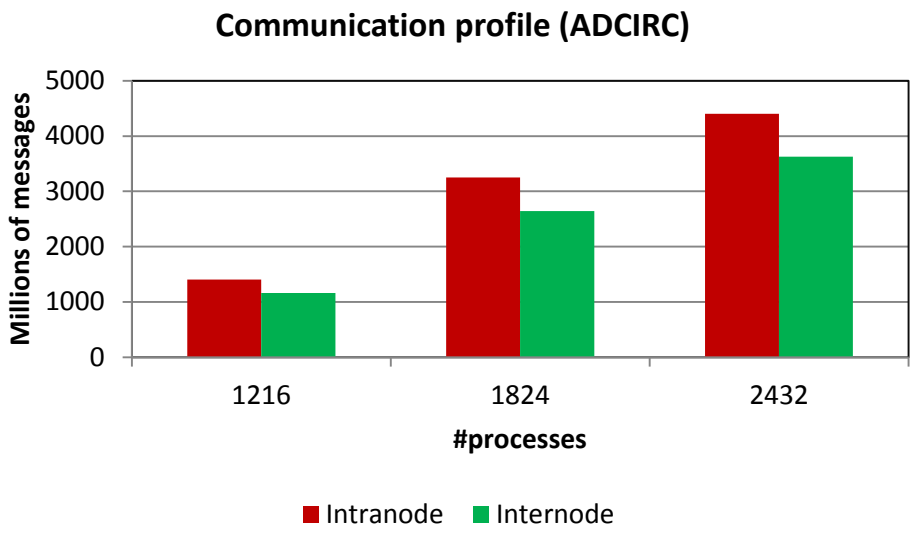
```
int MPI_T_cvar_get_info(int cvar_index, char *name, int *name_len, int *verbosity,
MPI_Datatype *datatype, MPI_T_enum *enumerator, int *binding, int *scope);
int MPI_T_pvar_handle_alloc(MPI_T_pvar_session session, int pvar_index,
void *obj_handle, MPI_T_pvar_handle *handle, int *count);
char *desc, int *desc_len, int *bind, int *scope);
```

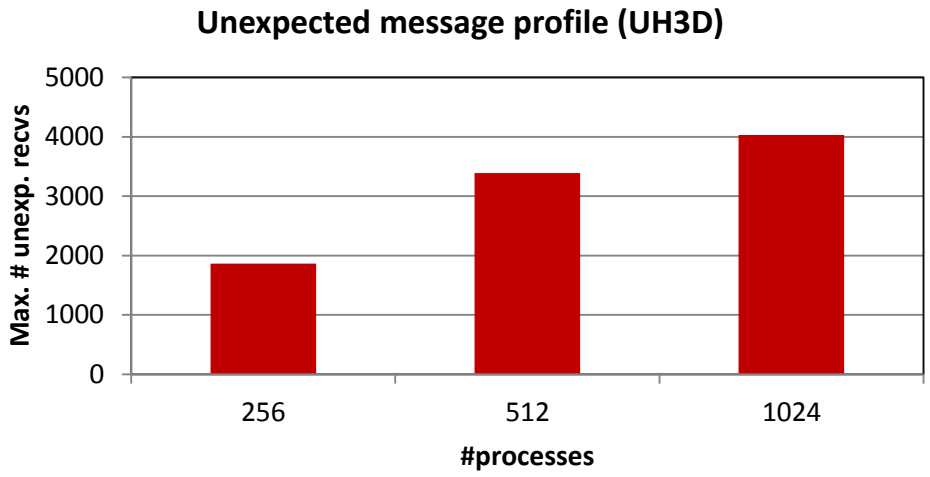
Co-designing Applications to use MPI-T

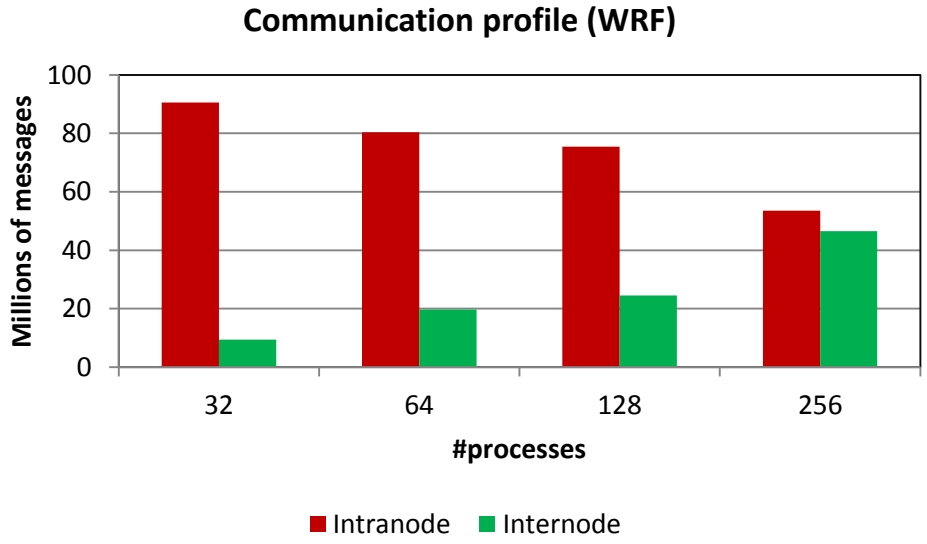
Example Pseudo-code: Optimizing the eager limit dynamically:

```
MPI_T_init_thread(..)
MPI_T_cvar_get_info(MV2_EAGER_THRESHOLD)
if (msg_size < MV2_EAGER_THRESHOLD + 1KB)
    MPI_T_cvar_write(MV2_EAGER_THRESHOLD, +1024)
MPI_Send(..)
MPI_T_finalize(..)
```

Evaluating Applications with MPI-T



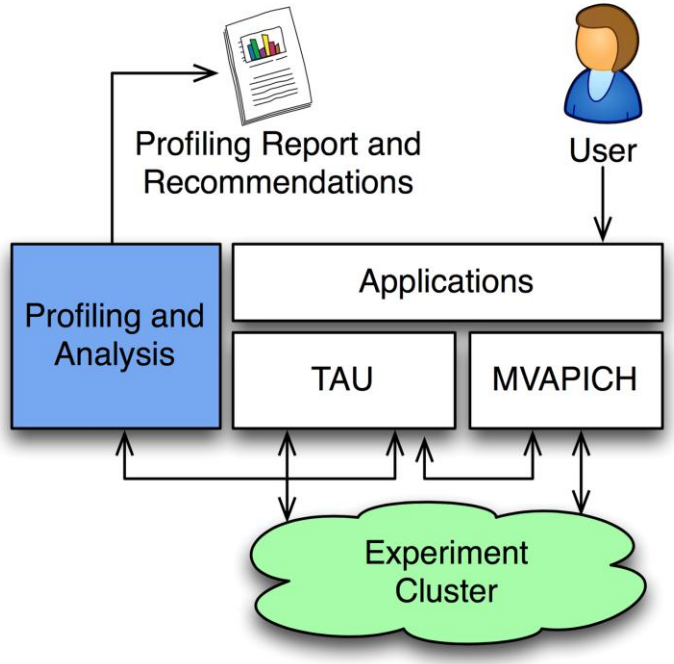




- **Users can gain insights into application communication characteristics!**

Performance Engineering Applications using MVAPICH2 and TAU

- Enhance existing support for MPI_T in MVAPICH2 to expose a richer set of performance and control variables
- Get and display MPI Performance Variables (PVARs) made available by the runtime in TAU
- Control the runtime's behavior via MPI Control Variables (CVARs)
- Introduced support for new MPI_T based CVARs to MVAPICH2
  - MPIR_CVAR_MAX_INLINE_MSG_SZ, MPIR_CVAR_VBUF_POOL_SIZE, MPIR_CVAR_VBUF_SECONDARY_POOL_SIZE
- TAU enhanced with support for setting MPI_T CVARs in a non-interactive mode for uninstrumented applications
- S. Ramesh, A. Maheo, S. Shende, A. Malony, H. Subramoni, and D. K. Panda, *MPI Performance Engineering with the MPI Tool Interface: the Integration of MVAPICH and TAU, EuroMPI/USA '17, Best Paper Finalist*
- More details in Prof. Malony's talk today and poster presentations



VBUF usage without CVAR based tuning as displayed by ParaProf

TAU: ParaProf: Context Events for: node 0 - mpit_withoutcvar_bt.C.1k.ppk

Name	MaxValue	MinValue	MeanValue	Std. Dev.	NumSamples	Total
mv2_total_vbuf_memory (Total amount of memory in bytes used for VBUFs)	3,313,056	3,313,056	3,313,056	0	1	3,313,056
mv2_ud_vbuf_allocated (Number of UD VBUFs allocated)	0	0	0	0	0	0
mv2_ud_vbuf_available (Number of UD VBUFs available)	0	0	0	0	0	0
mv2_ud_vbuf_freed (Number of UD VBUFs freed)	0	0	0	0	0	0
mv2_ud_vbuf_inuse (Number of UD VBUFs inuse)	0	0	0	0	0	0
mv2_ud_vbuf_max_use (Maximum number of UD VBUFs used)	0	0	0	0	0	0
mv2_vbuf_allocated (Number of VBUFs allocated)	320	320	320	0	1	320
mv2_vbuf_available (Number of VBUFs available)	255	255	255	0	1	255
mv2_vbuf_freed (Number of VBUFs freed)	25,545	25,545	25,545	0	1	25,545
mv2_vbuf_inuse (Number of VBUFs inuse)	65	65	65	0	1	65
mv2_vbuf_max_use (Maximum number of VBUFs used)	65	65	65	0	1	65
num_calloc_calls (Number of MPIT_calloc calls)	89	89	89	0	1	89

VBUF usage with CVAR based tuning as displayed by ParaProf

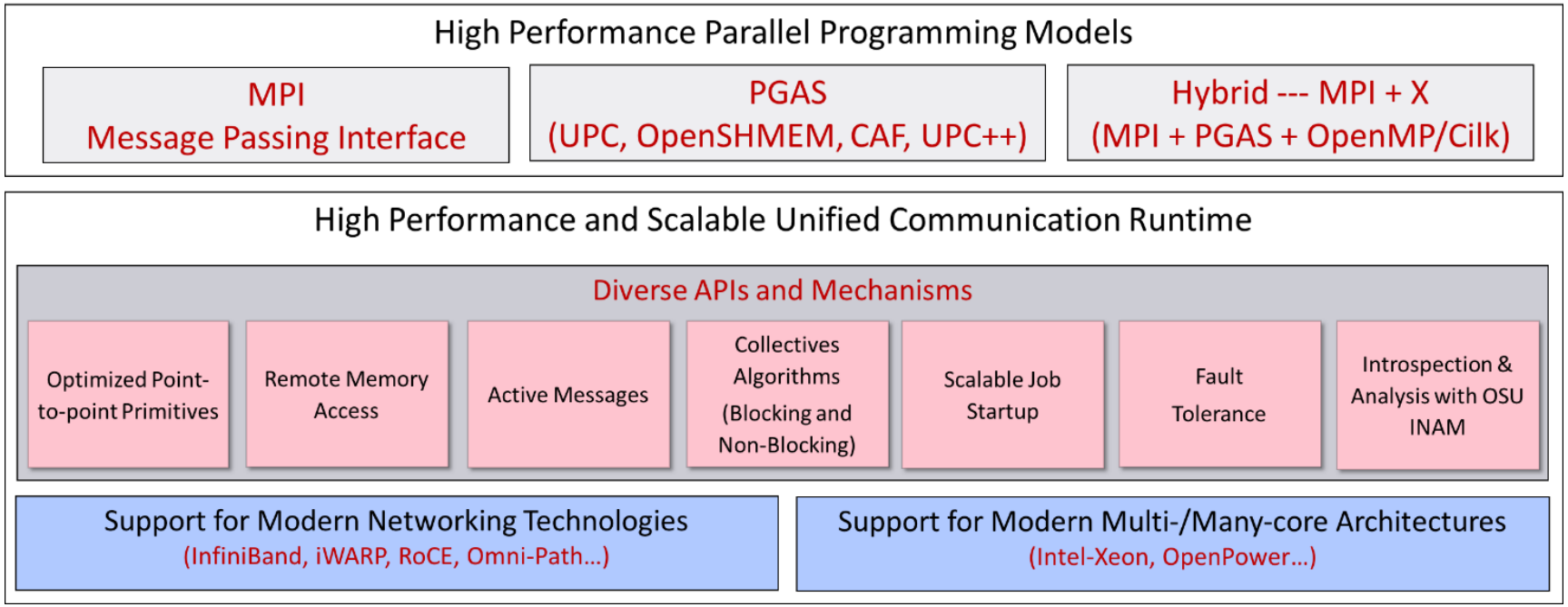
TAU: ParaProf: Context Events for: node 0 - bt-mz.E.vbuf_pool_16.1k.ppk

Name	MaxValue	MinValue	MeanValue	Std. Dev.	NumSamp...	Total
mv2_total_vbuf_memory (Total amount of memory in bytes used for VBUFs)	1,815,056	1,815,056	1,815,056	0	1	1,815,056
mv2_ud_vbuf_allocated (Number of UD VBUFs allocated)	0	0	0	0	0	0
mv2_ud_vbuf_available (Number of UD VBUFs available)	0	0	0	0	0	0
mv2_ud_vbuf_freed (Number of UD VBUFs freed)	0	0	0	0	0	0
mv2_ud_vbuf_inuse (Number of UD VBUFs inuse)	0	0	0	0	0	0
mv2_ud_vbuf_max_use (Maximum number of UD VBUFs used)	0	0	0	0	0	0
mv2_vbuf_allocated (Number of VBUFs allocated)	160	160	160	0	1	160
mv2_vbuf_available (Number of VBUFs available)	94	94	94	0	1	94
mv2_vbuf_freed (Number of VBUFs freed)	5,479	5,479	5,479	0	1	5,479
mv2_vbuf_inuse (Number of VBUFs inuse)	66	66	66	0	1	66

MVAPICH2 Software Family

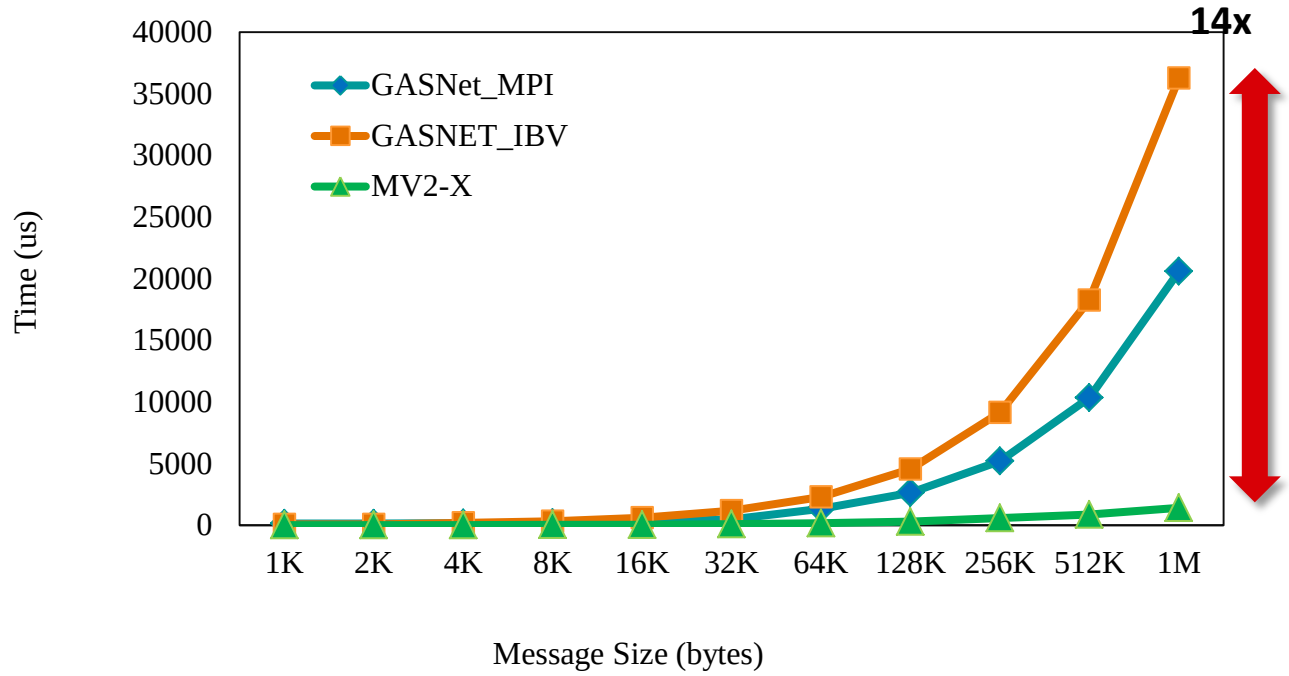
Requirements	Library
MPI with IB, iWARP and RoCE	MVAPICH2
Advanced MPI, OSU INAM, PGAS and MPI+PGAS with IB and RoCE	MVAPICH2-X
MPI with IB & GPU	MVAPICH2-GDR
MPI with IB & MIC	MVAPICH2-MIC
HPC Cloud with MPI & IB	MVAPICH2-Virt
Energy-aware MPI with IB, iWARP and RoCE	MVAPICH2-EA
MPI Energy Monitoring Tool	OEMT
InfiniBand Network Analysis and Monitoring	OSU INAM
Microbenchmarks for Measuring MPI and PGAS Performance	OMB

MVAPICH2-X for Hybrid MPI + PGAS Applications

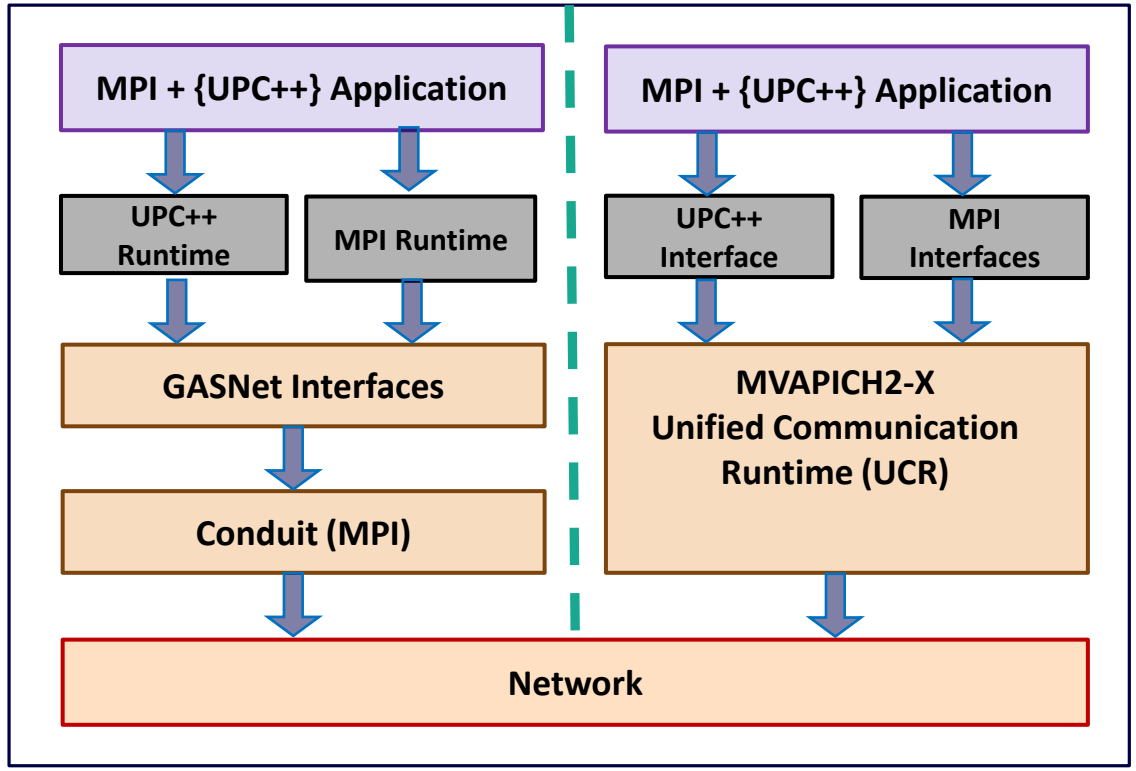


- Current Model – Separate Runtimes for OpenSHMEM/UPC/UPC++/CAF and MPI
  - Possible deadlock if both runtimes are not progressed
  - Consumes more network resource
- Unified communication runtime for MPI, UPC, UPC++, OpenSHMEM, CAF
  - Available with since 2012 (starting with MVAPICH2-X 1.9)
  - http://mvapich.cse.ohio-state.edu

UPC++ Support in MVAPICH2-X



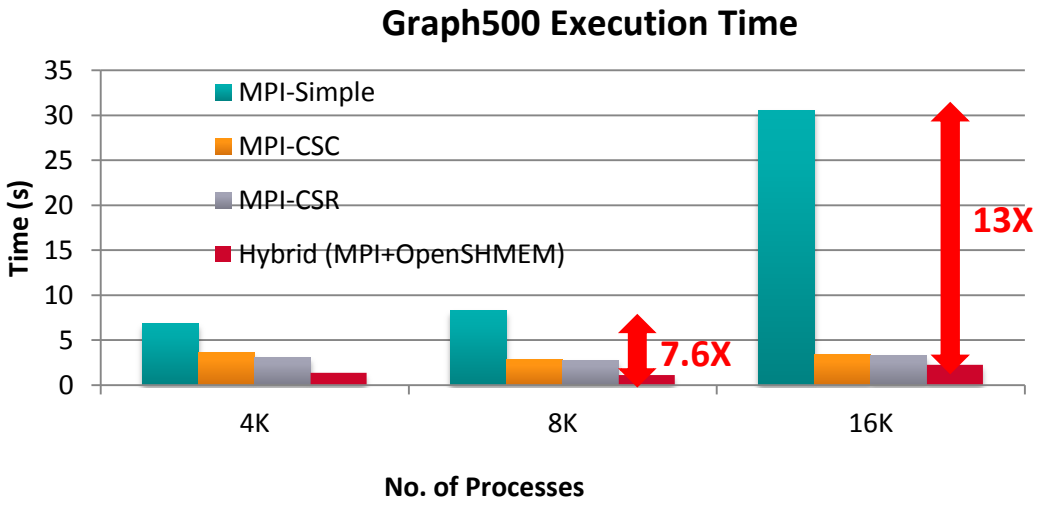
Inter-node Broadcast (64 nodes 1:ppn)

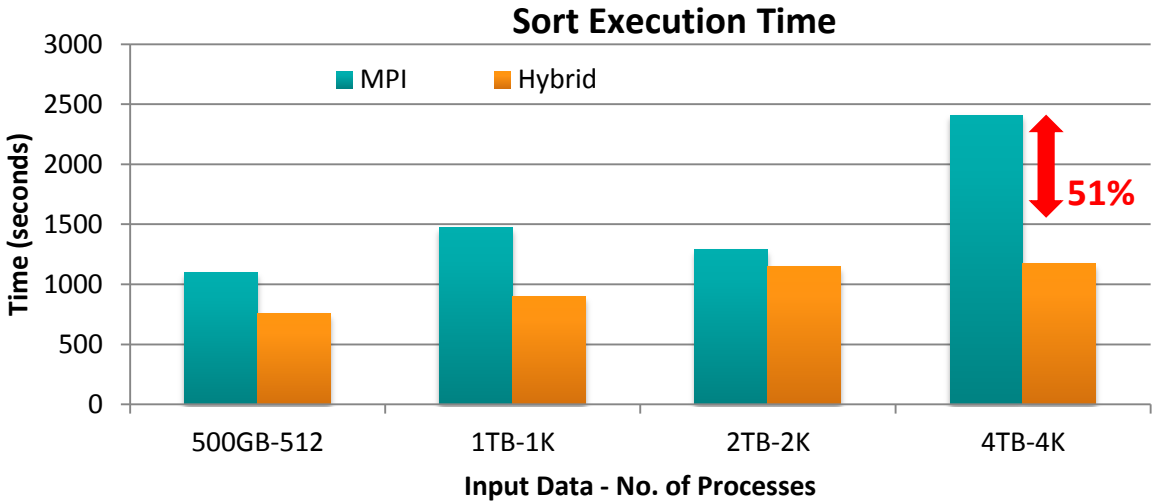


- Full and native support for hybrid MPI + UPC++ applications
- Better performance compared to IBV and MPI conduits
- OSU Micro-benchmarks (OMB) support for UPC++
- Available since MVAPICH2-X (2.2rc1)

More Details in Student Poster Presentation

Application Level Performance with Graph500 and Sort





- Performance of Hybrid (MPI+ OpenSHMEM) Graph500 Design
  - 8,192 processes
    - **2.4X** improvement over MPI-CSR
    - **7.6X** improvement over MPI-Simple
  - 16,384 processes
    - **1.5X** improvement over MPI-CSR
    - **13X** improvement over MPI-Simple
- Performance of Hybrid (MPI+OpenSHMEM) Sort Application
  - 4,096 processes, 4 TB Input Size
    - MPI – 2408 sec; 0.16 TB/min
    - Hybrid – 1172 sec; 0.36 TB/min
    - **51%** improvement over MPI-design

J. Jose, K. Kandalla, S. Potluri, J. Zhang and D. K. Panda, Optimizing Collective Communication in OpenSHMEM, Int'l Conference on Partitioned Global Address Space Programming Models (PGAS '13), October 2013.

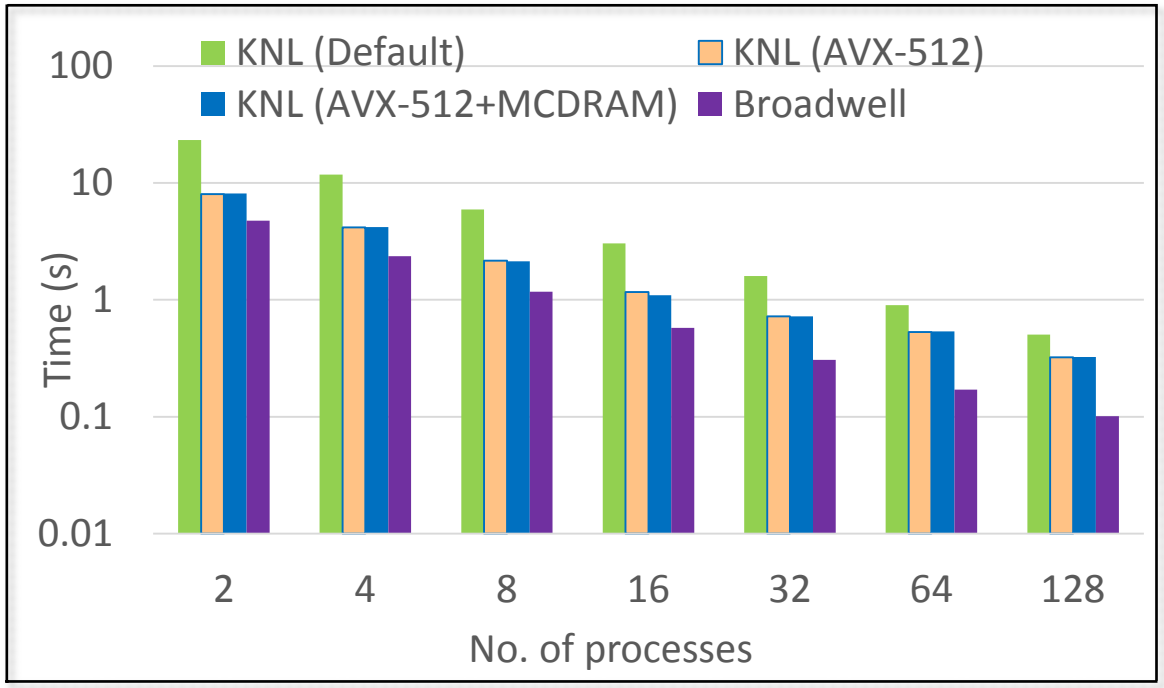
J. Jose, S. Potluri, K. Tomko and D. K. Panda, Designing Scalable Graph500 Benchmark with Hybrid MPI+OpenSHMEM Programming Models, International Supercomputing Conference (ISC'13), June 2013

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

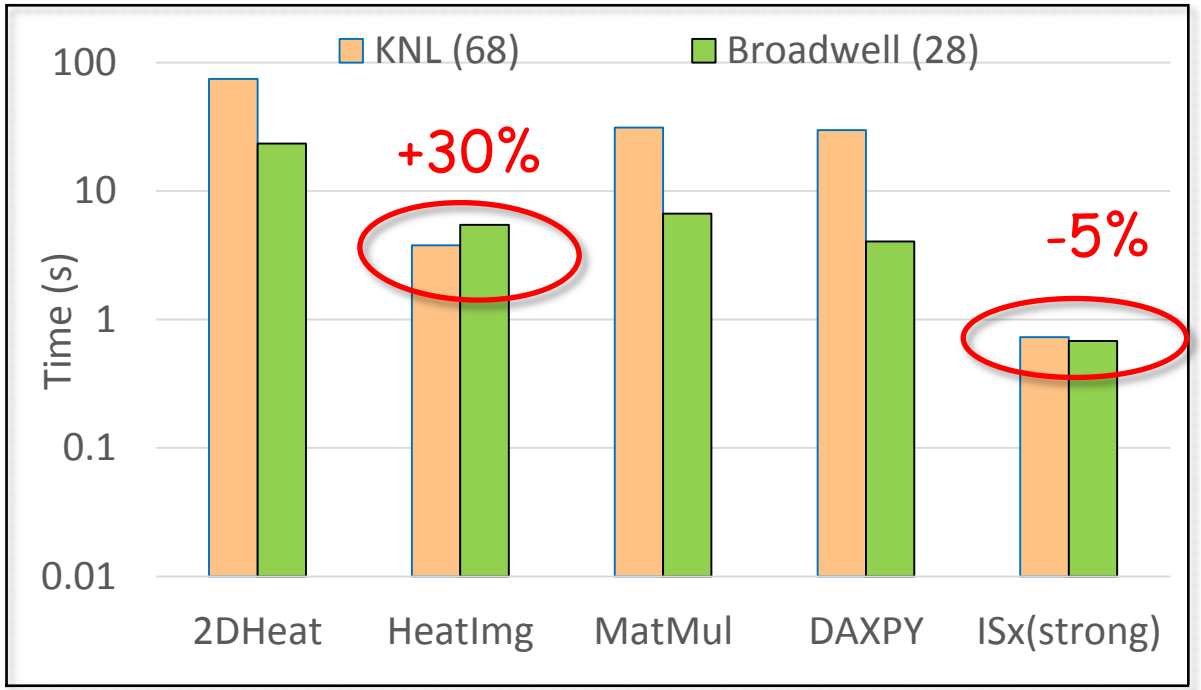
OpenSHMEM Application kernels

- AVX-512 vectorization and MCDRAM based optimizations for MVAPICH2-X
  - Will be available in future MVAPICH2-X release

Scalable Integer Sort Kernel (ISx)



Single KNL and Single Broadwell node

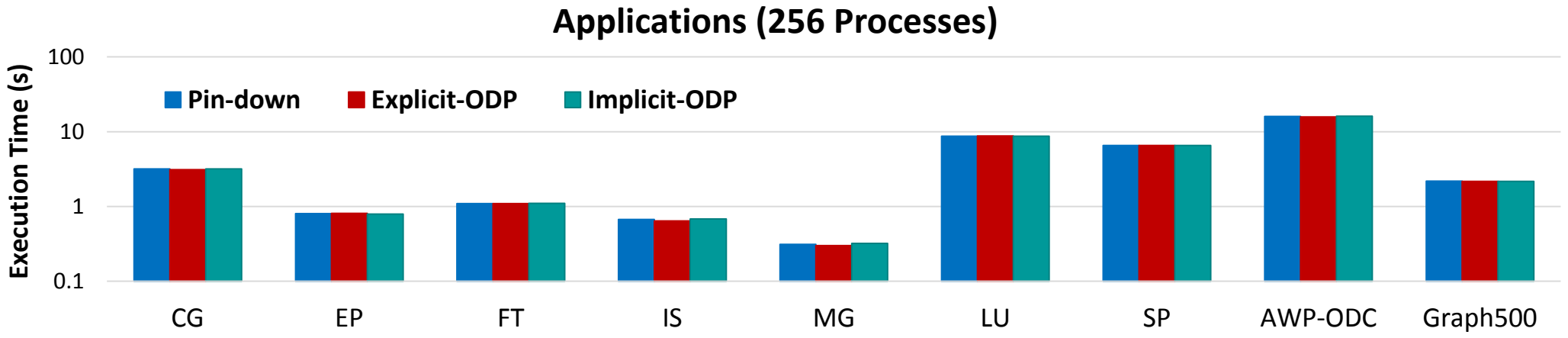


- AVX-512 vectorization showed up to 3X improvement over default
- KNL showed on-par performance as Broadwell on Node-by-node comparison

Exploiting and Evaluating OpenSHMEM on KNL Architecture J. Hashmi, M. Li, H. Subramoni, D. Panda Fourth Workshop on OpenSHMEM and Related Technologies, Aug 2017.

Implicit On-Demand Paging (ODP)

- Introduced by Mellanox to avoid pinning the pages of registered memory regions
- ODP-aware runtime could reduce the size of pin-down buffers while maintaining performance



M. Li, X. Lu, H. Subramoni, and D. K. Panda, "Designing Registration Caching Free High-Performance MPI Library with Implicit On-Demand Paging (ODP) of InfiniBand", Under Review

MVAPICH2 Software Family

Requirements	Library
MPI with IB, iWARP and RoCE	MVAPICH2
Advanced MPI, OSU INAM, PGAS and MPI+PGAS with IB and RoCE	MVAPICH2-X
MPI with IB & GPU	MVAPICH2-GDR
MPI with IB & MIC	MVAPICH2-MIC
HPC Cloud with MPI & IB	MVAPICH2-Virt
Energy-aware MPI with IB, iWARP and RoCE	MVAPICH2-EA
MPI Energy Monitoring Tool	OEMT
InfiniBand Network Analysis and Monitoring	OSU INAM
Microbenchmarks for Measuring MPI and PGAS Performance	OMB

CUDA-Aware MPI: MVAPICH2-GDR 1.8-2.2 Releases

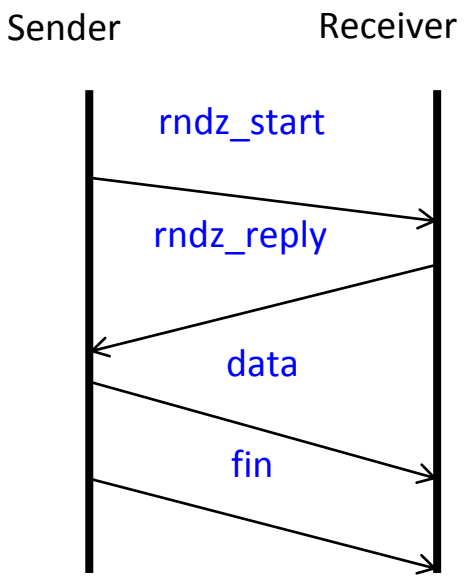
- Support for MPI communication from NVIDIA GPU device memory
- High performance RDMA-based inter-node point-to-point communication (GPU-GPU, GPU-Host and Host-GPU)
- High performance intra-node point-to-point communication for multi-GPU adapters/node (GPU-GPU, GPU-Host and Host-GPU)
- Taking advantage of CUDA IPC (available since CUDA 4.1) in intra-node communication for multiple GPU adapters/node
- Optimized and tuned collectives for GPU device buffers
- MPI datatype support for point-to-point and collective communication from GPU device buffers

Presentation Overview

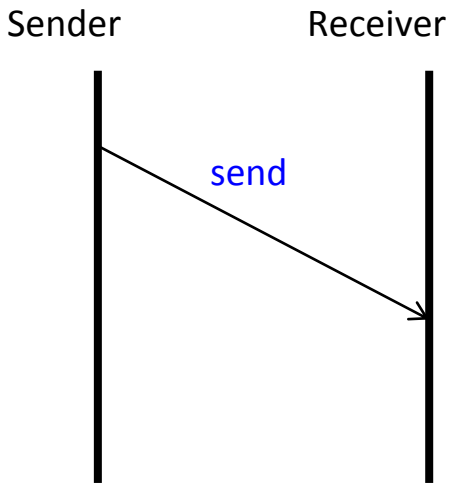
- **Support for Efficient Small Message Communication with GPUDirect RDMA**
- Multi-rail Support
- Support for Efficient Intra-node Communication using CUDA IPC
- MPI Datatype Support
- On-going Work

Enhanced MPI Design with GPUDirect RDMA

- Current MPI design using GPUDirect RDMA uses Rendezvous protocol
  - Has higher latency for small messages
- Can eager protocol be supported to improve performance for small messages?
- Two schemes proposed and used
  - Loopback (using network adapter to copy data)
  - Fastcopy/GDRCOPY (using CPU to copy data)



Rendezvous Protocol

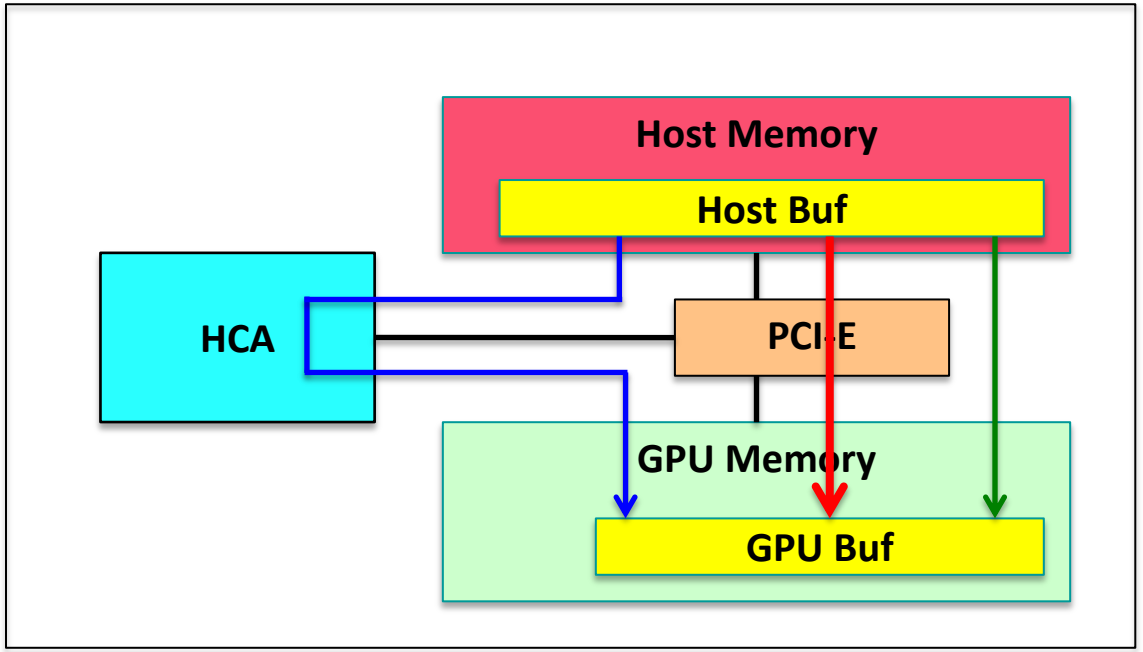


Eager Protocol

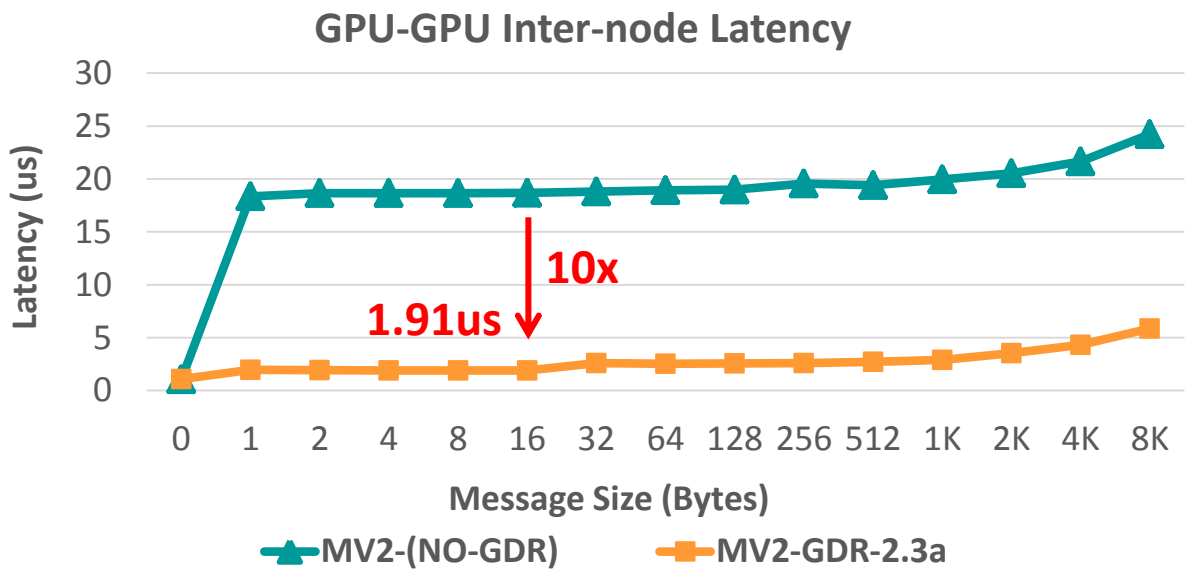
R. Shi, S. Potluri, K. Hamidouche M. Li, J. Perkins D. Rossetti and D. K. Panda, Designing Efficient Small Message Transfer Mechanism for Inter-node MPI Communication on InfiniBand GPU Clusters IEEE International Conference on High Performance Computing (HiPC'2014)

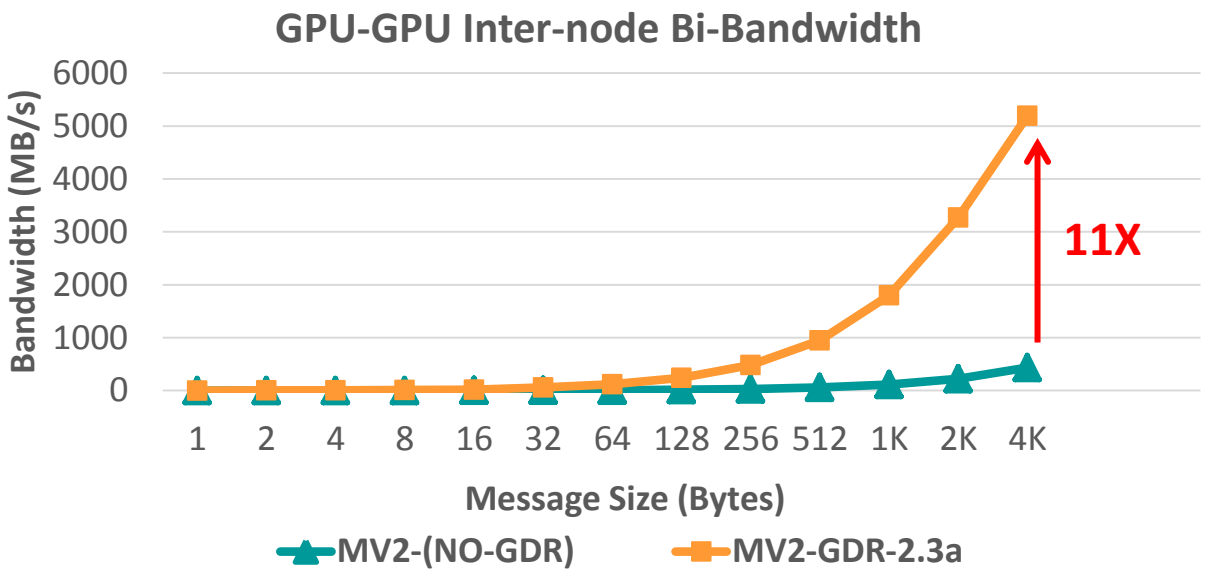
Alternative Approaches

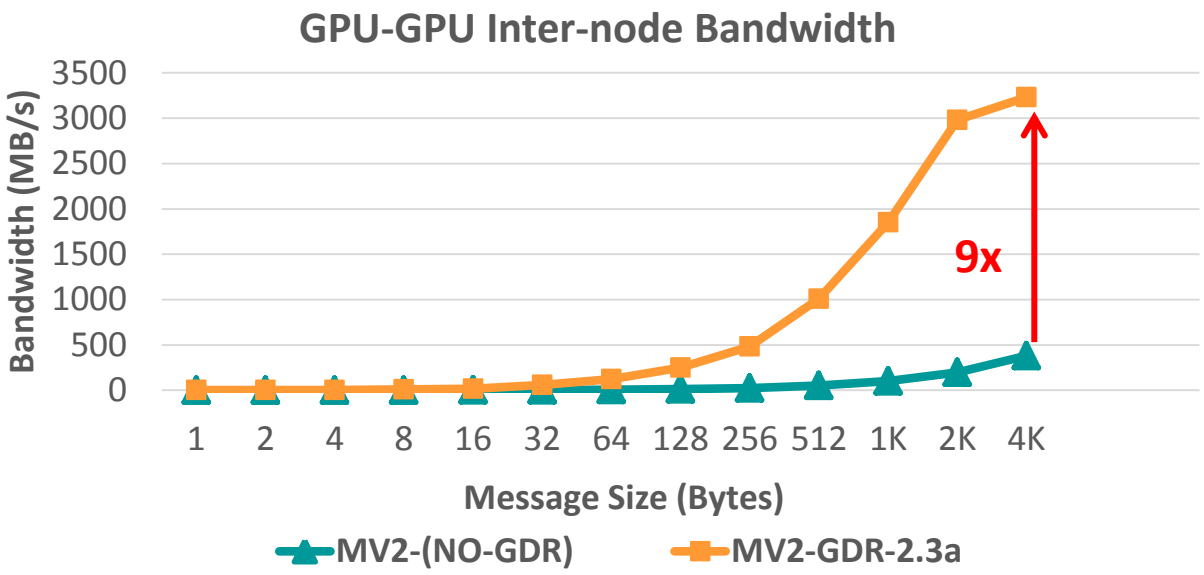
- **Can we substitute the cudaMemcpy with a better design?**
  - CudaMemcpy: Default Scheme
    - Big overhead for small message
  - Loopback-based design: Uses GDR feature
    - Process establishes self-connection
    - Copy H-D ⇒ RDMA write (H, D)
    - Copy D-H ⇒ RDMA write (D, H)
    - P2P bottleneck ⇒ good for small and medium sizes
  - GDRCOPY-based design: New module for fast copies
    - Involves GPU PCIe BAR1 mapping
    - CPU performing the copy ⇒ block until completion
    - Very good performance for H-D for small and medium sizes
    - Very good performance for D-H only for very small sizes



Performance of MVAPICH2-GPU with GPU-Direct RDMA (GDR)

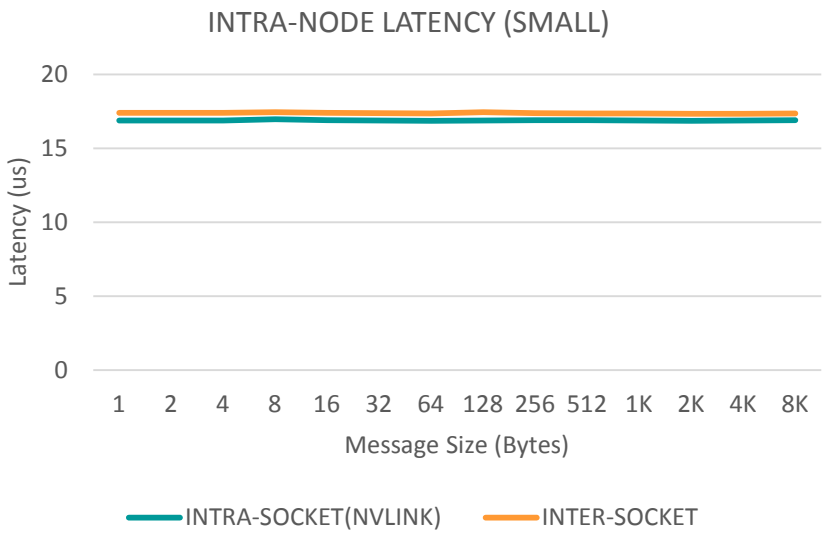


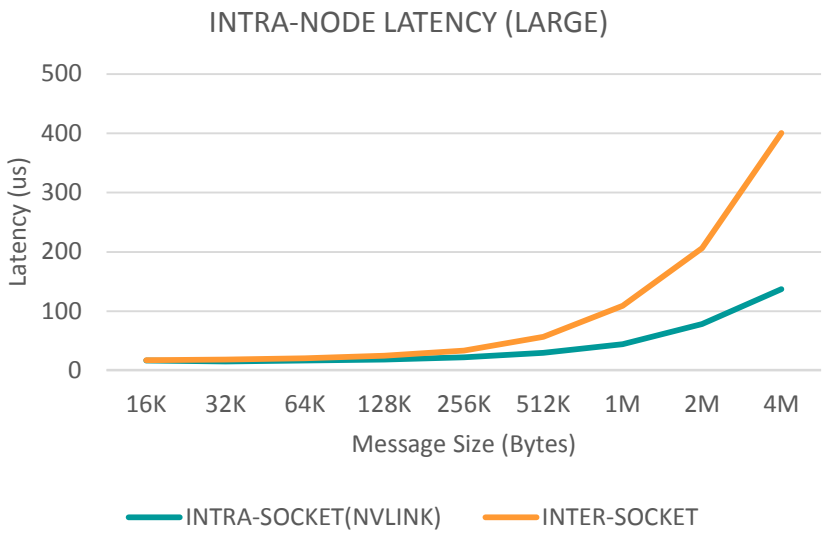


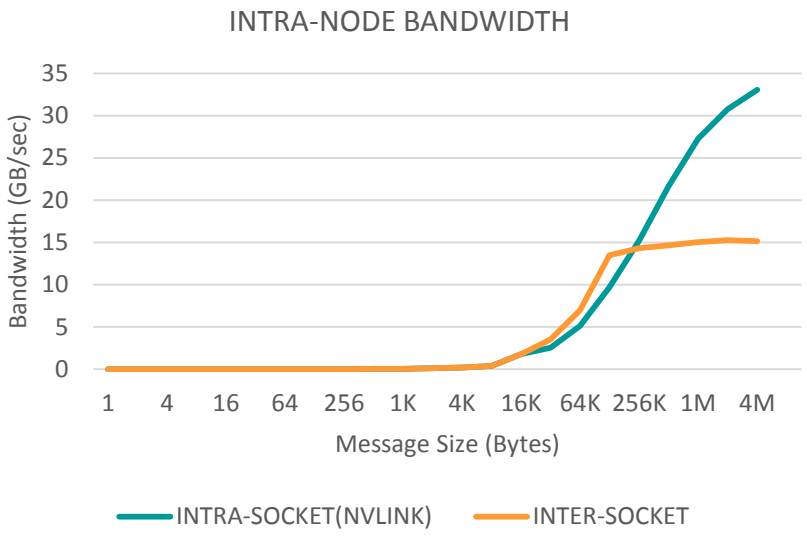


MVAPICH2-GDR-2.3a
Intel Haswell (E5-2687W) node - 20 cores
NVIDIA Pascal P100 GPU
Mellanox Connect-X5 EDR HCA
CUDA 8.0
Mellanox OFED 4.0 with GPU-Direct-RDMA

MVAPICH2-GDR: Performance on OpenPOWER (NVLink + Pascal)

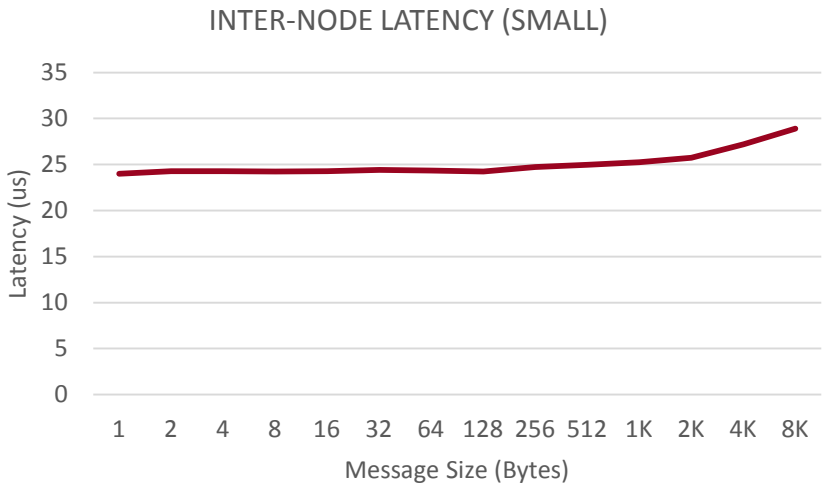


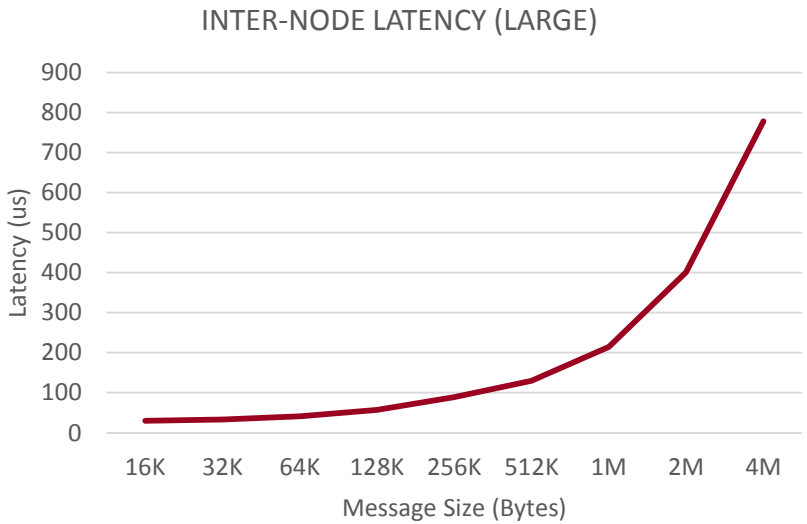


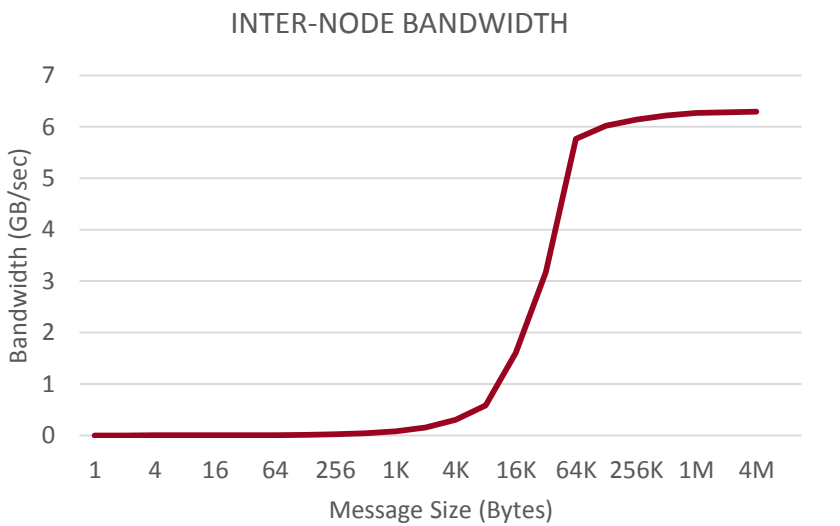


Intra-node Latency: 16.8 us (without GPUDirectRDMA)

Intra-node Bandwidth: 32 GB/sec (NVLINK)







Inter-node Latency: 22 us (without GPUDirectRDMA)

Will be available in upcoming MVAPICH2-GDR

Inter-node Bandwidth: 6 GB/sec (FDR)

Platform: OpenPOWER (ppc64le) nodes equipped with a dual-socket CPU, 4 Pascal P100-SXM GPUs, and 4X-FDR InfiniBand Inter-connect

Tuning GDRCOPY Designs in MVAPICH2-GDR

Parameter	Significance	Default	Notes
MV2_USE_GPUDIRECT_GDRCOPY	• Enable / Disable GDRCOPY-based designs	1 (Enabled)	• Always enable
MV2_GPUDIRECT_GDRCOPY_LIMIT	• Controls messages size until which GDRCOPY is used	8 KByte	• Tune for your system • GPU type, host architecture. Impacts the eager performance
MV2_GPUDIRECT_GDRCOPY_LIB	• Path to the GDRCOPY library	Unset	• Always set
MV2_USE_GPUDIRECT_D2H_GDRCOPY_LIMIT	• Controls messages size until which GDRCOPY is used at sender	16Bytes	• Tune for your systems • CPU and GPU type

- **Refer to Tuning and Usage Parameters section of MVAPICH2-GDR user guide for more information**
- http://mvapich.cse.ohio-state.edu/userguide/gdr/#_tuning_and_usage_parameters

Tuning Loopback Designs in MVAPICH2-GDR

Parameter	Significance	Default	Notes
MV2_USE_GPUDIRECT_LOOPBACK	• Enable / Disable LOOPBACK-based designs	1 (Enabled)	• Always enable
MV2_GPUDIRECT_LOOPBACK_LIMIT	• Controls messages size until which LOOPBACK is used	8 KByte	• Tune for your system • GPU type, host architecture and HCA. Impacts the eager performance •Sensitive to the P2P issue

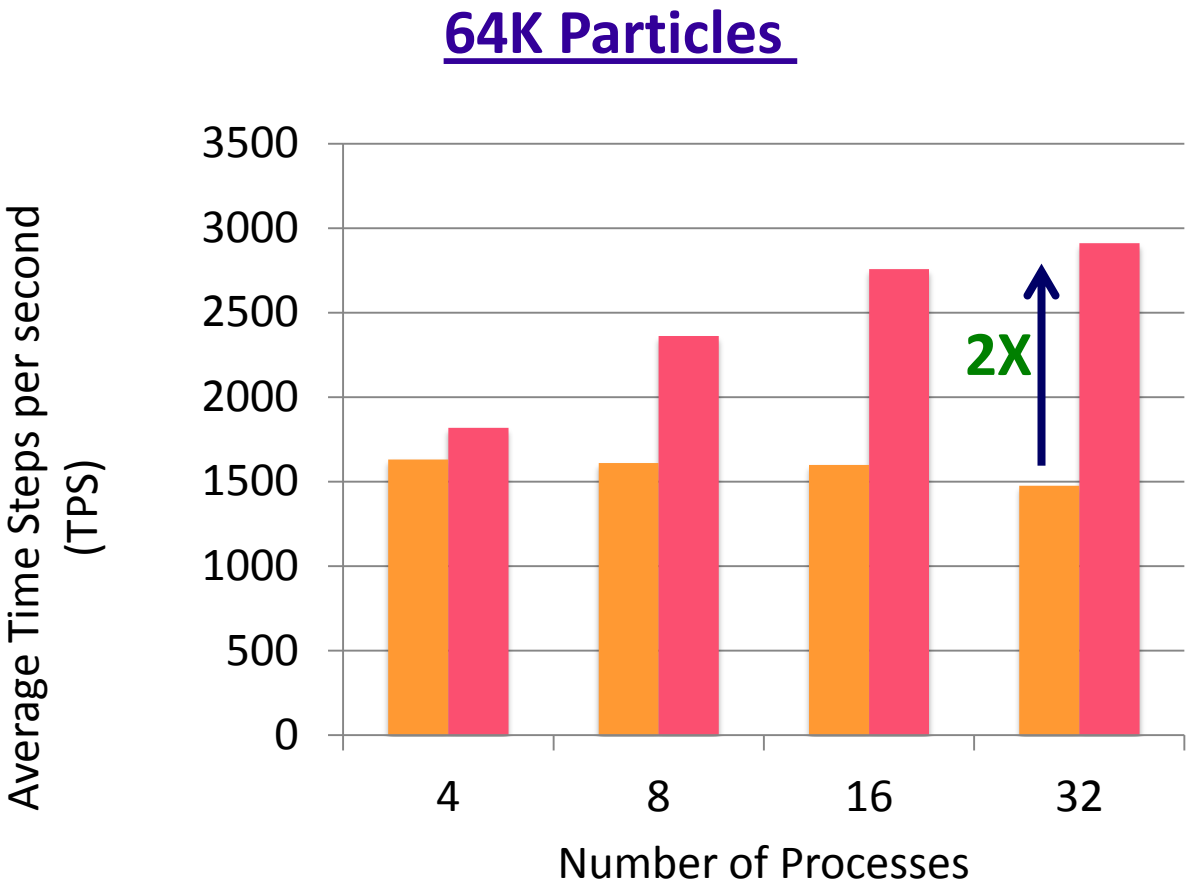
- **Refer to Tuning and Usage Parameters section of MVAPICH2-GDR user guide for more information**
- http://mvapich.cse.ohio-state.edu/userguide/gdr/#_tuning_and_usage_parameters

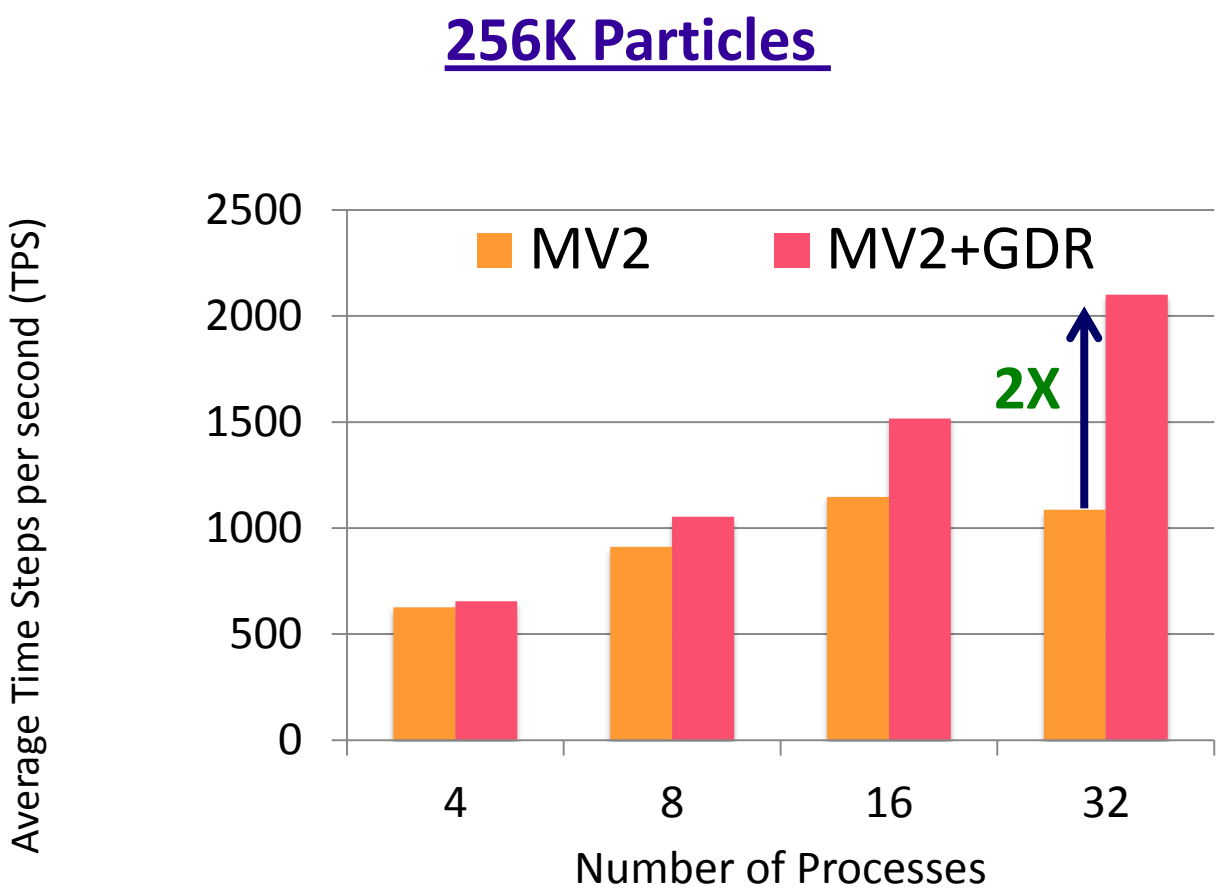
Tuning GPUDirect RDMA (GDR) Designs in MVAPICH2-GDR

Parameter	Significance	Default	Notes
MV2_USE_GPUDIRECT	• Enable / Disable GDR-based designs	1 (Enabled)	• Always enable
MV2_GPUDIRECT_LIMIT	• Controls messages size until which GPUDirect RDMA is used	8 KByte	• Tune for your system • GPU type, host architecture and CUDA version: impact pipelining overheads and P2P bandwidth bottlenecks
MV2_USE_GPUDIRECT_RECEIVE_LIMIT	• Controls messages size until which 1 hop design is used (GDR Write at the receiver)	256KBytes	• Tune for your system • GPU type, HCA type and configuration

- **Refer to Tuning and Usage Parameters section of MVAPICH2-GDR user guide for more information**
- http://mvapich.cse.ohio-state.edu/userguide/gdr/#_tuning_and_usage_parameters

Application-Level Evaluation (HOOMD-blue)





- Platform: Wilkes (Intel Ivy Bridge + NVIDIA Tesla K20c + Mellanox Connect-IB)
- HoomdBlue Version 1.0.5
  - GDRCOPY enabled: MV2_USE_CUDA=1 MV2_IBA_HCA=mlx5_0 MV2_IBA_EAGER_THRESHOLD=32768 MV2_VBUF_TOTAL_SIZE=32768 MV2_USE_GPUDIRECT_LOOPBACK_LIMIT=32768 MV2_USE_GPUDIRECT_GDRCOPY=1 MV2_USE_GPUDIRECT_GDRCOPY_LIMIT=16384

Presentation Overview

- Support for Efficient Small Message Communication with GPUDirect RDMA
- **Multi-rail Support**
- Support for Efficient Intra-node Communication using CUDA IPC
- MPI Datatype Support
- On-going Work

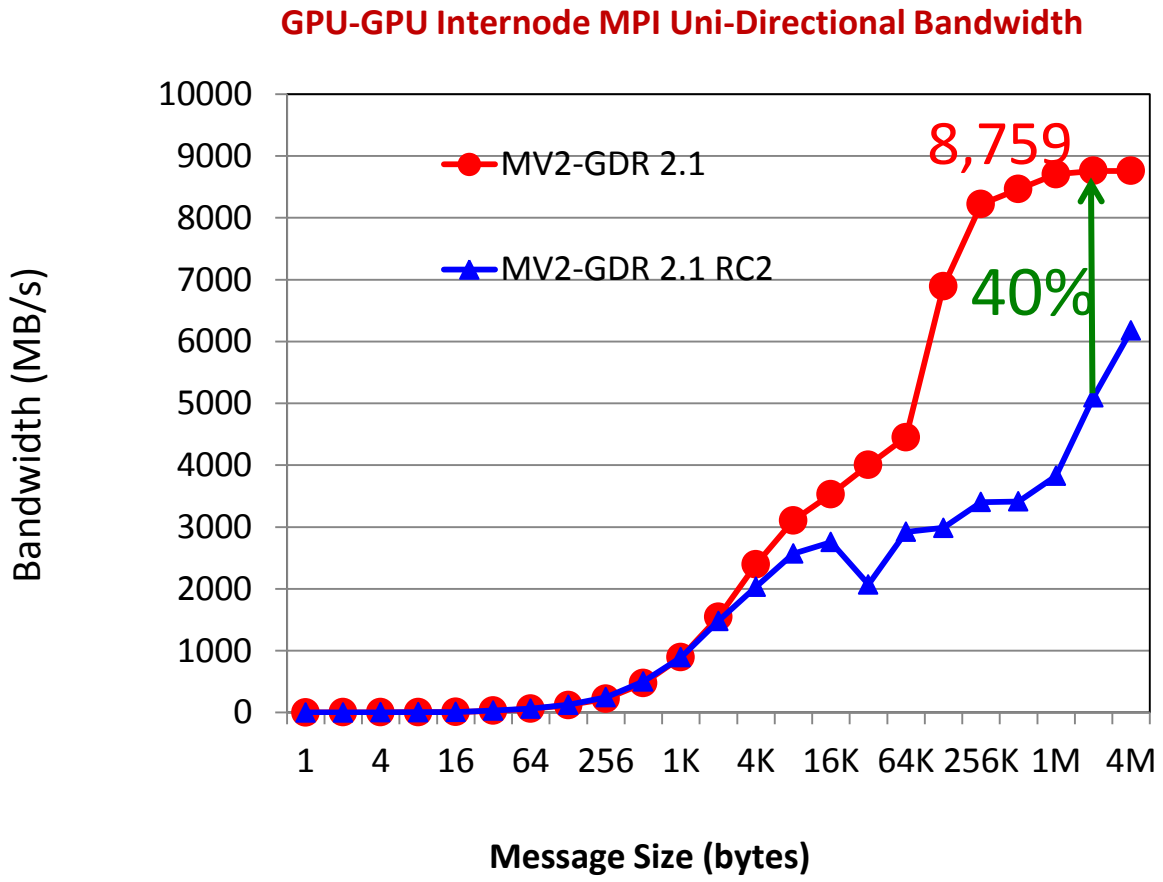
Tuning Multi-rail Support in MVAPICH2-GDR

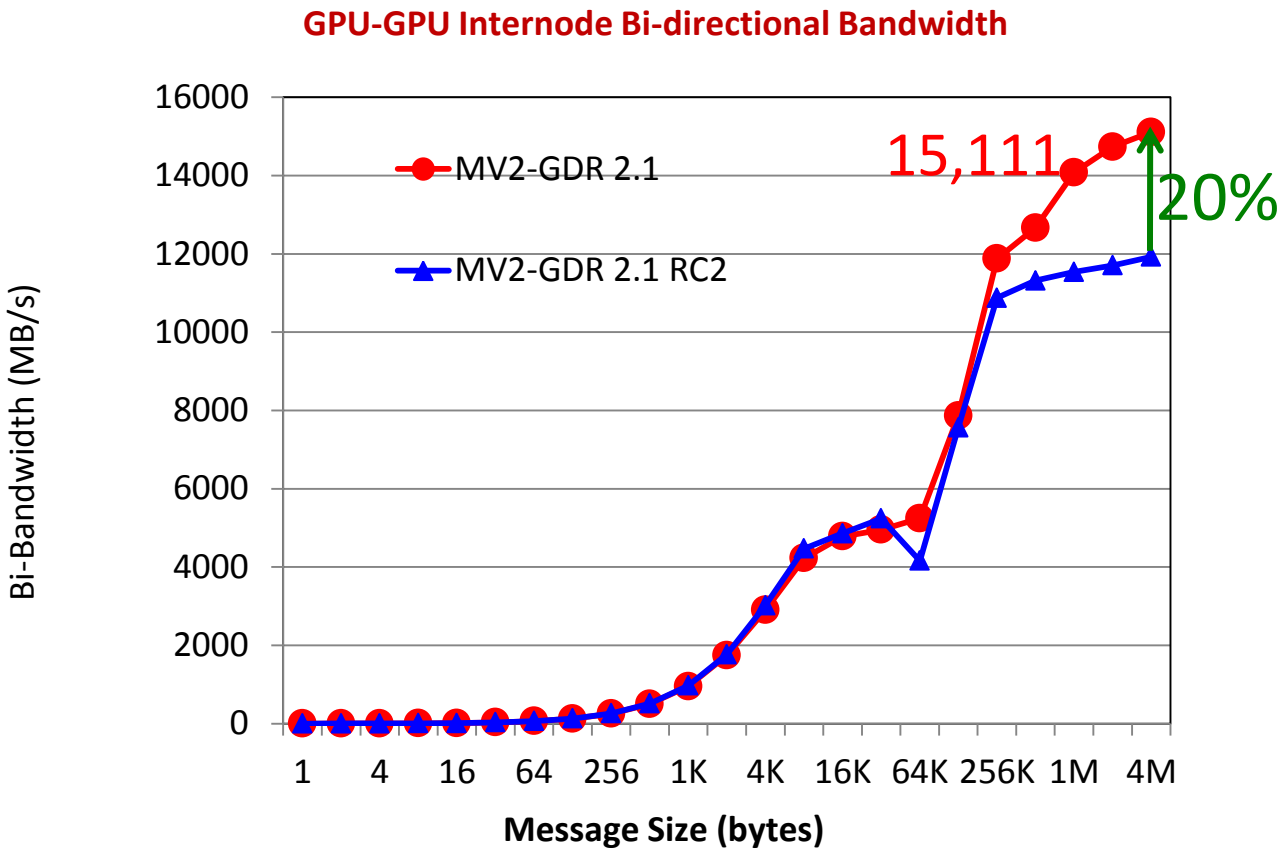
- Automatic rail and CPU binding depending on the GPU selection
  - User selects the GPU and MVAPICH2-GDR selects the best HCA (avoids the P2P bottleneck)
  - Multi-rail selection for large message size for better Bandwidth utilization (pipeline design)

Parameter	Significance	Default	Notes
MV2_RAIL_SHARING_POLICY	• How the Rails are bind/selected by processes	Shared	• Sharing gives the best performance for pipeline design
PROCESS_TO_RAIL_MAPPING	• Explicit binding of the HCAs to the CPU	First HCA	•

- **Refer to Tuning and Usage Parameters section of MVAPICH2-GDR user guide for more information**
- http://mvapich.cse.ohio-state.edu/userguide/gdr/#_tuning_and_usage_parameters

Performance of MVAPICH2-GDR with GPU-Direct RDMA and Multi-Rail Support





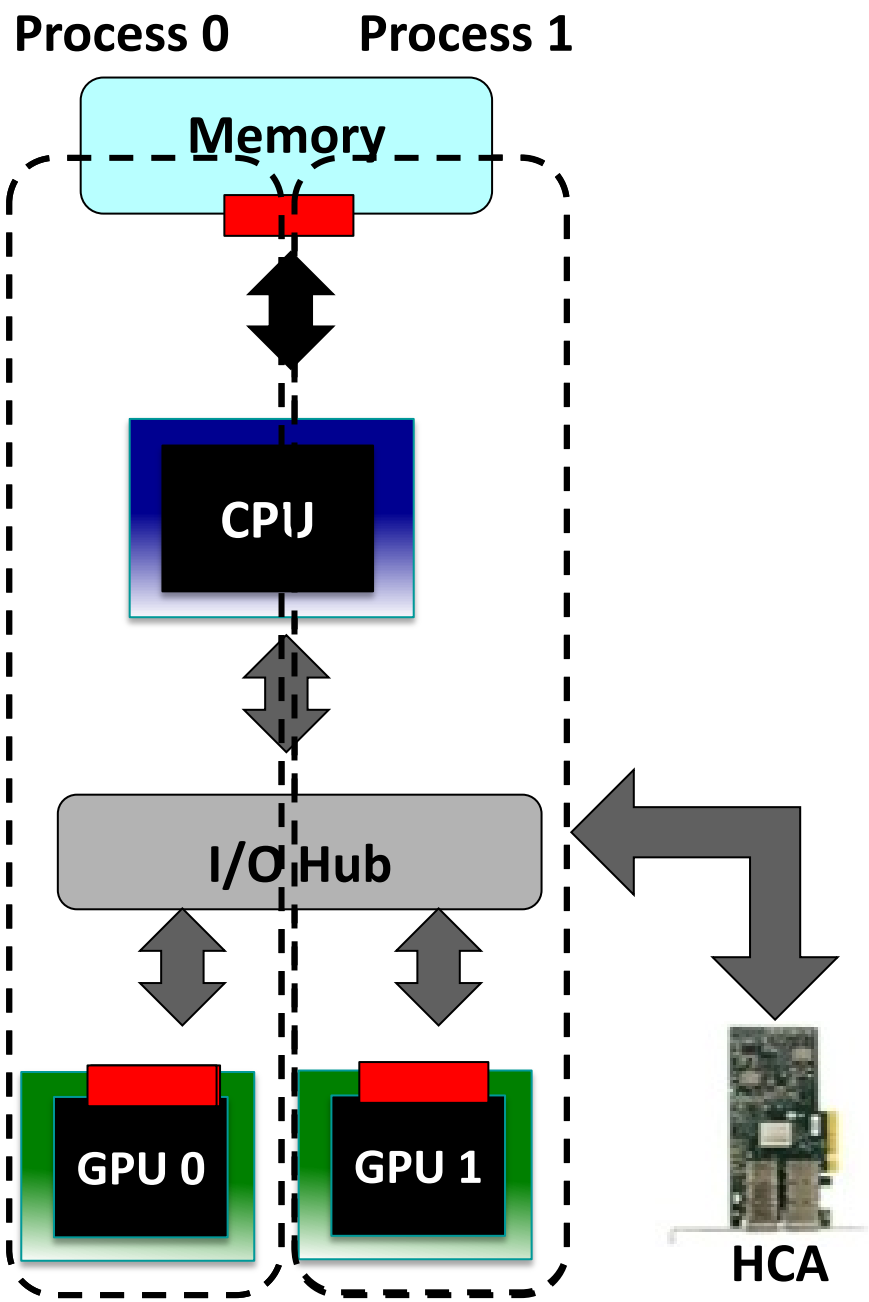
MVAPICH2-GDR-2.2.b
Intel Ivy Bridge (E5-2680 v2) node - 20 cores, NVIDIA Tesla K40c GPU
Mellanox Connect-IB Dual-FDR HCA CUDA 7
Mellanox OFED 2.4 with GPU-Direct-RDMA

Presentation Overview

- Support for Efficient Small Message Communication with GPUDirect RDMA
- Multi-rail Support
- **Support for Efficient Intra-node Communication using CUDA IPC**
- MPI Datatype Support
- On-going Work

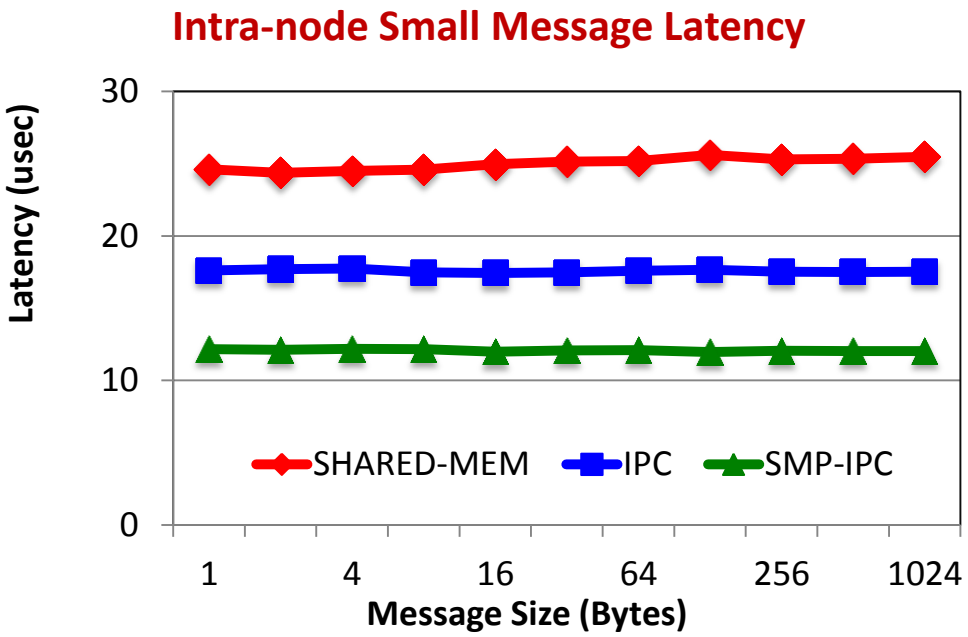
Multi-GPU Configurations

- Multi-GPU node architectures are becoming common
- Until CUDA 3.2
  - Communication between processes staged through the host
  - Shared Memory (pipelined)
  - Network Loopback [asynchronous)
- CUDA 4.0 and later
  - Inter-Process Communication (IPC)
  - Host bypass
  - Handled by a DMA Engine
  - Low latency and Asynchronous
  - Requires creation, exchange and mapping of memory handles
  - Overhead



Tuning IPC designs in MVAPICH2-GDR

- Works between GPUs within the same socket or IOH
- Leads to significant benefits in appropriate scenarios



Parameter	Significance	Default	Notes
MV2_CUDA_IPC	• Enable / Disable CUDA IPC-based designs	1 (Enabled)	• Always leave set to 1
MV2_CUDA_SMP_IPC	• Enable / Disable CUDA IPC fastpath design for short messages	0 (Disabled)	• Benefits Device-to-Device transfers • Hurts Device-to-Host/Host-to-Device transfers • Always set to 1 if application involves only Device-to-Device transfers
MV2_IPC_THRESHOLD	• Message size where IPC code path will be used	16 KBytes	• Tune for your system

Alternative Designs

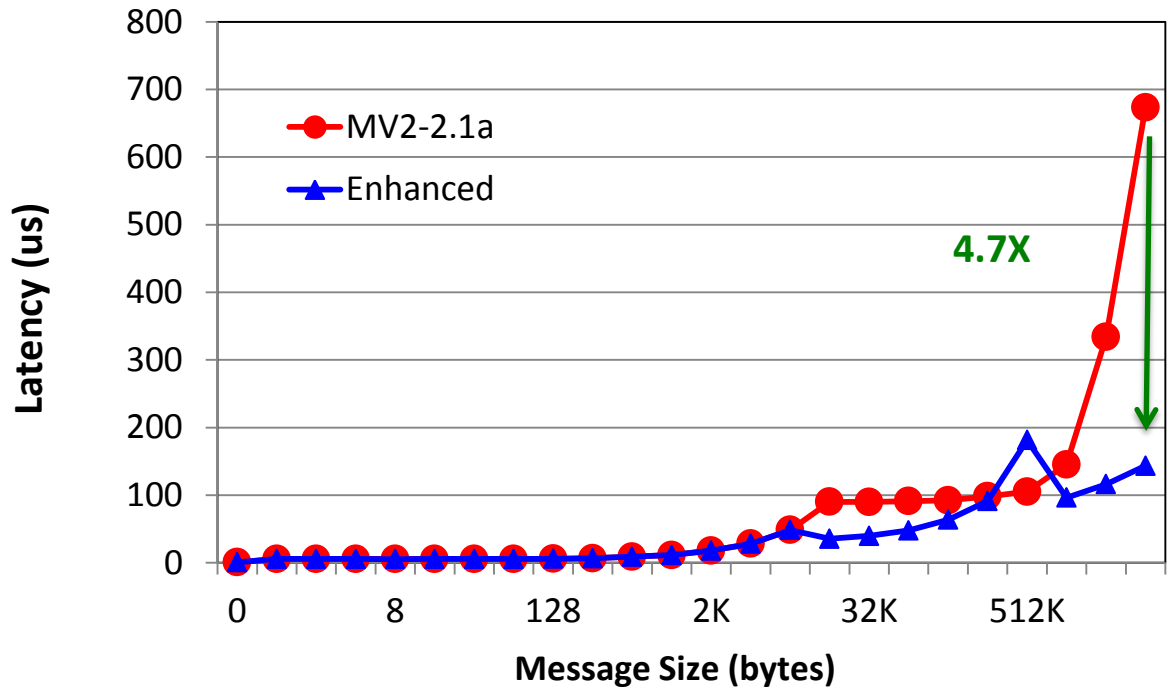
- Double Buffering schemes
  - Uses intermediate buffers (IPC Pinned)
  - Control information through Host memories
    - Exchange the handlers through the host for IPC completion
  - Works for all CUDA versions (Since 5.5)
  - Memory Overhead
- Cache based design
  - Rendezvous based design
  - Cache the IPC handlers at the source and destination (through the control messages)
  - With Cache hit => direct data movement
  - Requires CUDA 6.5 and onwards
  - High Performance and memory efficiency

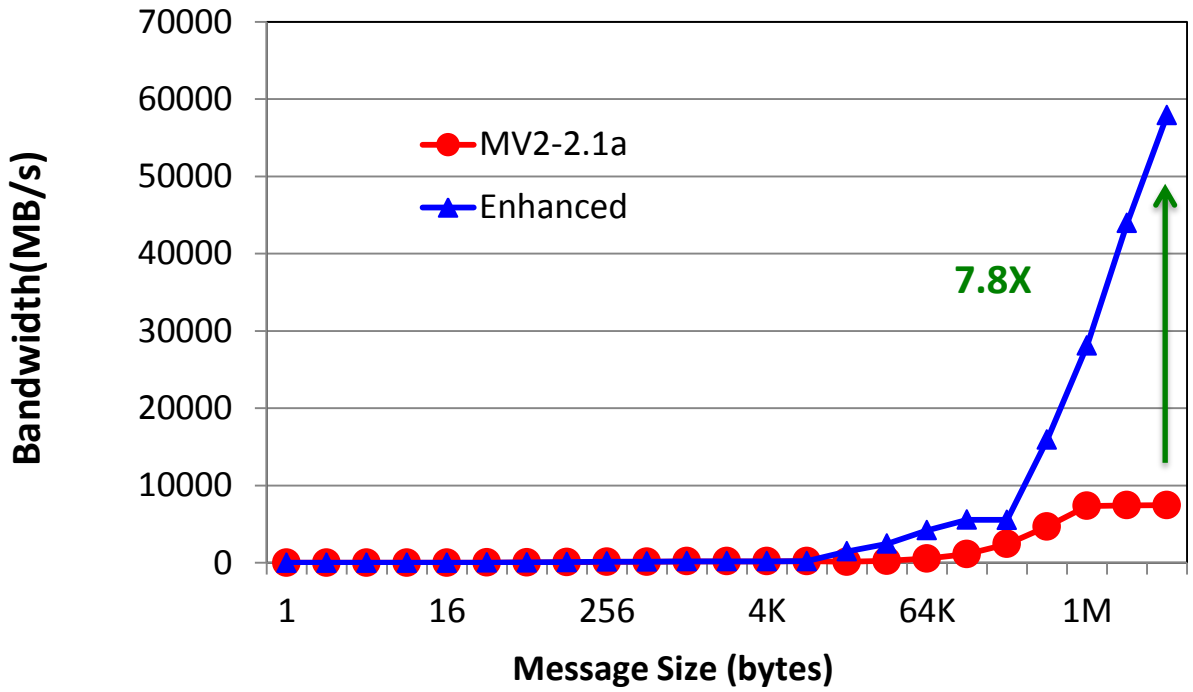
Tuning IPC designs in MVAPICH2-GDR

- Works between GPUs within the same socket or IOH

Parameter	Significance	Default	Notes
MV2_CUDA_ENABLE_IPC_CACHE	• Enable / Disable CUDA IPC_CACHE-based designs	1 (Enabled)	• Always leave set to 1 • Best performance • Enables one-sided semantics
MV2_CUDA_IPC_BUFFERED	• Enable / Disable CUDA IPC_BUFFERED design	1 (Enabled)	• Used for subset of operations • Backup for the IPC-Cache design • Uses double buffering schemes • Used for efficient Managed support
MV2_CUDA_IPC_MAX_CACHE_ENTRIES	• Number of entries in the cache	64	• Tuned for your application • Depends on the communication patterns • Increase the value for irregular applications
MV2_CUDA_IPC_STAGE_BUF_SIZE	• The size of the staging buffers in the double buffering schemes		

IPC-Cache Communication Enhancement





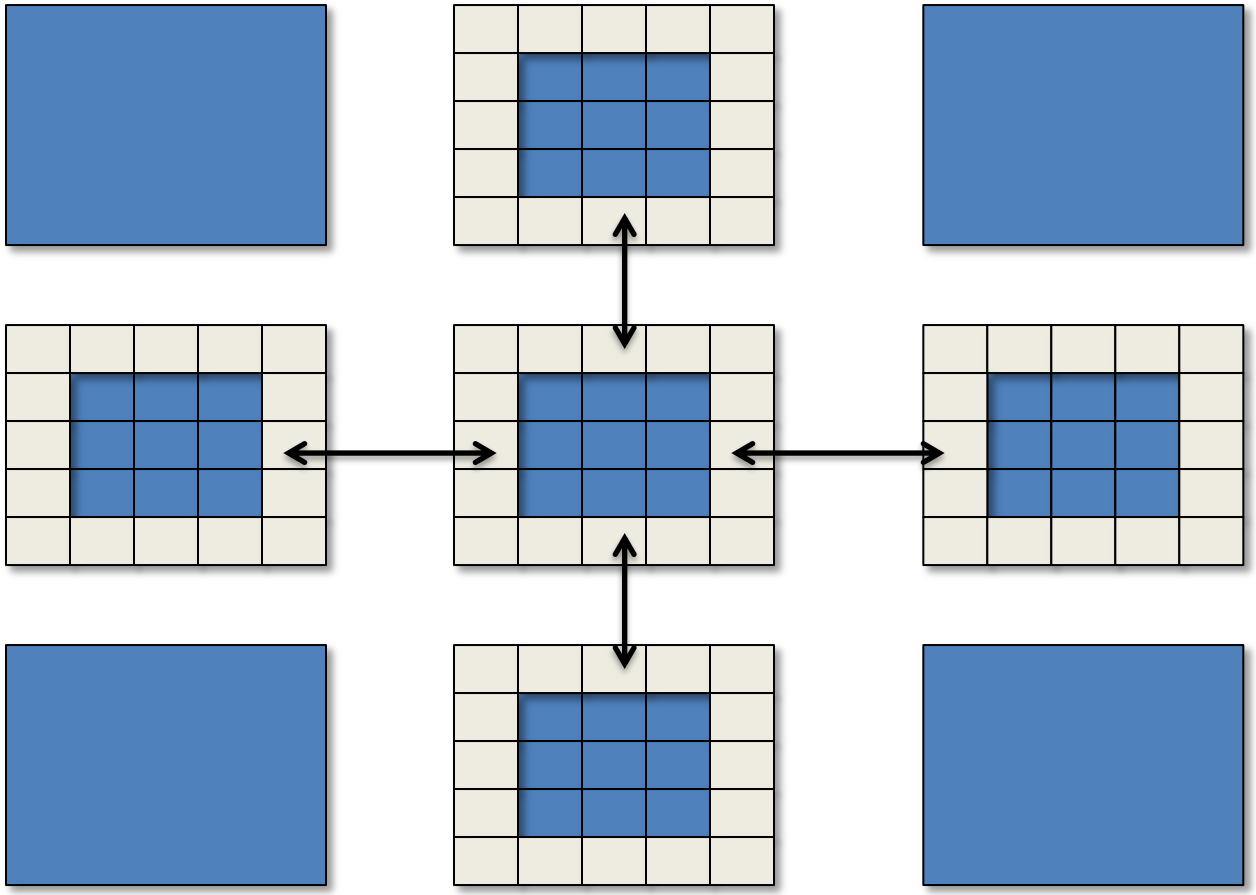
- Two Processes sharing the same K80 GPUs
- Proposed designs achieve 4.7X improvement in latency
- 7.8X improvement is delivered for Bandwidth
- Available with the latest release of MVAPICH2-GDR 2.2

Presentation Overview

- Support for Efficient Small Message Communication with GPUDirect RDMA
- Multi-rail Support
- Support for Efficient Intra-node Communication using CUDA IPC
- **MPI Datatype Support**
- On-going Work

Non-contiguous Data Exchange

Halo data exchange



- Multi-dimensional data
  - Row based organization
  - Contiguous on one dimension
  - Non-contiguous on other dimensions
- Halo data exchange
  - Duplicate the boundary
  - Exchange the boundary in each iteration

MPI Datatype support in MVAPICH2

- Datatypes support in MPI
  - Operate on customized datatypes to improve productivity
  - Enable MPI library to optimize non-contiguous data

At Sender:

```
MPI_Type_vector (n_blocks, n_elements, stride, old_type, &new_type);
MPI_Type_commit(&new_type);
…
MPI_Send(s_buf, size, new_type, dest, tag, MPI_COMM_WORLD);
```

- Inside MVAPICH2
  - Use datatype specific CUDA Kernels to pack data in chunks
  - Efficiently move data between nodes using RDMA
  - In progress - currently optimizes *vector* and *hindexed* datatypes
  - Transparent to the user

H. Wang, S. Potluri, D. Bureddy, C. Rosales and D. K. Panda, GPU-aware MPI on RDMA-Enabled Clusters: Design, Implementation and Evaluation, IEEE Transactions on Parallel and Distributed Systems, Accepted for Publication.

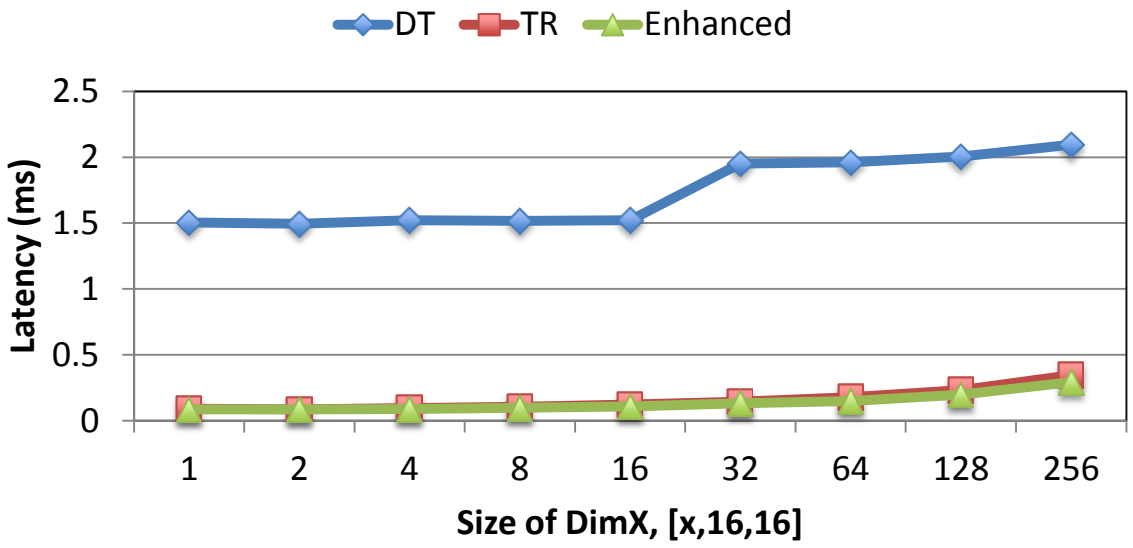
MPI Datatype Processing (Computation Optimization)

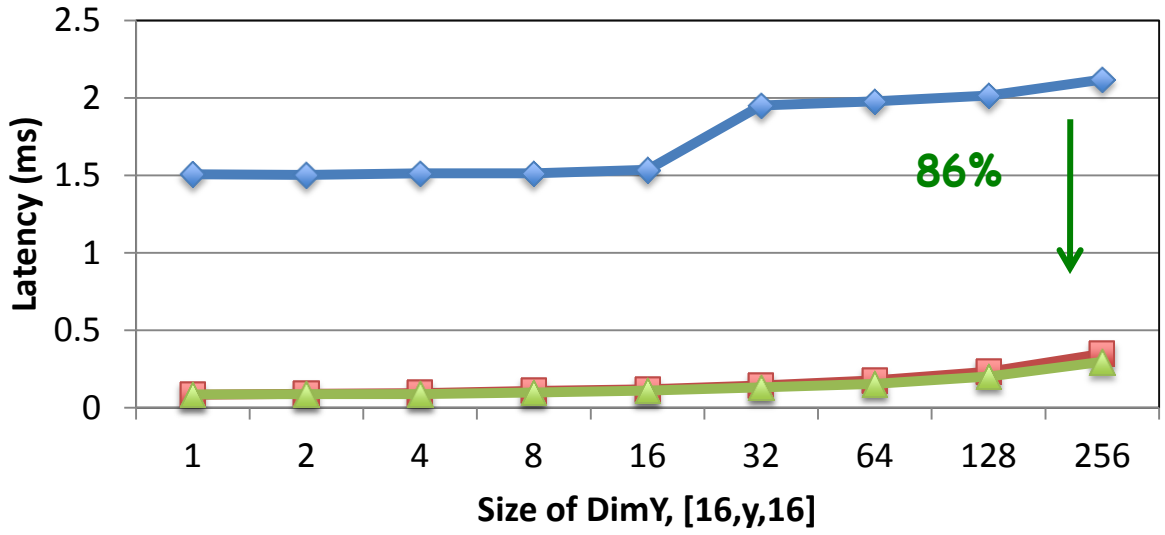
- Comprehensive support
  - Targeted kernels for regular datatypes - vector, subarray, indexed_block
  - Generic kernels for all other irregular datatypes
- Separate non-blocking stream for kernels launched by MPI library
  - Avoids stream conflicts with application kernels
- Flexible set of parameters for users to tune kernels
  - Vector
    - MV2_CUDA_KERNEL_VECTOR_TIDBLK_SIZE
    - MV2_CUDA_KERNEL_VECTOR_YSIZE
  - Subarray
    - MV2_CUDA_KERNEL_SUBARR_TIDBLK_SIZE
    - MV2_CUDA_KERNEL_SUBARR_XDIM
    - MV2_CUDA_KERNEL_SUBARR_YDIM
    - MV2_CUDA_KERNEL_SUBARR_ZDIM
  - Indexed_block
    - MV2_CUDA_KERNEL_IDXBLK_XDIM

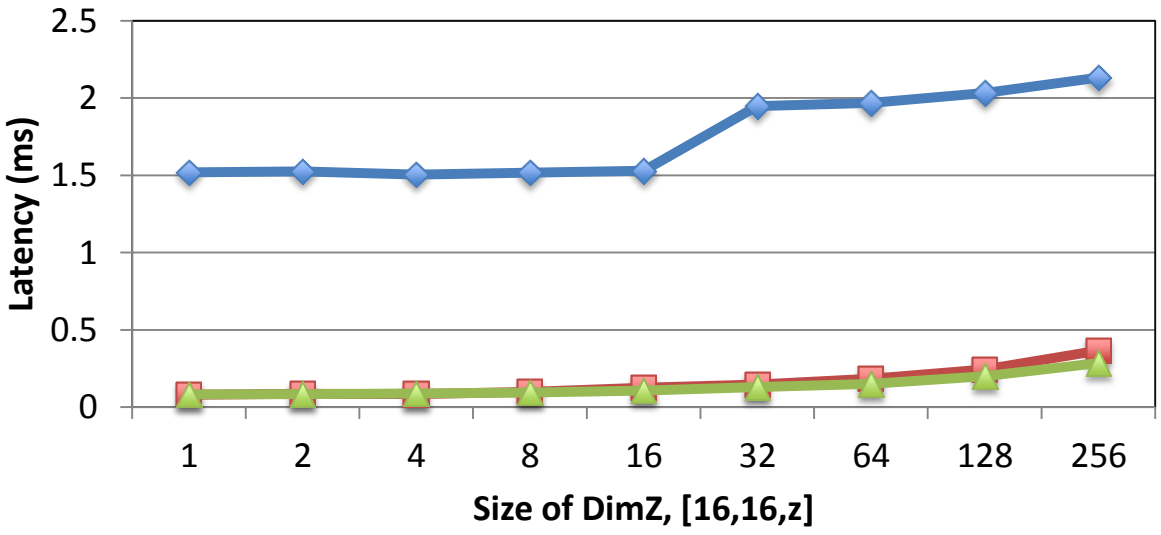
Performance of Stencil3D (3D subarray)

Stencil3D communication kernel on 2 GPUs with various X, Y, Z dimensions using MPI_Isend/Irecv

- DT: Direct Transfer, TR: Targeted Kernel
- Optimized design gains up to 15%, 15% and 22% compared to TR, and more than 86% compared to DT on X, Y and Z respectively







MPI Datatype Processing (Communication Optimization)

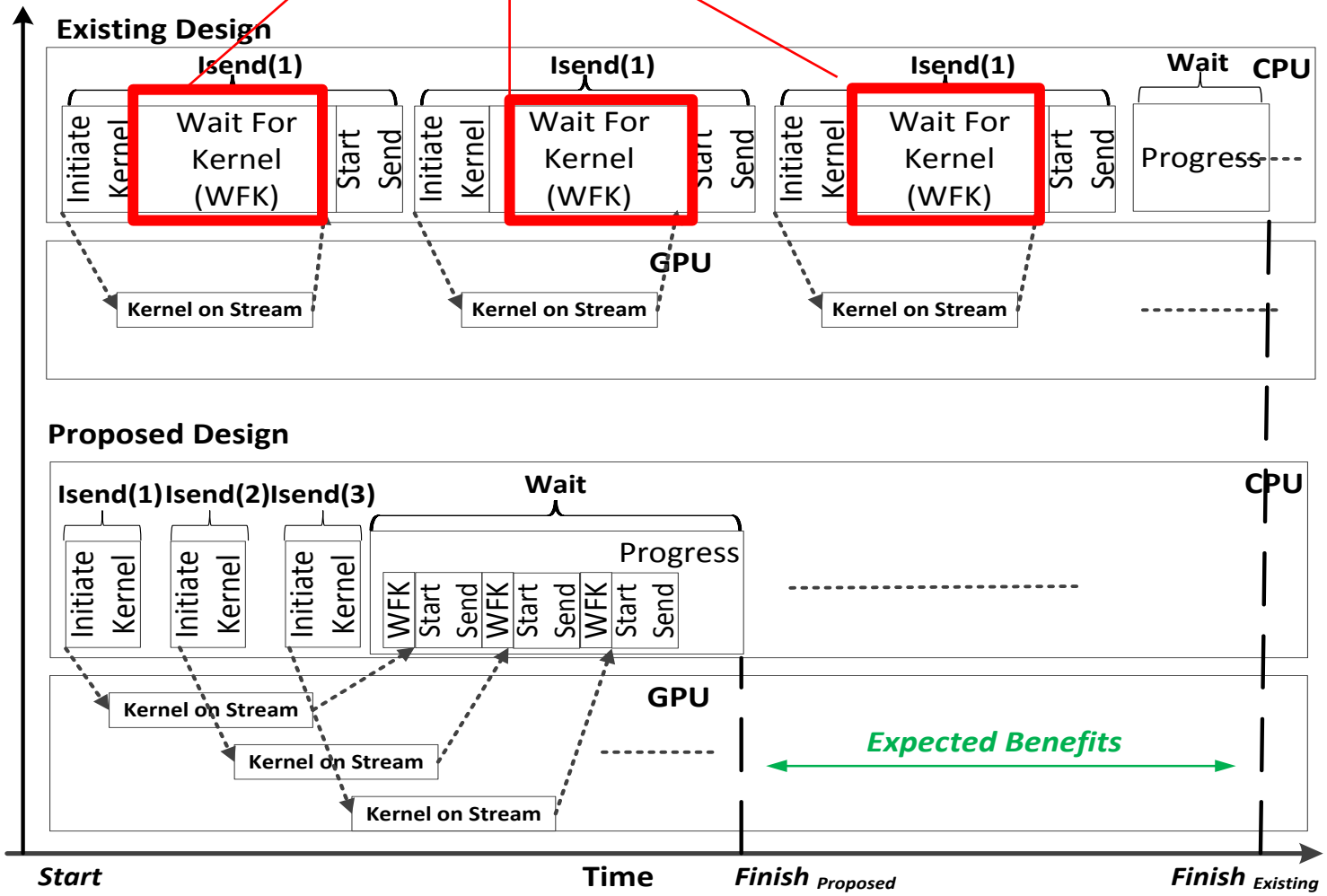
Common Scenario

MPI_Isend (A,.. Datatype,…)
MPI_Isend (B,.. Datatype,…)
MPI_Isend (C,.. Datatype,…)
MPI_Isend (D,.. Datatype,…)
…

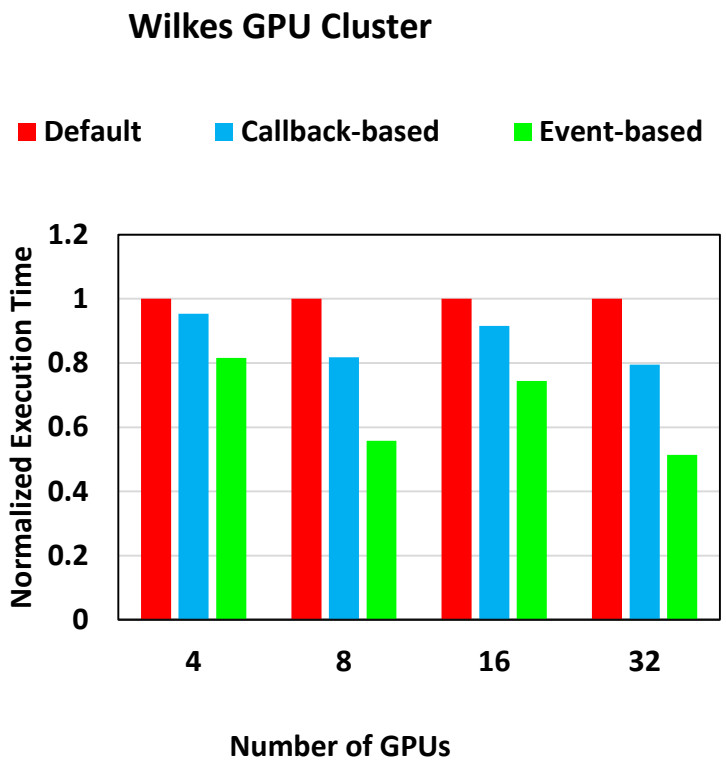
MPI_Waitall (…);

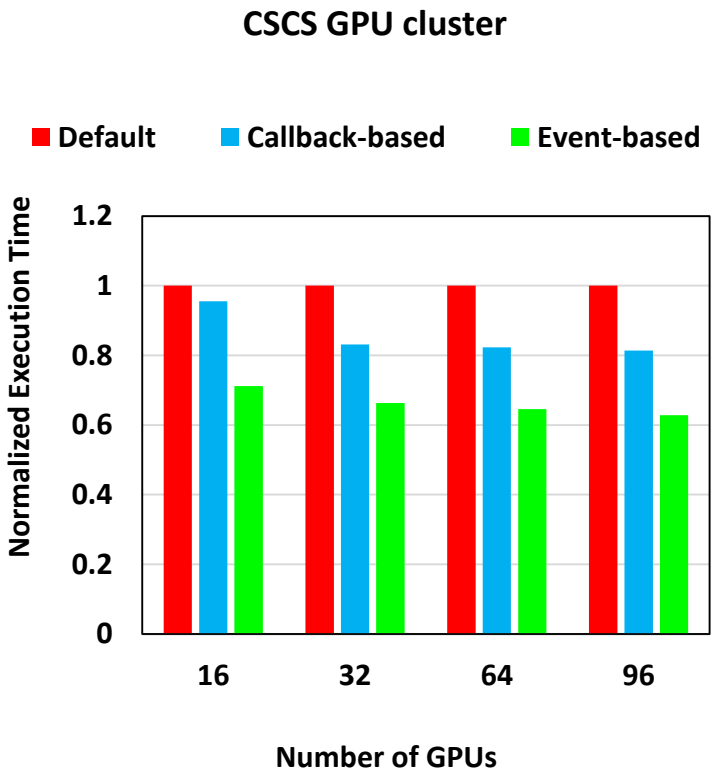
*A, B…contain non-contiguous MPI Datatype

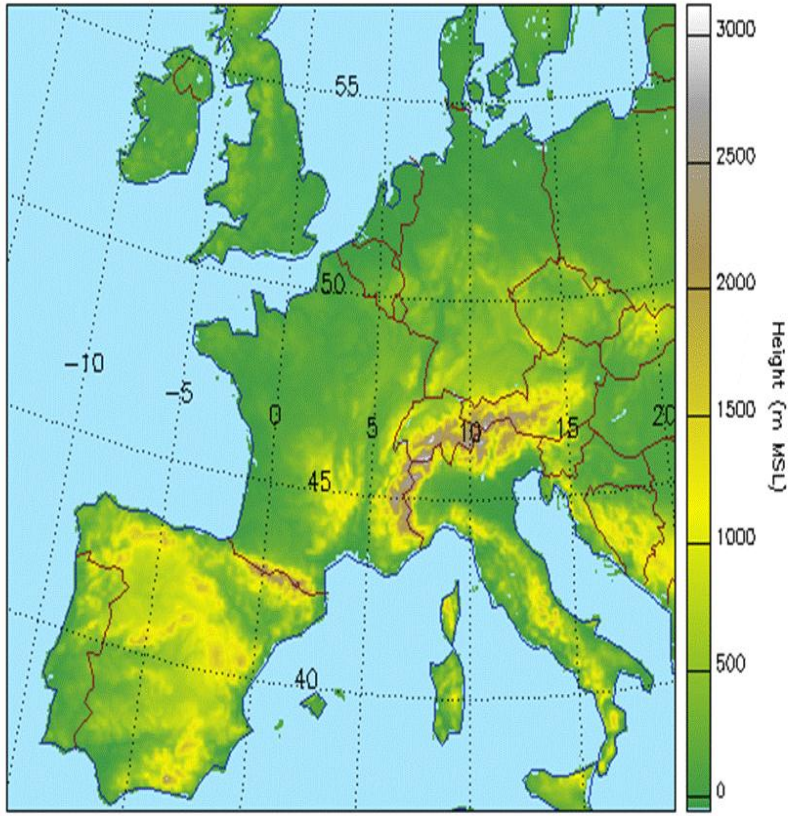
Waste of computing resources on CPU and GPU



Application-Level Evaluation (Cosmo) and Weather Forecasting in Switzerland







- 2X improvement on 32 GPUs nodes
- 30% improvement on 96 GPU nodes (8 GPUs/node)

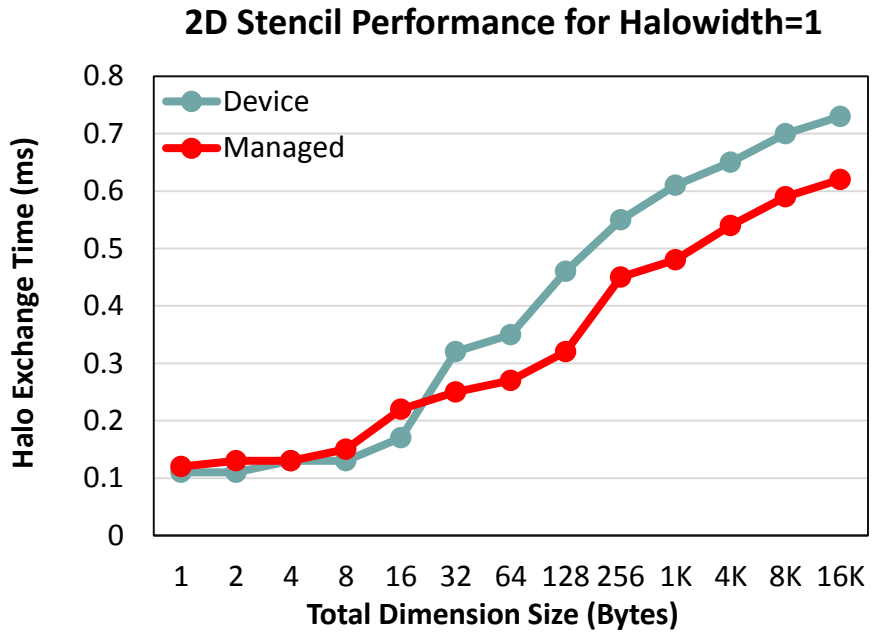
Cosmo model: http://www2.cosmo-model.org/content/tasks/operational/meteoSwiss/

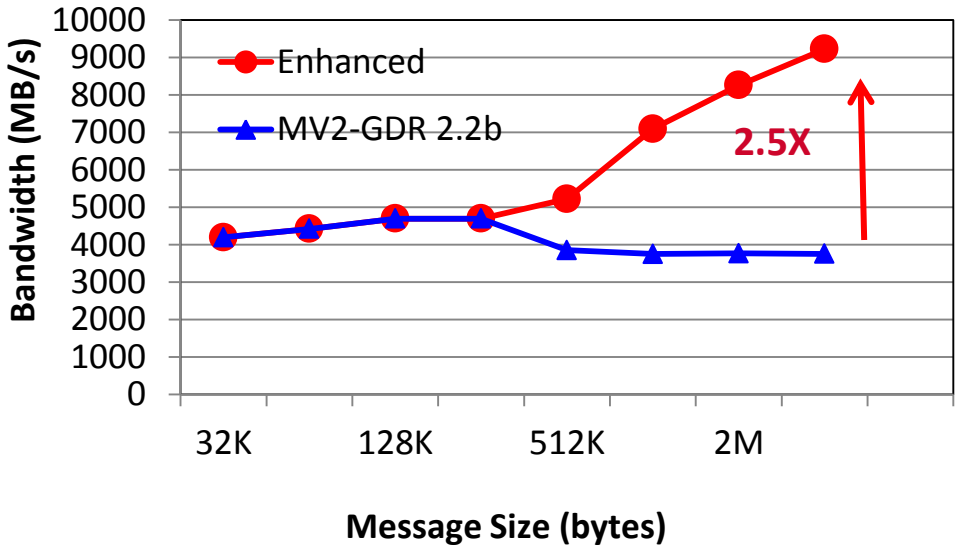
On-going collaboration with CSCS and MeteoSwiss (Switzerland) in co-designing MV2-GDR and Cosmo Application

C. Chu, K. Hamidouche, A. Venkatesh, D. Banerjee , H. Subramoni, and D. K. Panda, Exploiting Maximal Overlap for Non-Contiguous Data Movement Processing on Modern GPU-enabled Systems, IPDPS'16

Enhanced Support for GPU Managed Memory

- CUDA Managed => no memory pin down
  - No IPC support for intranode communication
  - No GDR support for Internode communication
- Significant productivity benefits due to abstraction of explicit allocation and *cudaMemcpy()*
- Initial and basic support in MVAPICH2-GDR
  - For both intra- and inter-nodes use "pipeline through" host memory
- Enhance intranode managed memory to use IPC
  - Double buffering pair-wise IPC-based scheme
  - Brings IPC performance to Managed memory
  - High performance and high productivity
  - 2.5 X improvement in bandwidth
- OMB extended to evaluate the performance of point-to-point and collective communications using managed buffers





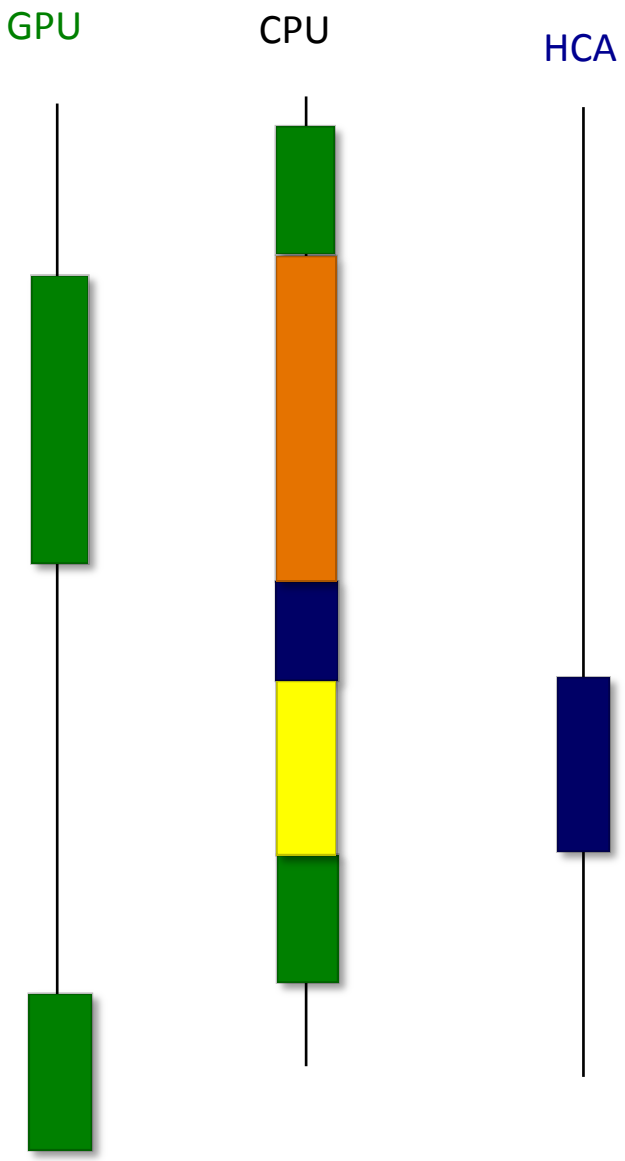
D. S. Banerjee, K Hamidouche, and D. K Panda, Designing High Performance Communication Runtime for GPUManaged Memory: Early Experiences, GPGPU-9 Workshop, held in conjunction with PPoPP '16

Presentation Overview

- Support for Efficient Small Message Communication with GPUDirect RDMA
- Multi-rail Support
- Support for Efficient Intra-node Communication using CUDA IPC
- MPI Datatype Support
- **On-going Work**

Overview of GPUDirect aSync (GDS) Feature: Current MPI+CUDA interaction

```
CUDA_Kernel_a<<<>>>(A…., stream1)
cudaStreamSynchronize(stream1)
MPI_Isend (A,…., req1)
MPI_Wait (req1)
CUDA_Kernel_b<<<>>>(B…., stream1)
```



100% CPU control

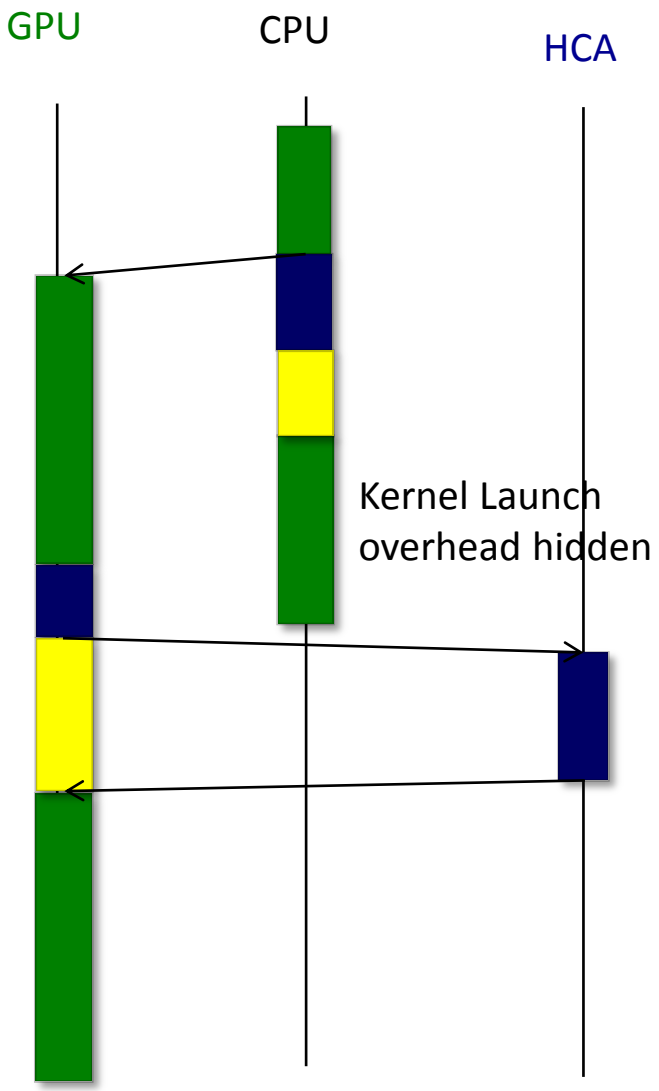
- Limit the throughput of a GPU
- Limit the asynchronous progress
- Waste GPU cycles

MVAPICH2-GDS: Decouple GPU Control Flow from CPU

CUDA_Kernel_a<<<>>>(A…., stream1)
MPI_Isend (A,…., req1, stream1)
MPI_Wait (req1, stream1) (non-blocking from CPU)
CUDA_Kernel_b<<<>>>(B…., stream1)

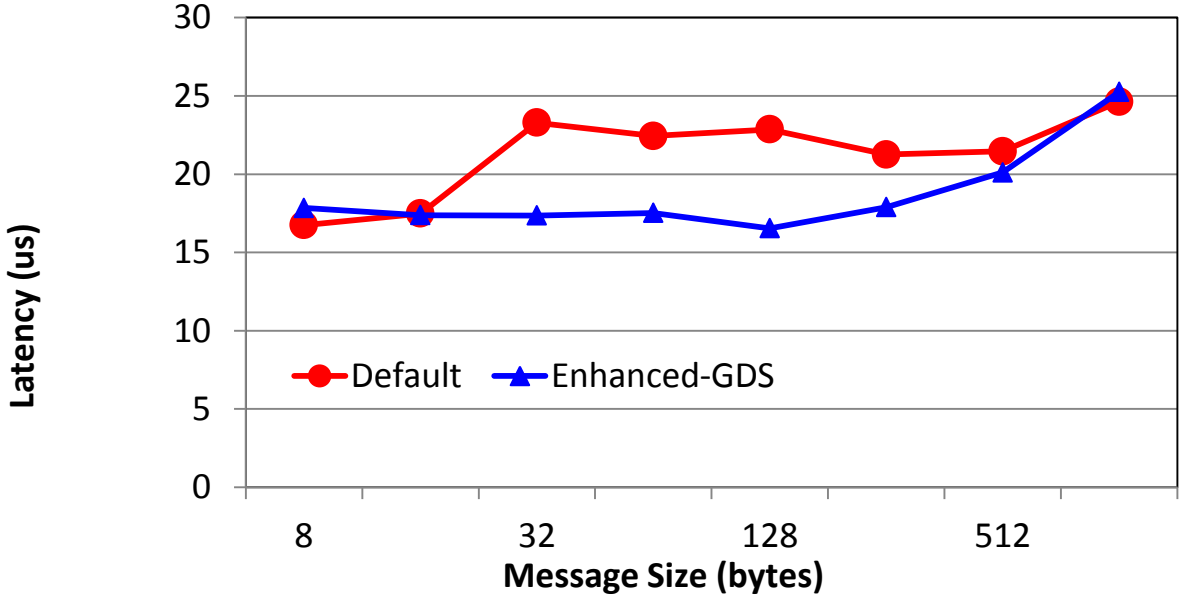
CPU offloads the compute, communication and synchronization tasks to GPU

- CPU is out of the critical path
- Tight interaction between GPU and HCA
- Hide the overhead of kernel launch
- Requires MPI semantics extensions
  - All operations are asynchronous from CPU
  - Extend MPI semantics with Stream-based semantics

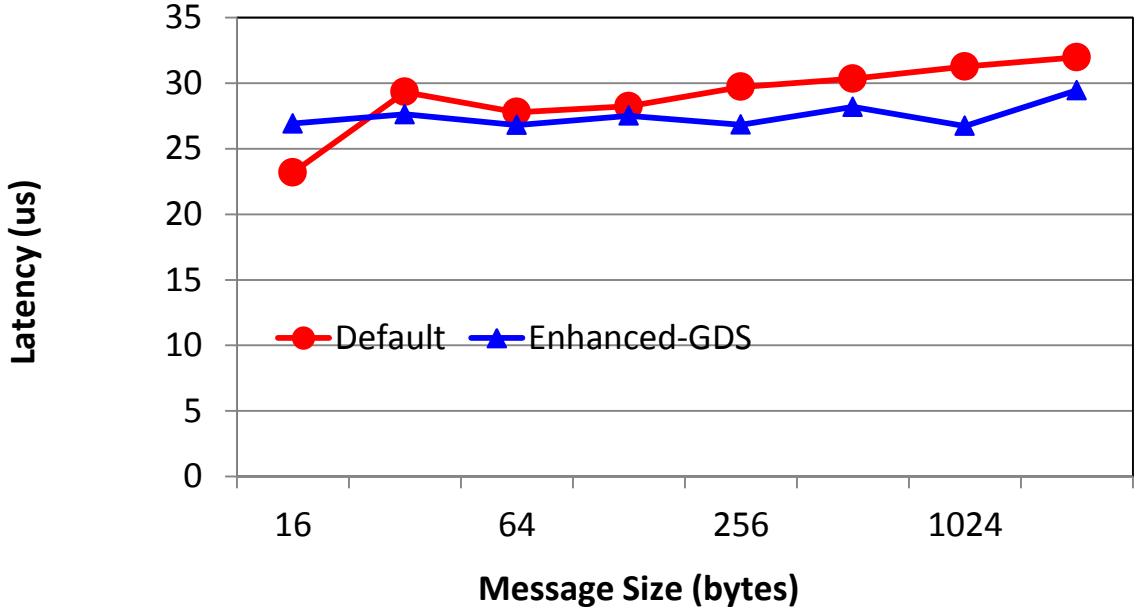


MVAPICH2-GDS: Preliminary Results

Latency oriented: Send+kernel and Recv+kernel



Throughput Oriented: back-to-back



- Latency Oriented: Able to hide the kernel launch overhead
  - 25% improvement at 256 Bytes compared to default behavior
- Throughput Oriented: Asynchronously to offload queue the Communication and computation tasks
  - 14% improvement at 1KB message size
  - Requires some tuning and expect better performance for Application with different Kernels

Intel Sandy Bridge, NVIDIA K20 and Mellanox FDR HCA

MVAPICH2 Software Family

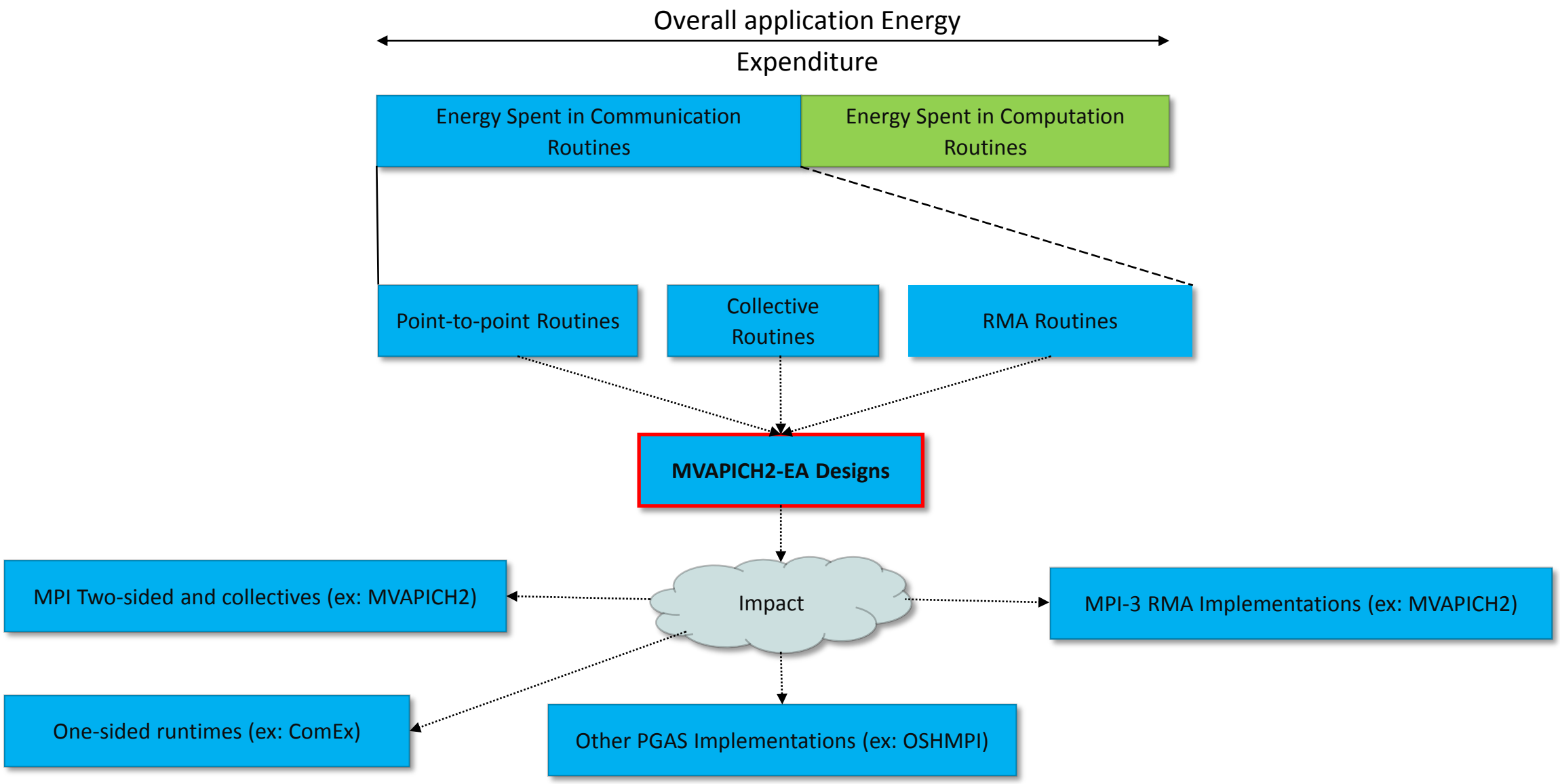
Requirements	Library
MPI with IB, iWARP and RoCE	MVAPICH2
Advanced MPI, OSU INAM, PGAS and MPI+PGAS with IB and RoCE	MVAPICH2-X
MPI with IB & GPU	MVAPICH2-GDR
MPI with IB & MIC	MVAPICH2-MIC
HPC Cloud with MPI & IB	MVAPICH2-Virt
Energy-aware MPI with IB, iWARP and RoCE	MVAPICH2-EA
MPI Energy Monitoring Tool	OEMT
InfiniBand Network Analysis and Monitoring	OSU INAM
Microbenchmarks for Measuring MPI and PGAS Performance	OMB

Energy-Aware MVAPICH2 & OSU Energy Management Tool (OEMT)

- MVAPICH2-EA 2.1 (Energy-Aware)
  - A white-box approach
  - New Energy-Efficient communication protocols for pt-pt and collective operations
  - Intelligently apply the appropriate Energy saving techniques
  - Application oblivious energy saving

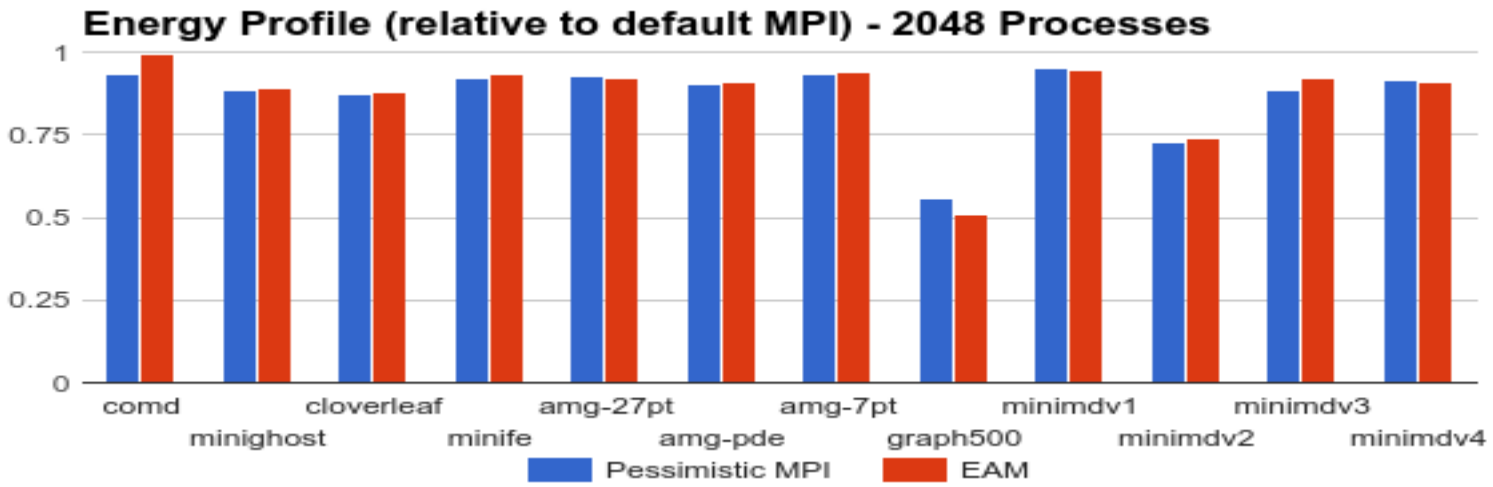
- OEMT
  - A library utility to measure energy consumption for MPI applications
  - Works with all MPI runtimes
  - PRELOAD option for precompiled applications
  - Does not require ROOT permission:
    - A safe kernel module to read only a subset of MSRs

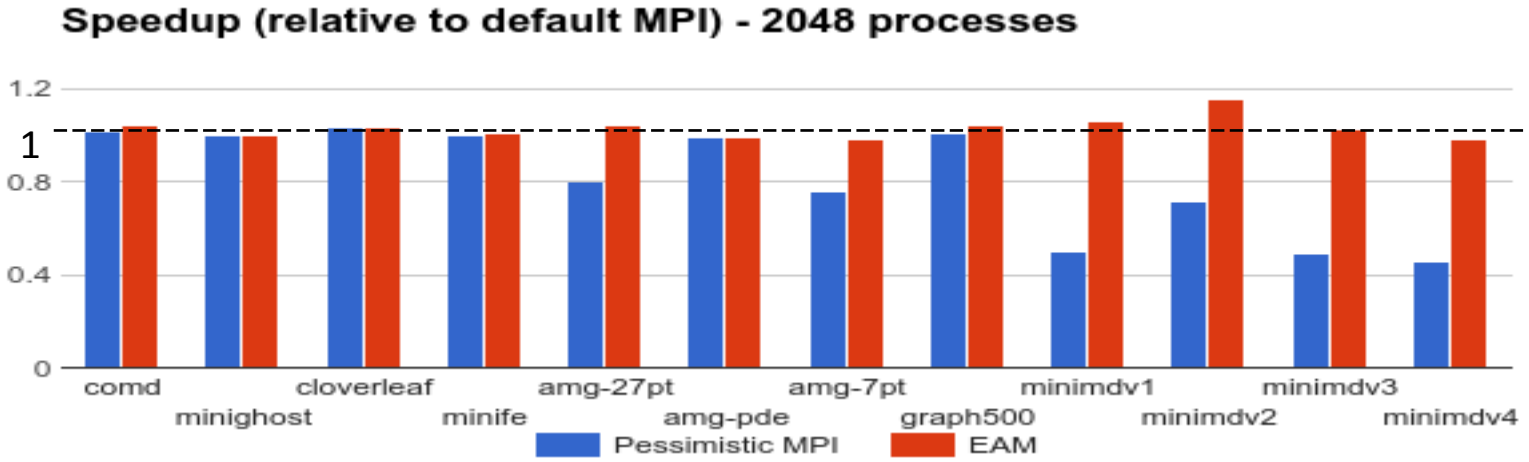
Designing Energy-Aware (EA) MPI Runtime



MVAPICH2-EA: Application Oblivious Energy-Aware-MPI (EAM)

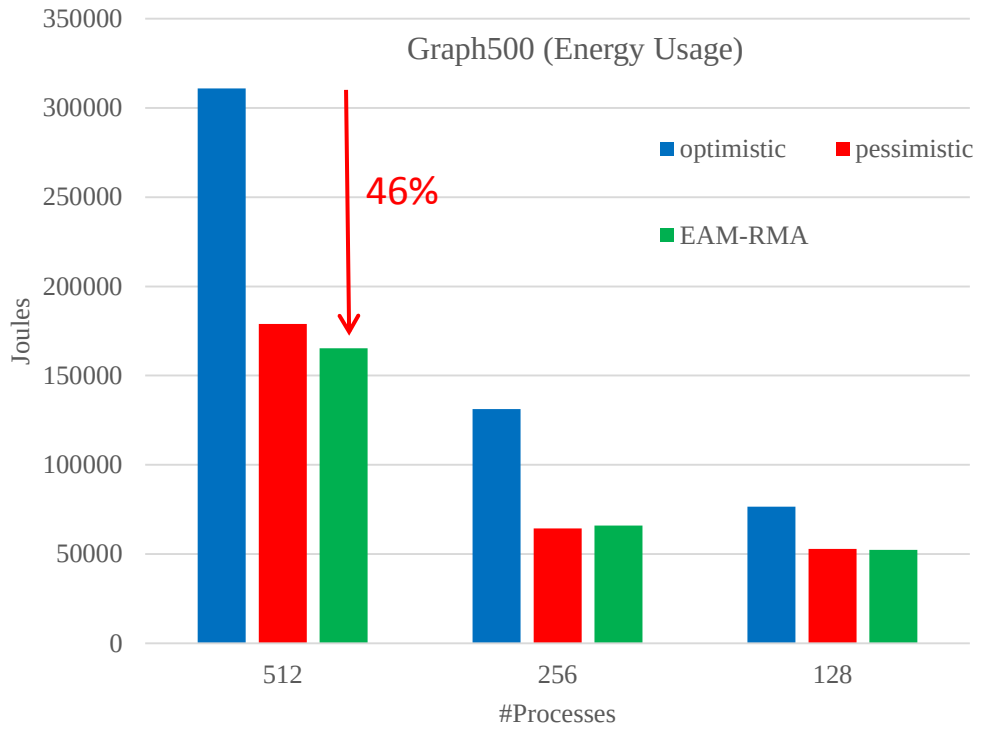
- An energy efficient runtime that provides energy savings without application knowledge
- Uses automatically and transparently the best energy lever
- Provides guarantees on maximum degradation with 5-41% savings at <= 5% degradation
- Pessimistic MPI applies energy reduction lever to each MPI call

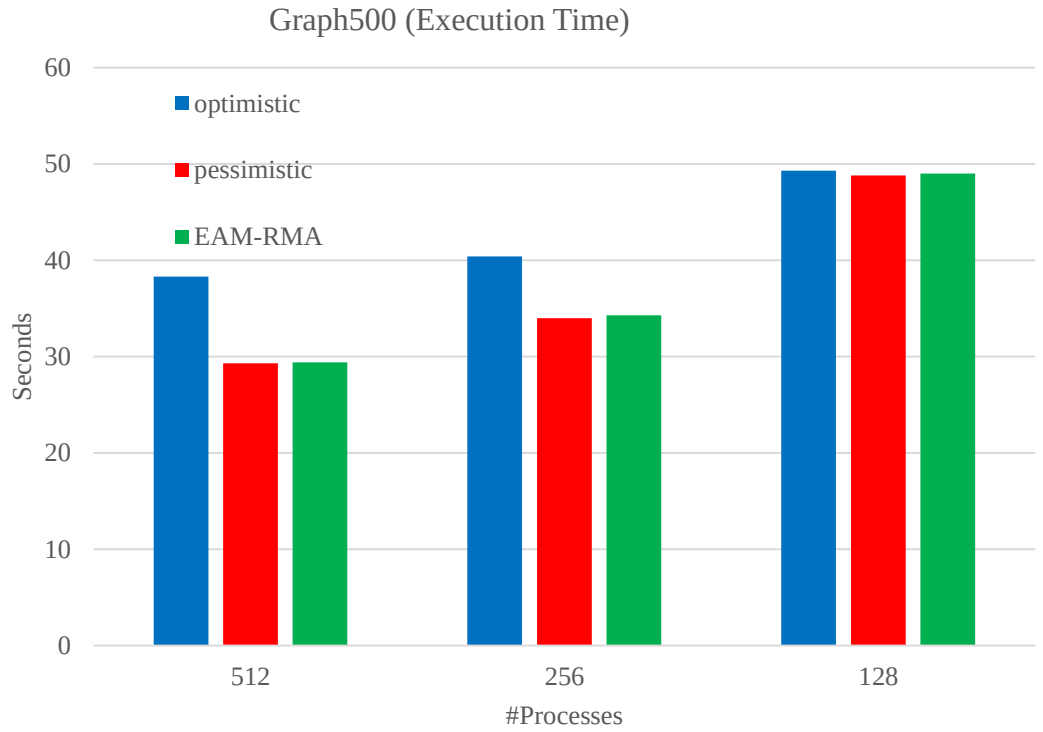




A Case for Application-Oblivious Energy-Efficient MPI Runtime A. Venkatesh, A. Vishnu, K. Hamidouche, N. Tallent, D. K. Panda, D. Kerbyson, and A. Hoise, Supercomputing '15, Nov 2015 *[Best Student Paper Finalist]*

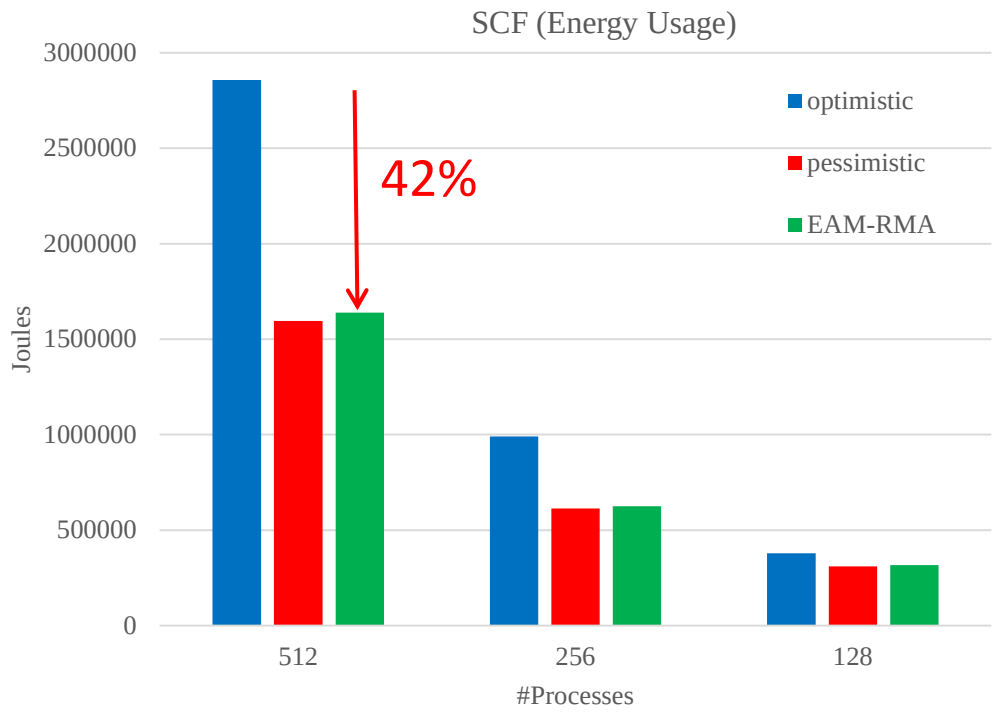
MPI-3 RMA Energy Savings with Proxy-Applications

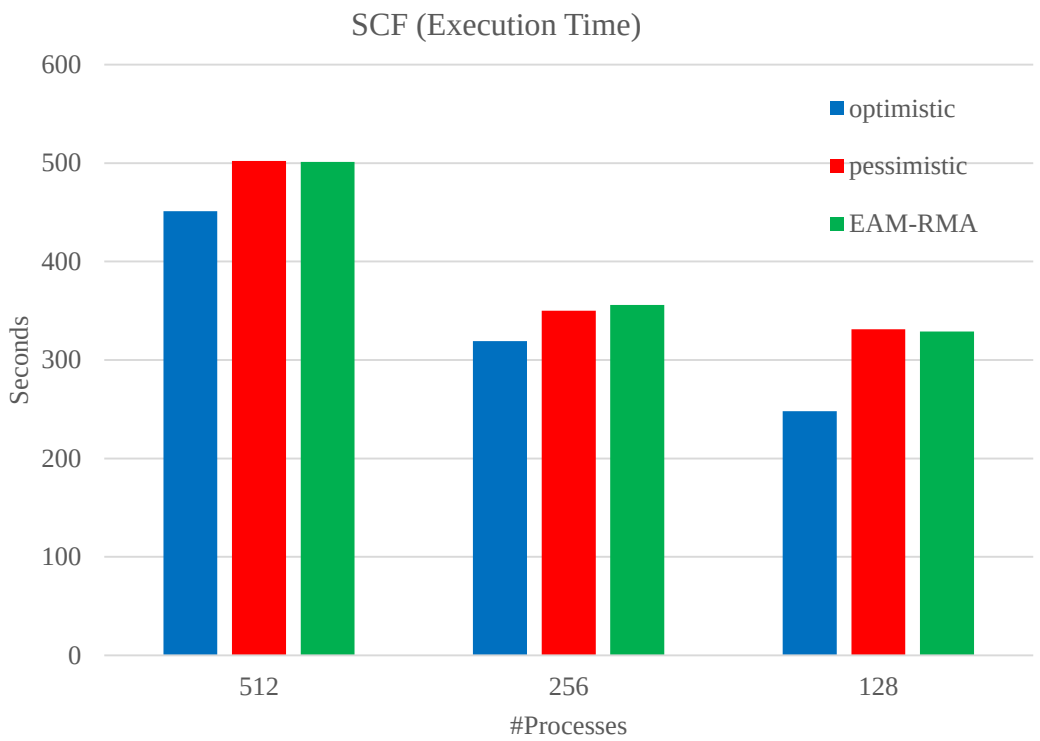




- MPI_Win_fence dominates application execution time in graph500
- Between 128 and 512 processes, EAM-RMA yields between 31% and 46% savings with no degradation in execution time in comparison with the default optimistic MPI runtime

MPI-3 RMA Energy Savings with Proxy-Applications





- SCF (self-consistent field) calculation spends nearly 75% total time in MPI_Win_unlock call
- With 256 and 512 processes, EAM-RMA yields 42% and 36% savings at 11% degradation (close to permitted degradation ρ = 10%)
- 128 processes is an exception due 2-sided and 1-sided interaction
- MPI-3 RMA Energy-efficient support will be available in upcoming MVAPICH2-EA release

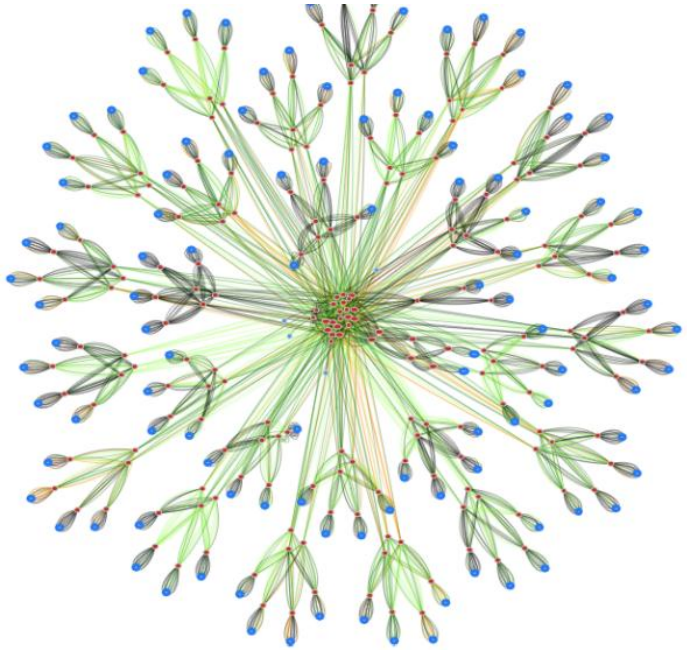
MVAPICH2 Software Family

Requirements	Library
MPI with IB, iWARP and RoCE	MVAPICH2
Advanced MPI, OSU INAM, PGAS and MPI+PGAS with IB and RoCE	MVAPICH2-X
MPI with IB & GPU	MVAPICH2-GDR
MPI with IB & MIC	MVAPICH2-MIC
HPC Cloud with MPI & IB	MVAPICH2-Virt
Energy-aware MPI with IB, iWARP and RoCE	MVAPICH2-EA
MPI Energy Monitoring Tool	OEMT
InfiniBand Network Analysis and Monitoring	OSU INAM
Microbenchmarks for Measuring MPI and PGAS Performance	OMB

Overview of OSU INAM

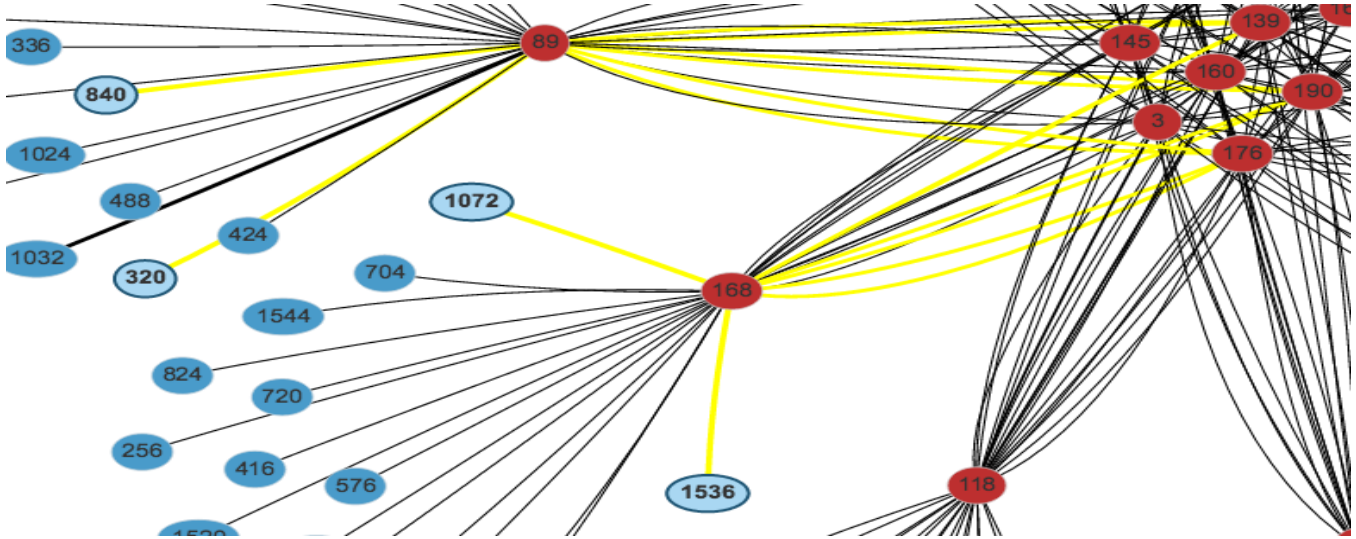
- A network monitoring and analysis tool that is capable of analyzing traffic on the InfiniBand network with inputs from the MPI runtime
  - http://mvapich.cse.ohio-state.edu/tools/osu-inam/
- Monitors IB clusters in real time by querying various subnet management entities and gathering input from the MPI runtimes
- OSU INAM v0.9.1 released on 05/13/16
- Significant enhancements to user interface to enable scaling to clusters with thousands of nodes
- Improve database insert times by using 'bulk inserts‘
- Capability to look up list of nodes communicating through a network link
- Capability to classify data flowing over a network link at job level and process level granularity in conjunction with MVAPICH2-X 2.2rc1
- “Best practices “ guidelines for deploying OSU INAM on different clusters
- Capability to analyze and profile node-level, job-level and process-level activities for MPI communication
  - Point-to-Point, Collectives and RMA
- Ability to filter data based on type of counters using “drop down” list
- Remotely monitor various metrics of MPI processes at user specified granularity
- "Job Page" to display jobs in ascending/descending order of various performance metrics in conjunction with MVAPICH2-X
- Visualize the data transfer happening in a “live” or “historical” fashion for entire network, job or set of nodes

OSU INAM Features



Comet@SDSC --- Clustered View

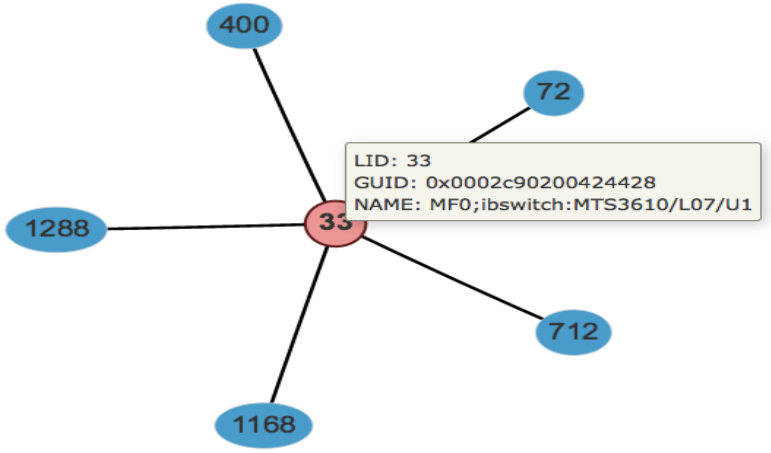
(1,879 nodes, 212 switches, 4,377 network links)



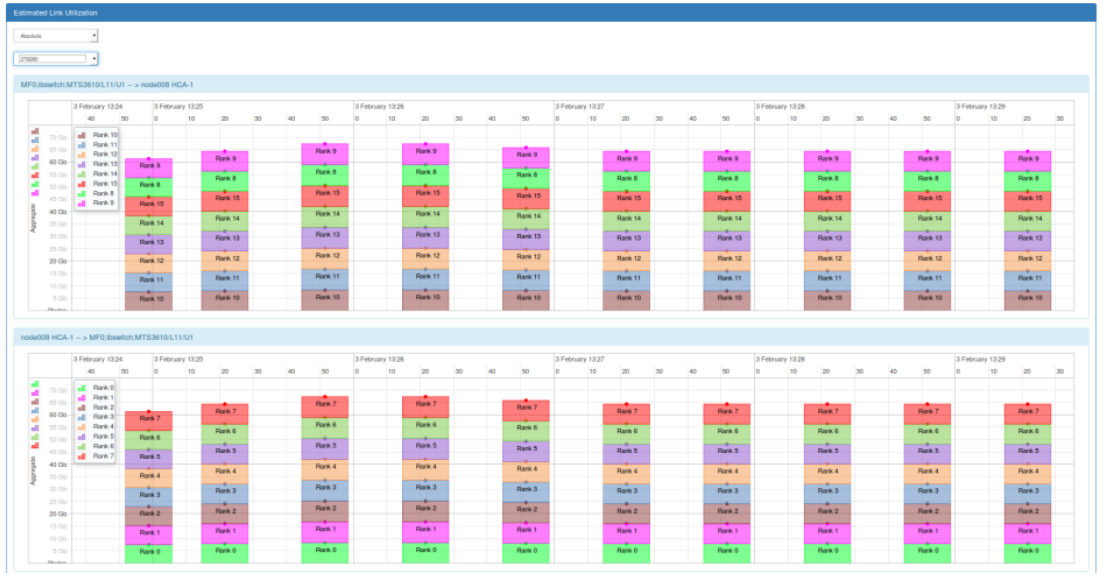
Finding Routes Between Nodes

- Show network topology of large clusters
- Visualize traffic pattern on different links
- Quickly identify congested links/links in error state
- See the history unfold – play back historical state of the network

OSU INAM Features (Cont.)



Visualizing a Job (5 Nodes)



Estimated Process Level Link Utilization

- Job level view
  - Show different network metrics (load, error, etc.) for any live job
  - Play back historical data for completed jobs to identify bottlenecks
- Node level view - details per process or per node
  - CPU utilization for each rank/node
  - Bytes sent/received for MPI operations (pt-to-pt, collective, RMA)
  - Network metrics (e.g. XmitDiscard, RcvError) per rank/node
- Estimated Link Utilization view
  - Classify data flowing over a network link at different granularity in conjunction with MVAPICH2-X 2.2rc1
    - Job level and
    - Process level

More Details in Tutorial/Demo Session Tomorrow

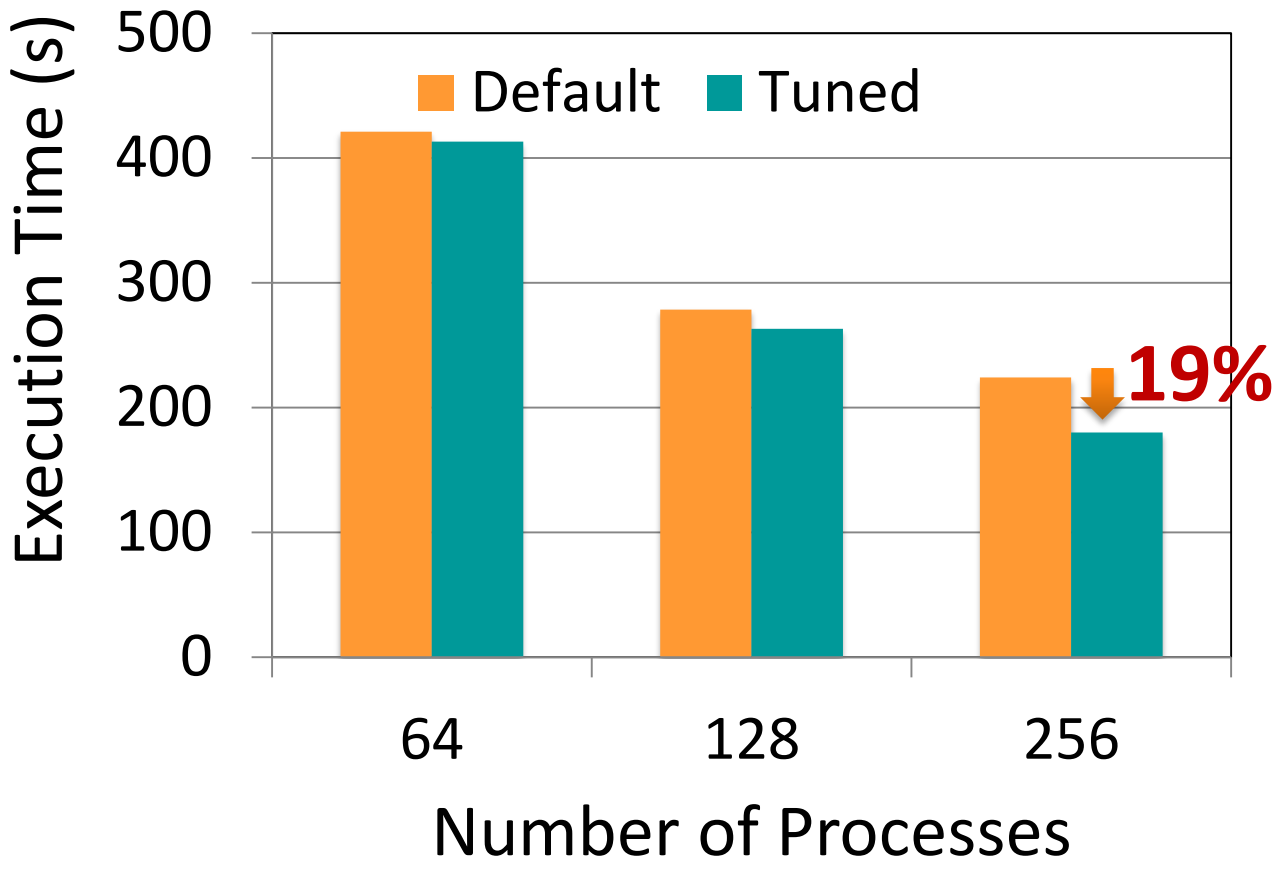
OSU Microbenchmarks

- Available since 2004
- Suite of microbenchmarks to study communication performance of various programming models
- Benchmarks available for the following programming models
  - Message Passing Interface (MPI)
  - Partitioned Global Address Space (PGAS)
    - Unified Parallel C (UPC)
    - Unified Parallel C++ (UPC++)
    - OpenSHMEM
- Benchmarks available for multiple accelerator based architectures
  - Compute Unified Device Architecture (CUDA)
  - OpenACC Application Program Interface
- Part of various national resource procurement suites like NERSC-8 / Trinity Benchmarks
- Continuing to add support for newer primitives and features
- Please visit the following link for more information
  - http://mvapich.cse.ohio-state.edu/benchmarks/

Applications-Level Tuning: Compilation of Best Practices

- MPI runtime has many parameters
- Tuning a set of parameters can help you to extract higher performance
- Compiled a list of such contributions through the MVAPICH Website
  - http://mvapich.cse.ohio-state.edu/best_practices/
- Initial list of applications
  - Amber
  - HoomDBlue
  - HPCG
  - Lulesh
  - MILC
  - Neuron
  - SMG2000
- Soliciting additional contributions, send your results to mvapich-help at cse.ohio-state.edu.
- We will link these results with credits to you.

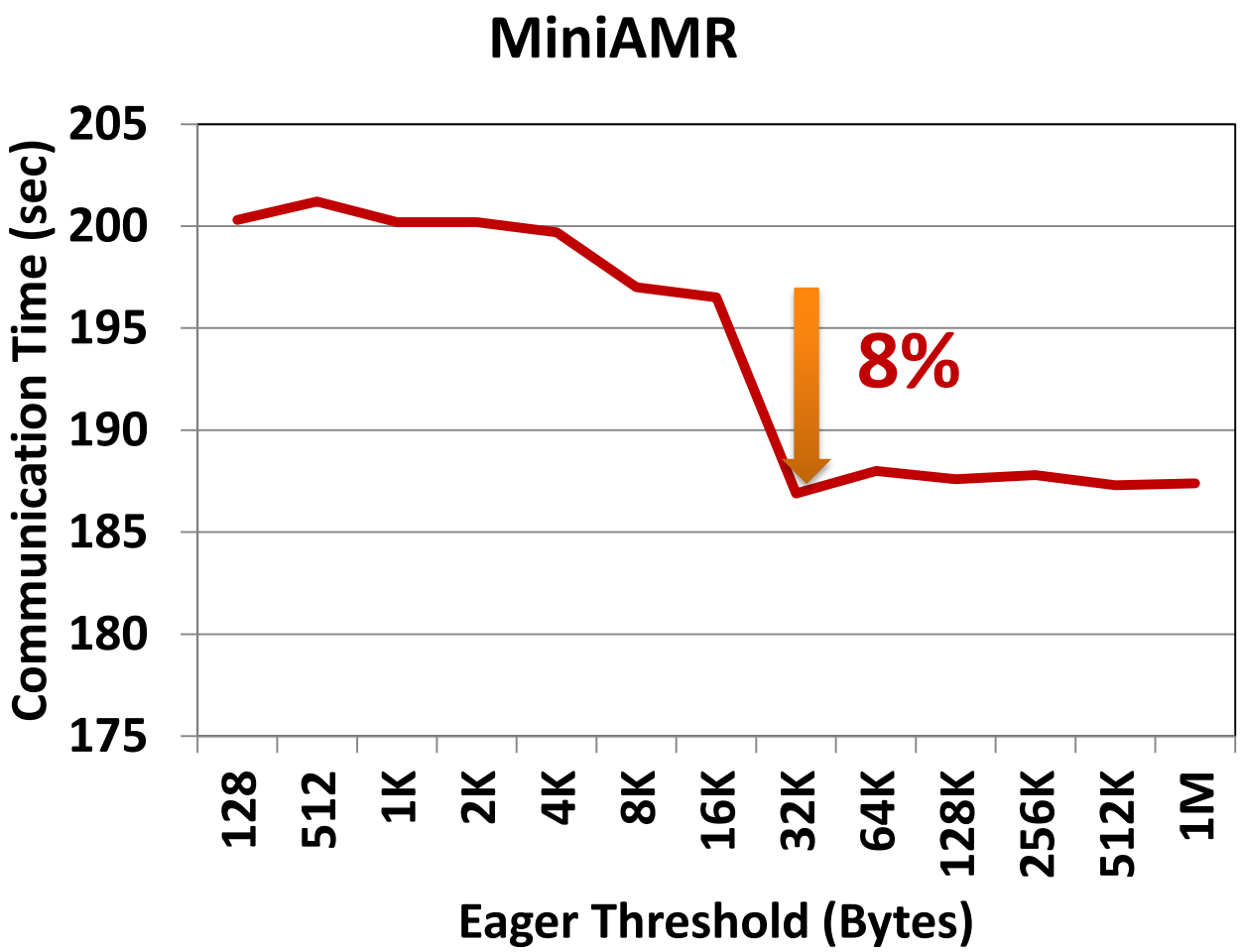
Amber: Impact of Tuning Eager Threshold



Data Submitted by: Dong Ju Choi @ UCSD

- Tuning the Eager threshold has a significant impact on application performance by avoiding the synchronization of rendezvous protocol and thus yielding better communication computation overlap
- 19% improvement in overall execution time at 256 processes
- Library Version: MVAPICH2 2.2
- MVAPICH Flags used
  - MV2_IBA_EAGER_THRESHOLD=131072
  - MV2_VBUF_TOTAL_SIZE=131072
- Input files used
  - Small: MDIN
  - Large: PMTOP

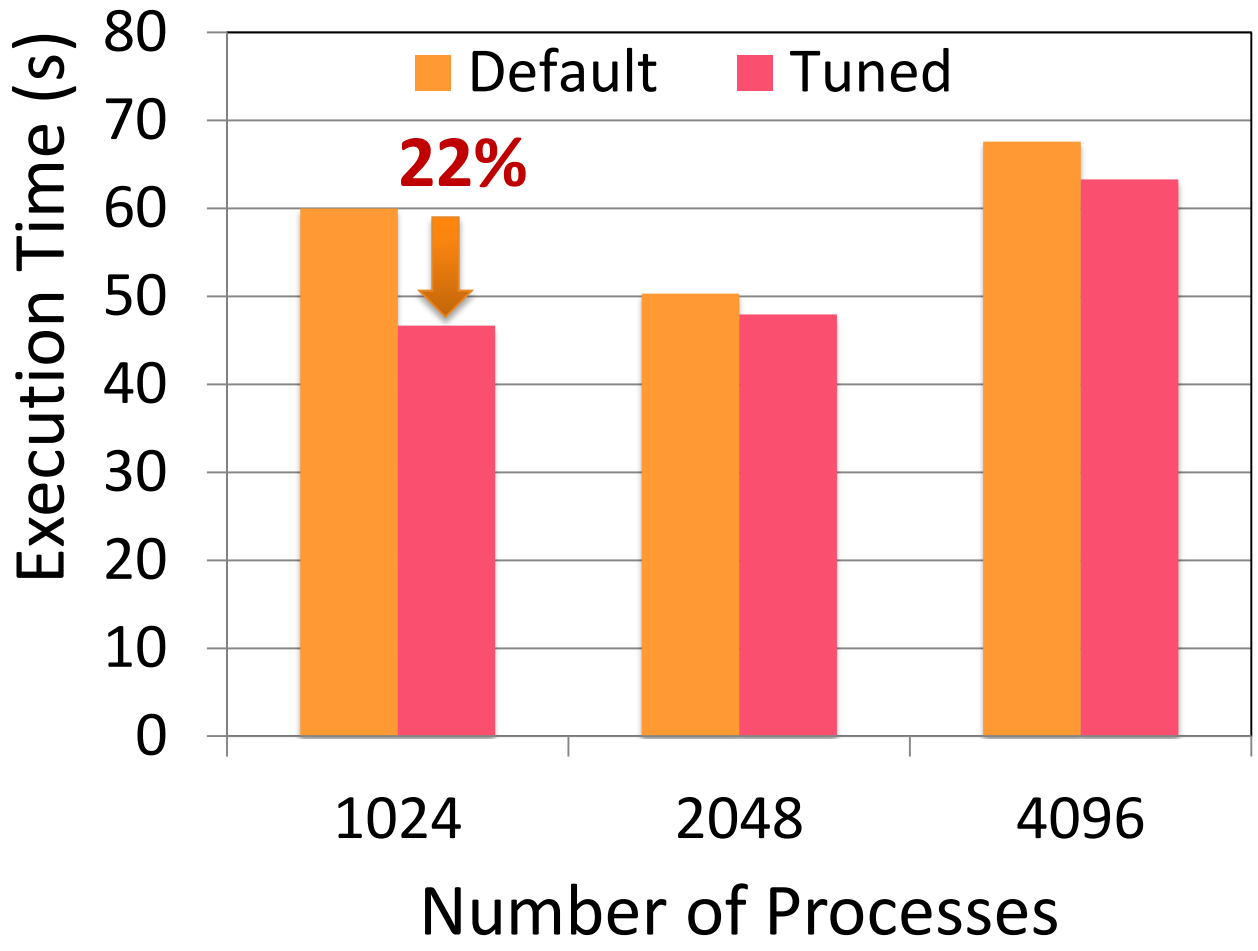
MiniAMR: Impact of Tuning Eager Threshold



- Tuning the Eager threshold has a significant impact on application performance by avoiding the synchronization of rendezvous protocol and thus yielding better communication computation overlap
- 8% percent reduction in total communication time
- Library Version: MVAPICH2 2.2
- MVAPICH Flags used
  - MV2_IBA_EAGER_THRESHOLD=32768
  - MV2_VBUF_TOTAL_SIZE=32768

Data Submitted by Karen Tomko @ OSC and Dong Ju Choi @ UCSD

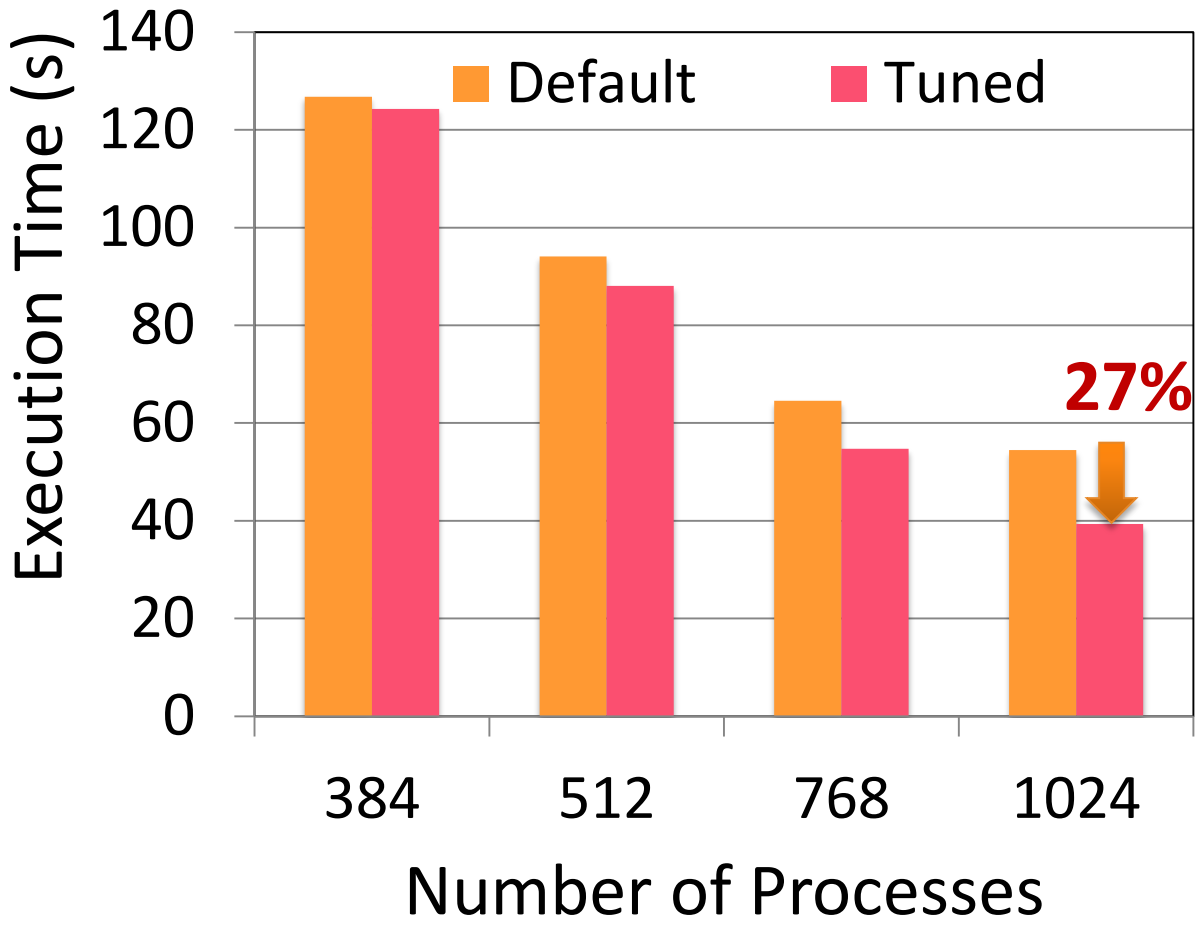
SMG2000: Impact of Tuning Transport Protocol



Data Submitted by Jerome Vienne @ TACC

- UD-based transport protocol selection benefits the SMG2000 application
- 22% and 6% on 1,024 and 4,096 cores, respectively
- Library Version: MVAPICH2 2.1
- MVAPICH Flags used
  - MV2_USE_ONLY_UD=1
- System Details
  - Stampede@ TACC
  - Sandybridge architecture with dual 8-cores nodes and ConnectX-3 FDR network

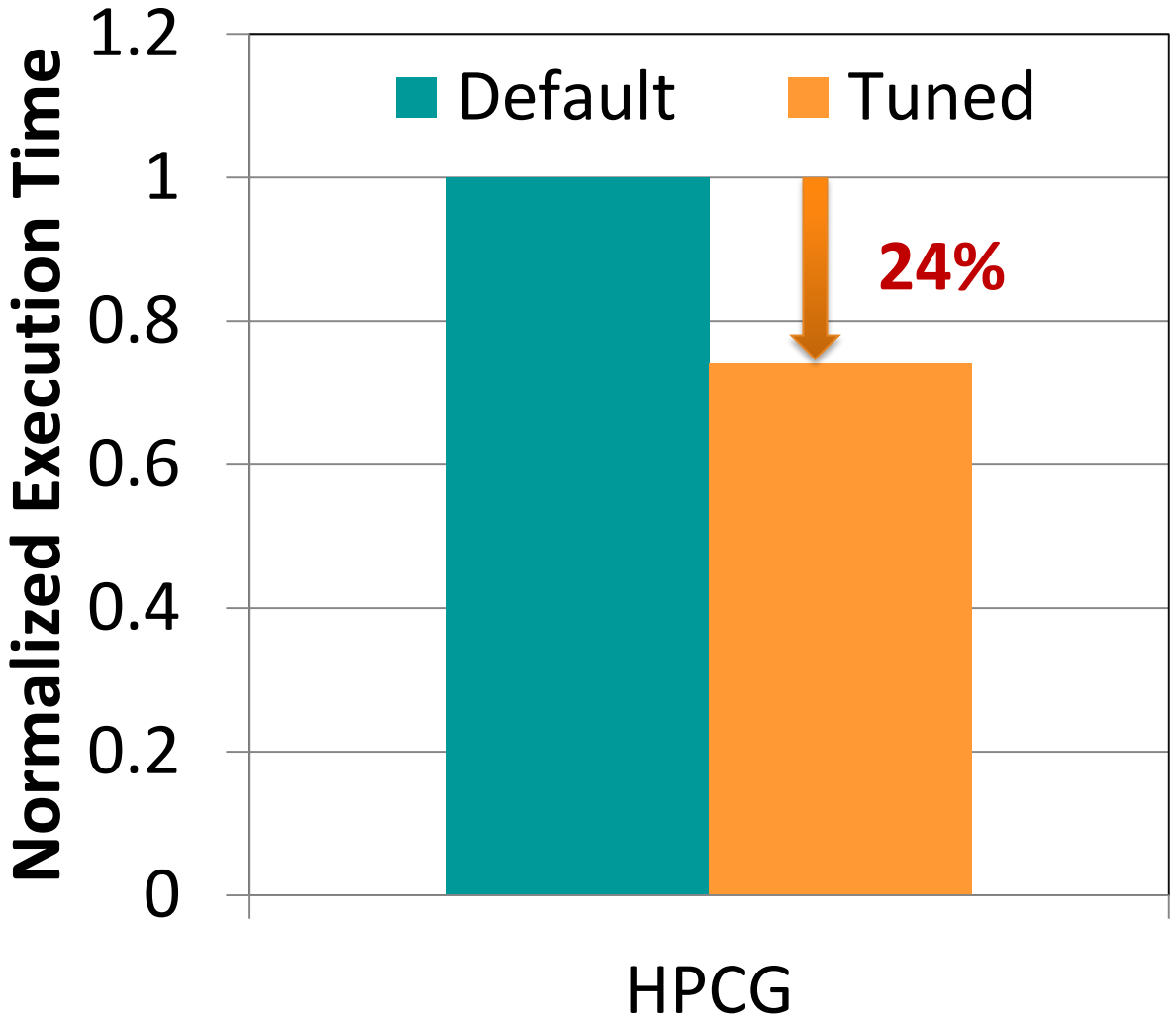
Neuron: Impact of Tuning Transport Protocol



Data Submitted by Mahidhar Tatineni @ SDSC

- UD-based transport protocol selection benefits the SMG2000 application
- 15% and 27% improvement is seen for 768 and 1,024 processes respectively
- Library Version: MVAPICH2 2.2
- MVAPICH Flags used
  - MV2_USE_ONLY_UD=1
- Input File
  - YuEtAl2012
- System Details
  - Comet@SDSC
  - Haswell nodes with dual 12-cores socket per node and Mellanox FDR (56 Gbps) network.

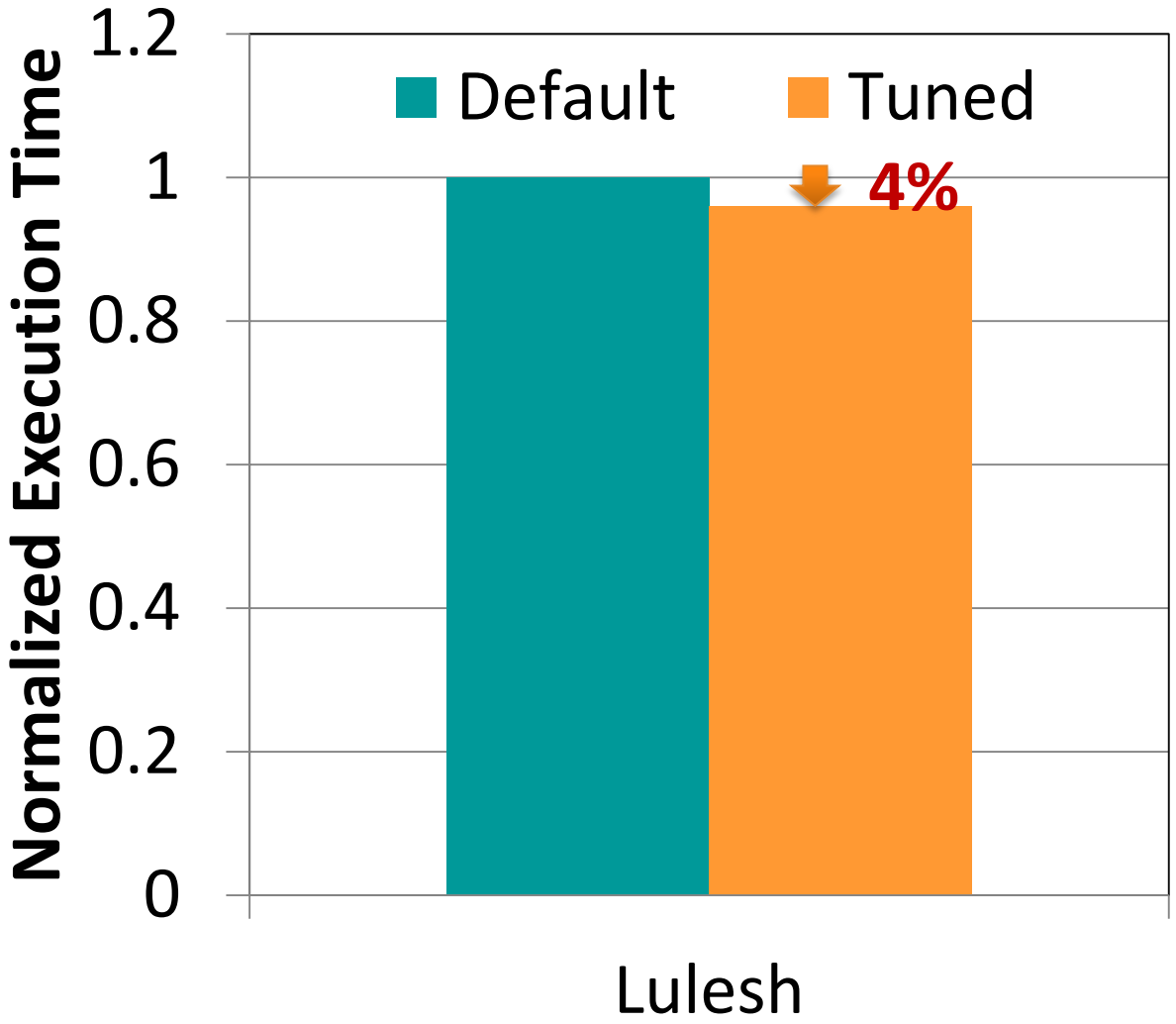
HPCG: Impact of Collective Tuning for MPI+OpenMP Programming Model



- Partial subscription nature of hybrid MPI+OpenMP programming requires a new level of collective tuning
  - For PPN=2 (Processes Per Node), the tuned version of MPI_Reduce shows 51% improvement on 2,048 cores
- 24% improvement on 512 cores
  - 8 OpenMP threads per MPI processes
- Library Version: MVAPICH2 2.1
- MVAPICH Flags used
  - The tuning parameters for hybrid MPI+OpenMP programming models is on by default from MVAPICH2-2.1 onward
- System Details
  - Stampede@ TACC
  - Sandybridge architecture with dual 8-cores nodes and ConnectX-3 FDR network

Data Submitted by Jerome Vienne and Carlos Rosales-Fernandez @ TACC

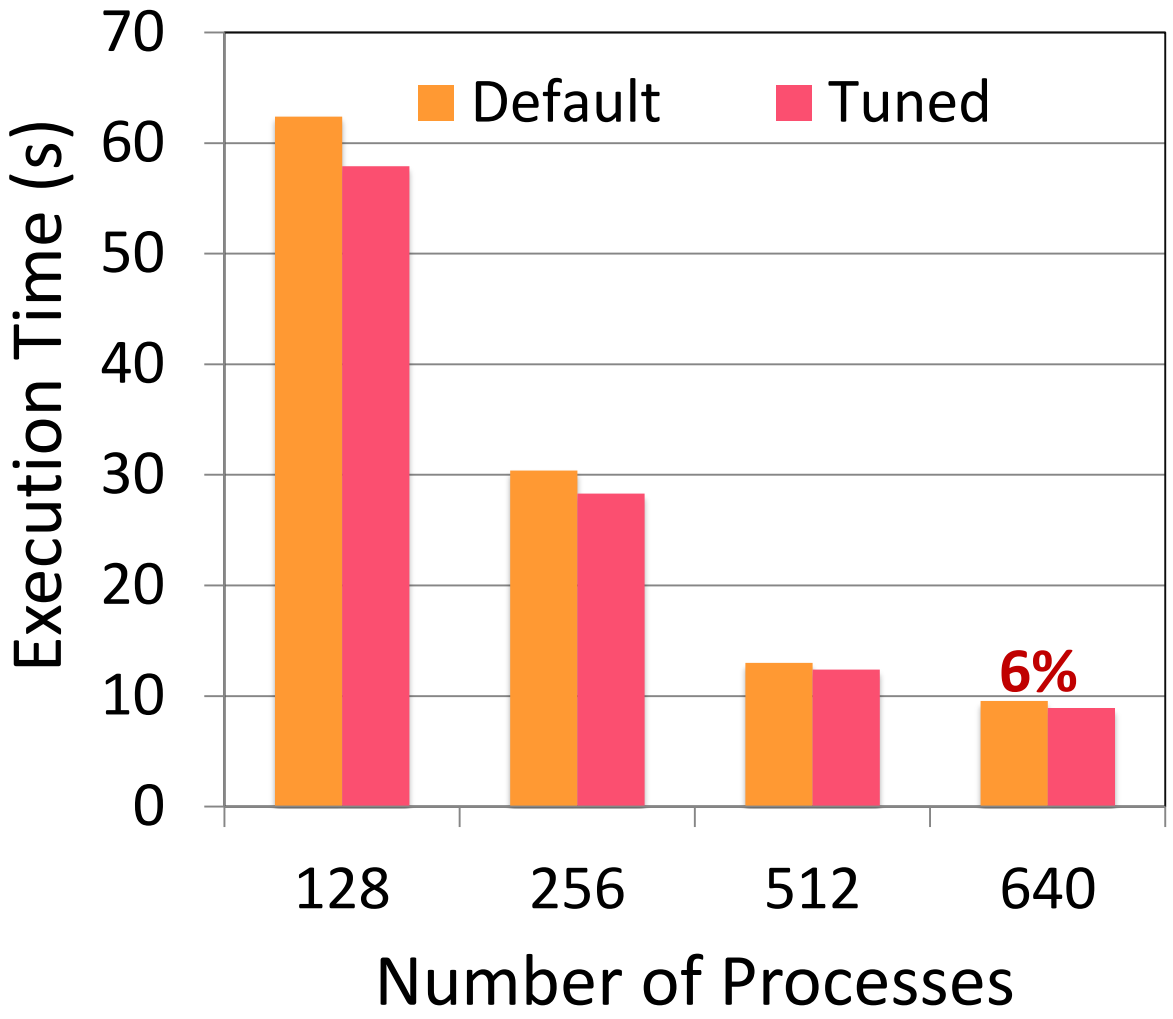
LULESH: Impact of Collective Tuning for MPI+OpenMP Programming Model



- Partial subscription nature of hybrid MPI+OpenMP programming requires a new level of collective tuning
  - For PPN=2 (Processes Per Node), the tuned version of MPI_Reduce shows 51% improvement on 2,048 cores
- 4% improvement on 512 cores
  - 8 OpenMP threads per MPI processes
- Library Version: MVAPICH2 2.1
- MVAPICH Flags used
  - The tuning parameters for hybrid MPI+OpenMP programming models is on by default from MVAPICH2-2.1 onward
- System Details
  - Stampede@ TACC
  - Sandybridge architecture with dual 8-cores nodes and ConnectX-3 FDR network

Data Submitted by Jerome Vienne and Carlos Rosales-Fernandez @ TACC

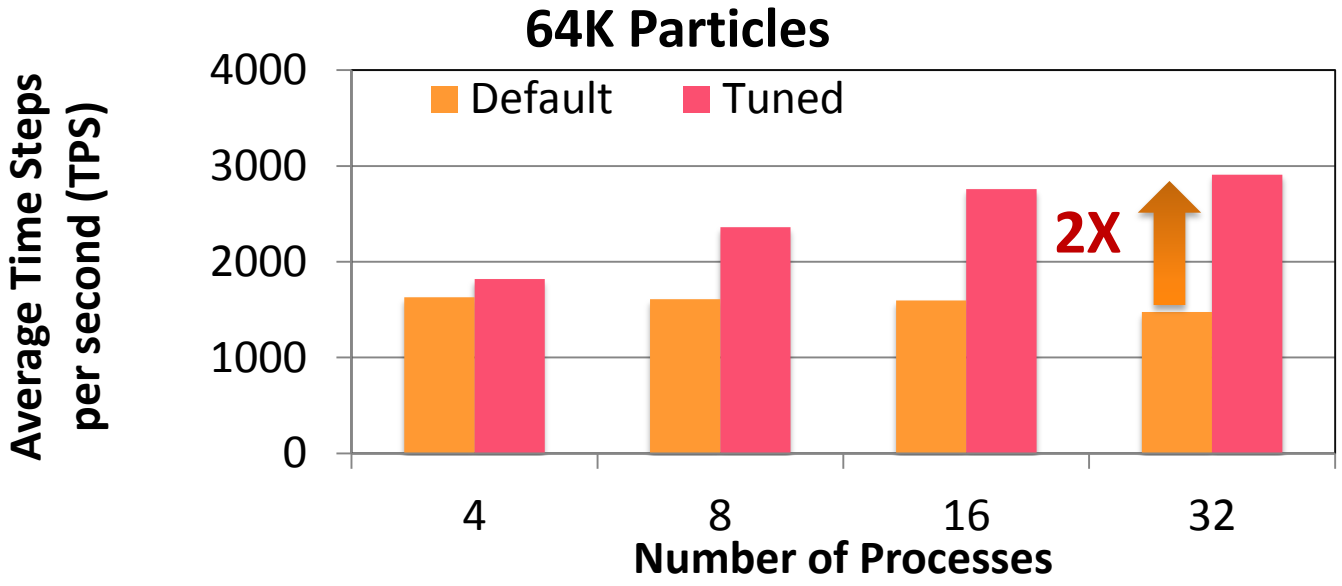
MILC: Impact of User-mode Memory Registration (UMR) based tuning

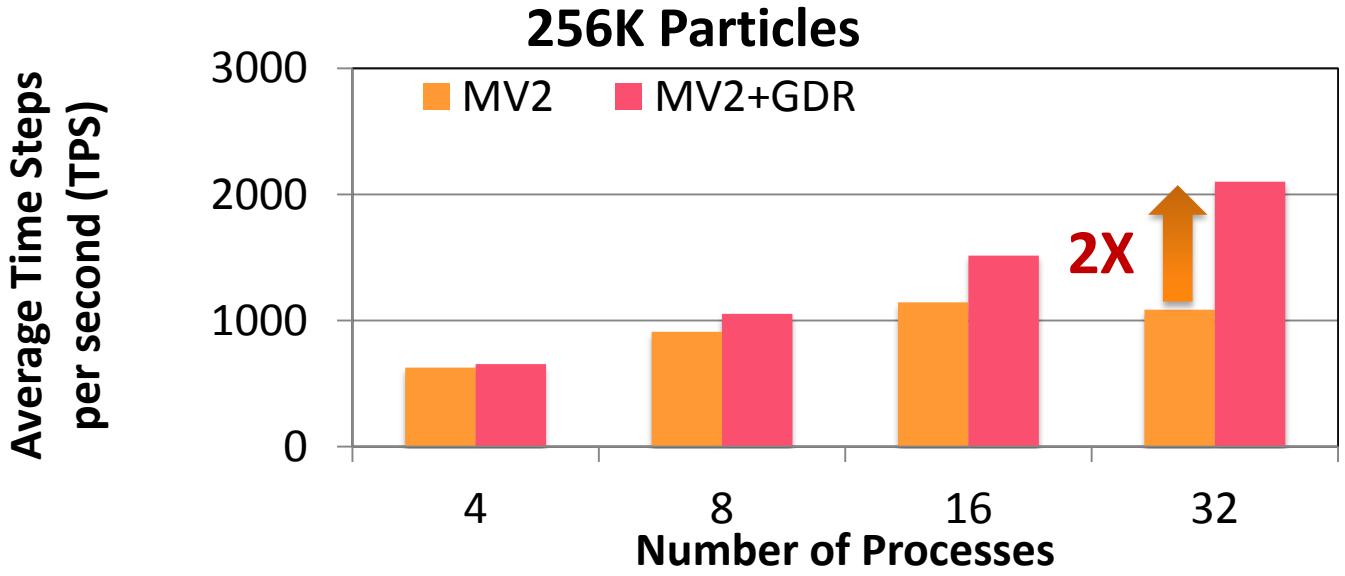


Data Submitted by Mingzhe Li @ OSU

- Non-contiguous data processing is very common on HPC applications. MVAPICH2 offers efficient designs for MPI Datatype support using novel hardware features such as UMR
- UMR-based protocol selection benefits the MILC application.
  - 4% and 6% improvement in execution time at 512 and 640 processors, respectively
- Library Version: MVAPICH2-X 2.2
- MVAPICH Flags used
  - MV2_USE_UMR=1
- System Details
  - The experimental cluster consists of 32 Ivy Bridge Compute nodes interconnected by Mellanox FDR.
  - The Intel Ivy Bridge processors consist of Xeon dual ten-core sockets operating at 2.80GHz with 32GB RAM and Mellanox OFED version 3.2-1.0.1.1.

HOOMD-blue: Impact of GPUDirect RDMA Based Tuning





Data Submitted by Khaled Hamidouche @ OSU

- HOOMD-blue is a Molecular Dynamics simulation using a custom force field.
- GPUDirect specific features selection and tuning significantly benefit the HOOMD-blue application. We observe a factor of 2X improvement on 32 GPU nodes, with both 64K and 256K particles
- Library Version: MVAPICH2-GDR 2.2
- MVAPICH-GDR Flags used
  - MV2_USE_CUDA=1
  - MV2_USE_GPUDIRECT=1
  - MV2_GPUDIRECT_GDRCOPY=1
- System Details
  - Wilkes@Cambridge
  - 128 Ivybridge nodes, each node is a dual 6-cores socket with Mellanox FDR

MVAPICH2 – Plans for Exascale

- Performance and Memory scalability toward 1-10M cores
- Hybrid programming (MPI + OpenSHMEM, MPI + UPC, MPI + CAF …)
  - MPI + Task*
- Enhanced Optimization for GPU Support and Accelerators
- Taking advantage of advanced features of Mellanox InfiniBand
  - Tag Matching*
  - Adapter Memory*
- Enhanced communication schemes for upcoming architectures
  - Knights Landing with MCDRAM*
  - NVLINK*
  - CAPI*
- Extended topology-aware collectives
- Extended Energy-aware designs and Virtualization Support
- Extended Support for MPI Tools Interface (as in MPI 3.0)
- Extended FT support
- Support for * features will be available in future MVAPICH2 Releases

Thank You!

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

The MVAPICH2 Project
http://mvapich.cse.ohio-state.edu/