NETWORK-BASED COMPUTING LABORATORY

Designing and Evaluating MPI-2 Dynamic Process Management Support for InfiniBand

Tejus Gangadharappa, Matthew Koop and Dhabaleswar. K. (DK) Panda

Computer Science & Engineering Department
The Ohio State University

OHIO STATE

Outline

- Motivation and Problem Statement
- Dynamic Process Interface design
- Designing the Benchmark-suite
- Experimental results
- Future Work and Conclusions

Introduction

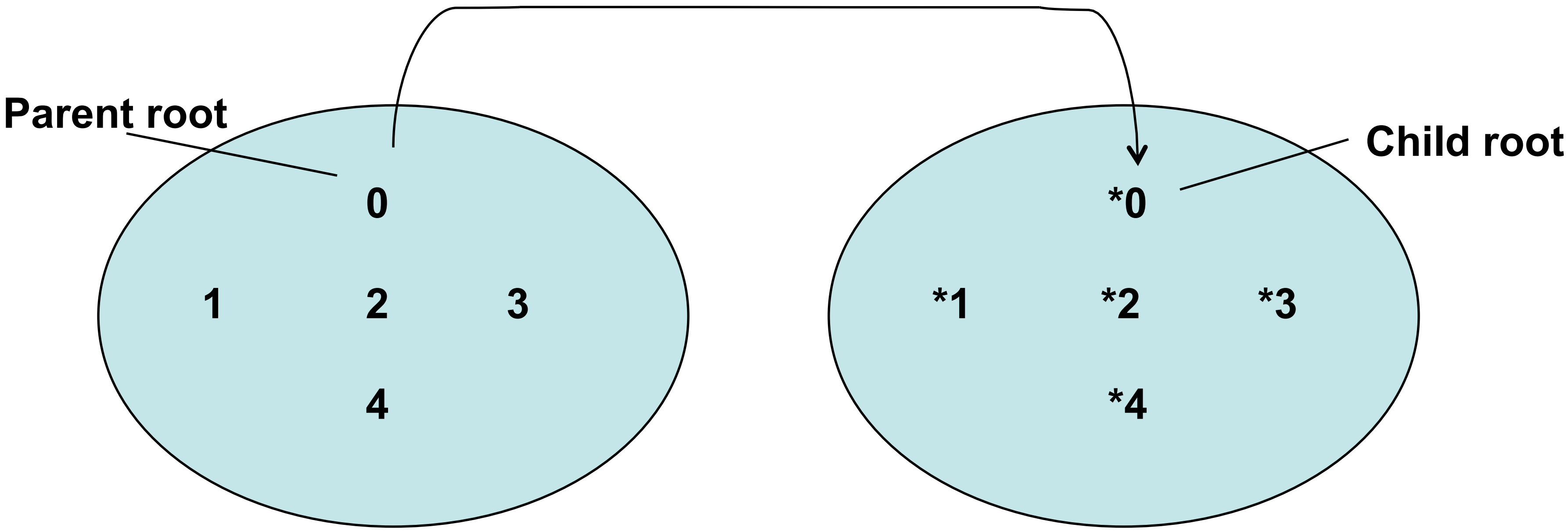
- Large scale multi-core clusters are becoming increasingly common
- MPI is the de-facto programming model for HPC
- The MPI-1 specification required the number of processes in a job to be fixed at job launch
- Dynamic Process Management (DPM) feature was introduced in MPI-2 to address this limitation

Dynamic Process Management Interface

- Applications can use the DPM interface to spawn new processes at run-time depending on compute node availability
- Beneficial for
  - Multi-scale modeling applications
  - Applications based on master/slave paradigm
- MPI offers two types of communicator objects
  - intra-communicator and inter-communicator
- The DPM interface uses an inter-communicator object for communication between the original process set and the spawned process set

Dynamic Process Interface

Inter-Communicator Creation



Parent root

0

1 2 3

4

Child root

*0

*1 *2 *3

*4

Initial Process
group

Spawned Process
group

InfiniBand

- Almost 30% of the TOP500 Supercomputers use InfiniBand as the high-speed interconnect
- Provides
  - Low latency (~1.0 microsec)
  - High bandwidth (~3.0 Gigabytes/sec unidirectional with QDR)
- Necessary to have MPI implementations that offer efficient dynamic process support over InfiniBand

InfiniBand (Cont'd)

- Remote DMA (RDMA) Operations
- Supports atomic operations
- Offers four transport modes
  - Reliable Connection (RC)
  - Unreliable Datagram (UD)
  - Reliable Datagram (RD)
  - Unreliable Connection (UC)
- Trade-off between network reliability, memory footprint and processing overheads

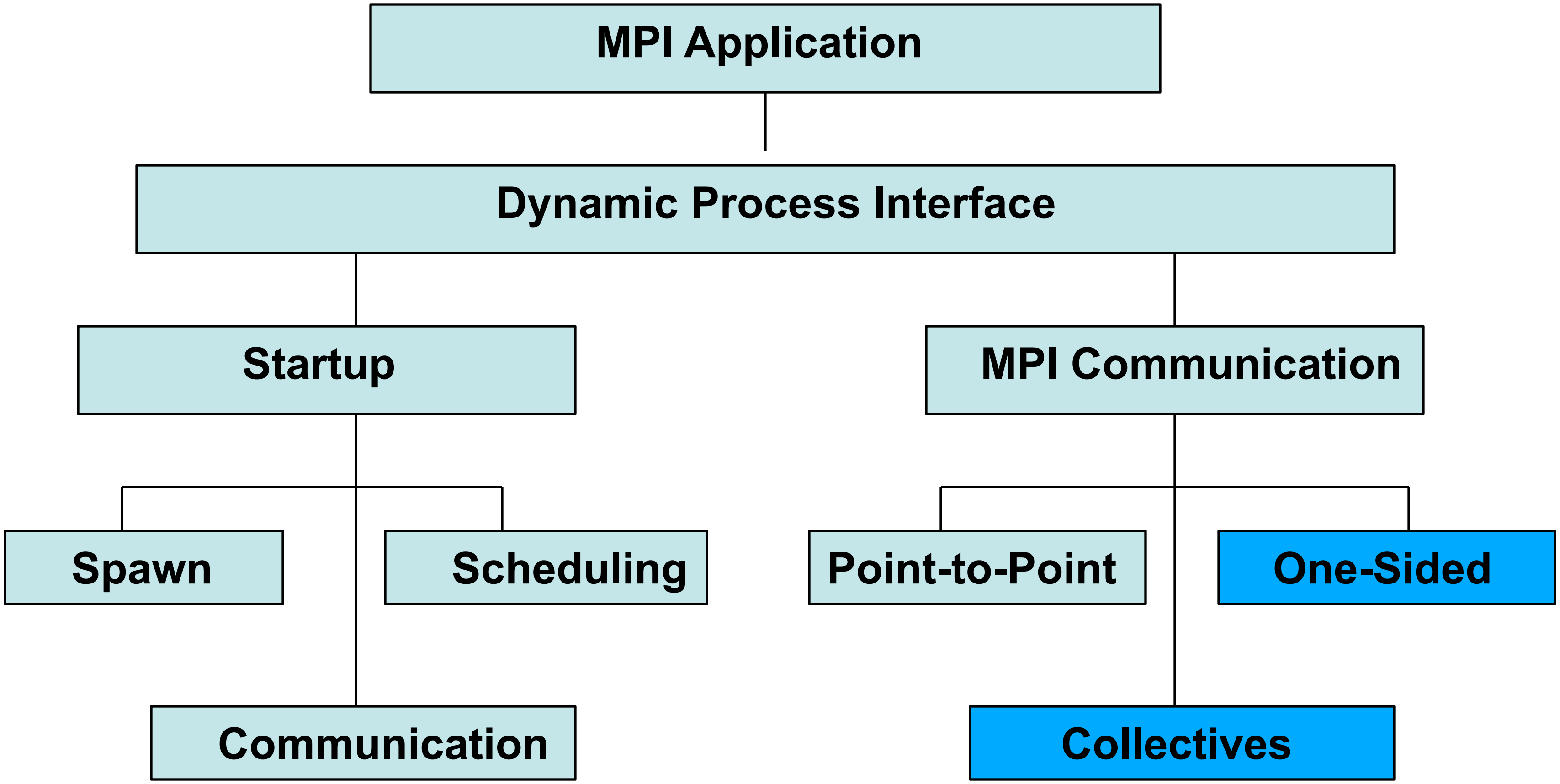
Problem Statement

- What are the challenges involved in designing dynamic process support over InfiniBand networks?
- What is the overhead of having a dynamic process interface?
- How do the InfiniBand transport modes (RC and UD) impact the performance of the dynamic process interface?
- Can we design a benchmark-suite to evaluate the performance of the dynamic process interface over InfiniBand?

Outline

- Motivation and Problem Statement
- Dynamic Process Interface design
- Designing the Benchmark-suite
- Experimental results
- Future Work and Conclusions

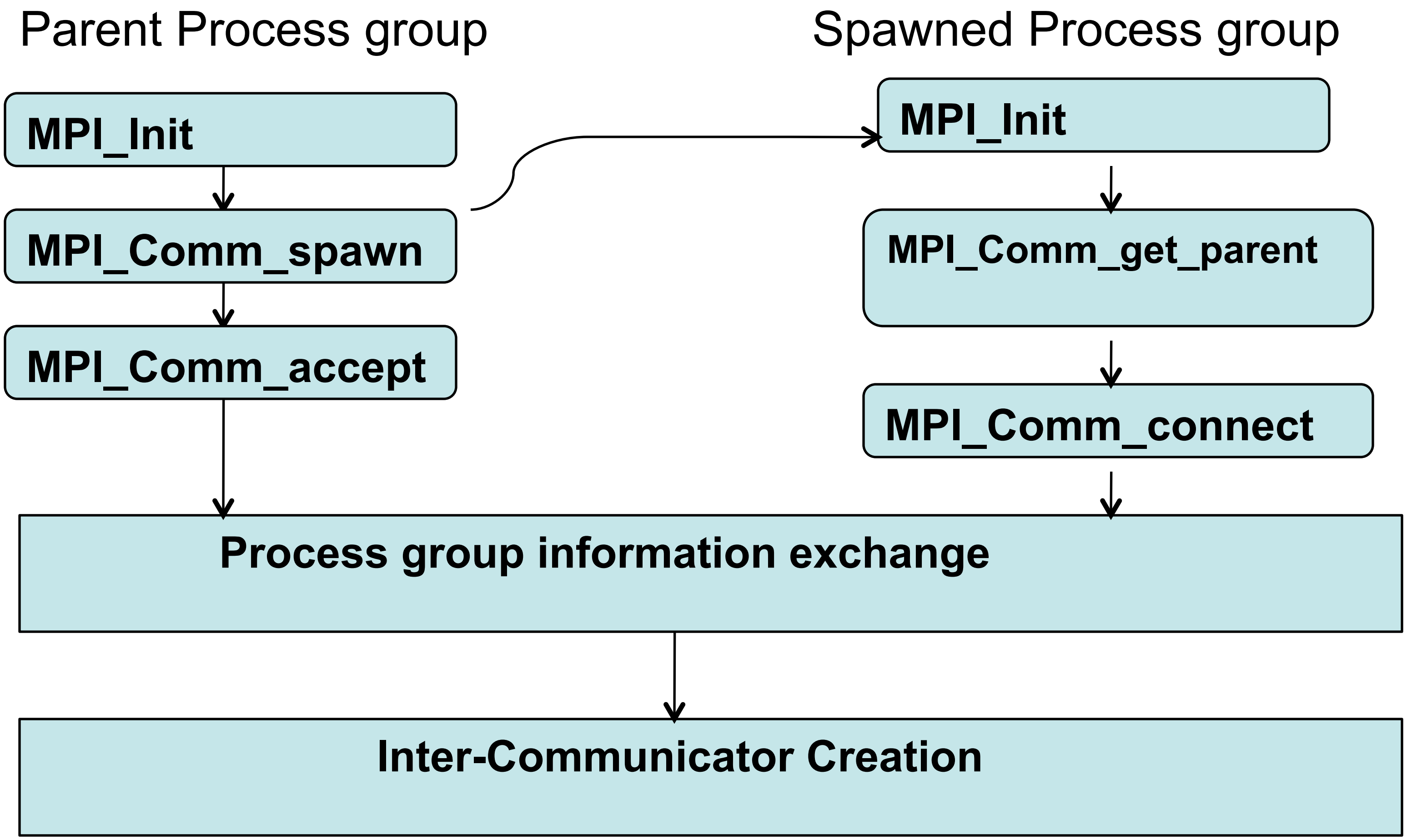
Dynamic Process Interface Design



Startup Component – Spawn and Scheduling

- Applications interact with the job launcher tool over the management network during the spawn phase
- Two job launchers considered
  - Multi-Purpose Daemon (MPD)
  - Mpirun_rsh (a scalable job launching framework)
- Scheduling and mapping the dynamically spawned processes is critical to the performance of the application
- Two allocations (block and cyclic) considered

Startup Component – Communication



Startup Component – Communication

- Connection establishment overhead for each spawn
- Design choices for inter-communicator setup
  - RC and UD transport modes
- UD mode has less overhead
  - Reliability needs to be added
  - Desirable for applications spawning small process groups and frequently
- RC mode has little higher overhead
  - Provides reliability
  - Desirable for large and infrequent spawns

Outline

- Motivation and Problem Statement
- Dynamic Process Interface design
- Designing the Benchmark-suite
- Experimental results
- Future Work and Conclusions

Spawn Latency Benchmark

- Measures the average time spent in the MPI_Comm_Spawn routine at the parent-root process
- Necessary to minimize the overhead of spawning new jobs as it has a significant impact on the overall application performance
- Benchmark has provision to change
  - size of the parent communicator
  - size of the spawned child communicator

Spawn Rate Benchmark

- Measures the rate at which an MPI implementation can perform the MPI_Comm_Spawn operation
- The spawn rate metric gives insights into how frequently MPI processes can spawn

Inter-Communicator Point-to-Point Latency Benchmark

- Average time required to exchange data between processes over an inter-communicator
- Inter-communicator message delivery involves mapping from local process group to the remote process group
- If connections are setup on-demand, this benchmark captures both the connection establishment and the message exchange steps
- Inter-Communicator point-to-point exchanges are critical to the performance of the applications

Implementation

- Proposed designs have been implemented in MVAPICH2 1.4
- MVAPICH/MVAPICH2
  - Open-source MPI project for InfiniBand and 10GigE/iWARP
  - Empowers many TOP500 systems
  - Used by more than 975 organizations in 51 countries
  - Available as a part of OFED and from many vendors and Linux Distributions (RedHat, SuSE, etc.)
  - http://mvapich.cse.ohio-state.edu
- Micro-benchmarks were implemented as a part of the OSU MPI micro-benchmarks (OMB)
  - http://mvapich/cse.ohio-state.edu/benchmarks/

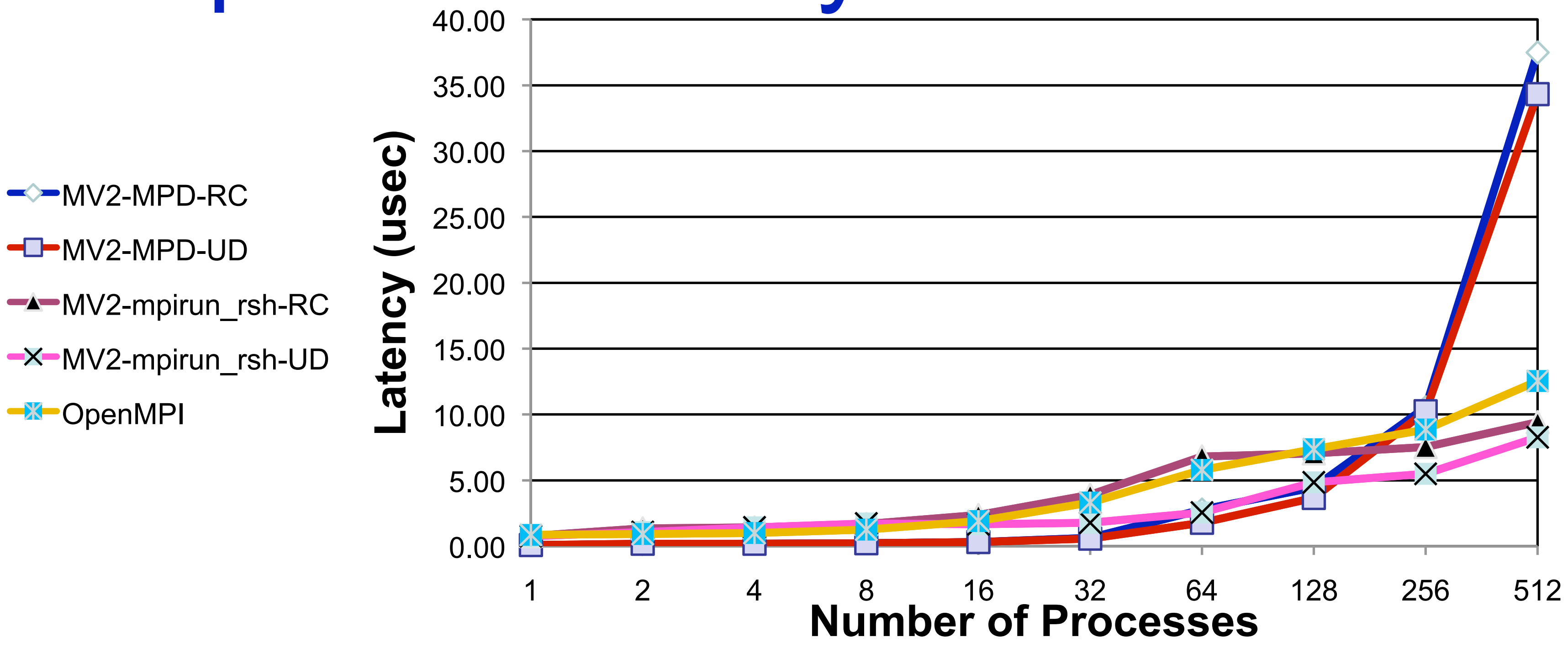
Outline

- Motivation and Problem Statement
- Dynamic Process Interface design
- Designing the Benchmark-suite
- Experimental results
- Future Work and Conclusions

Experimental Setup

- 64-node Intel Clovertown cluster
- Each node has
  - 8 cores and 6GB RAM
- Evaluations up to 512 cores
- InfiniBand Double Data Rate (DDR)
- MVAPICH2 1.4RC1 and OpenMPI 1.3

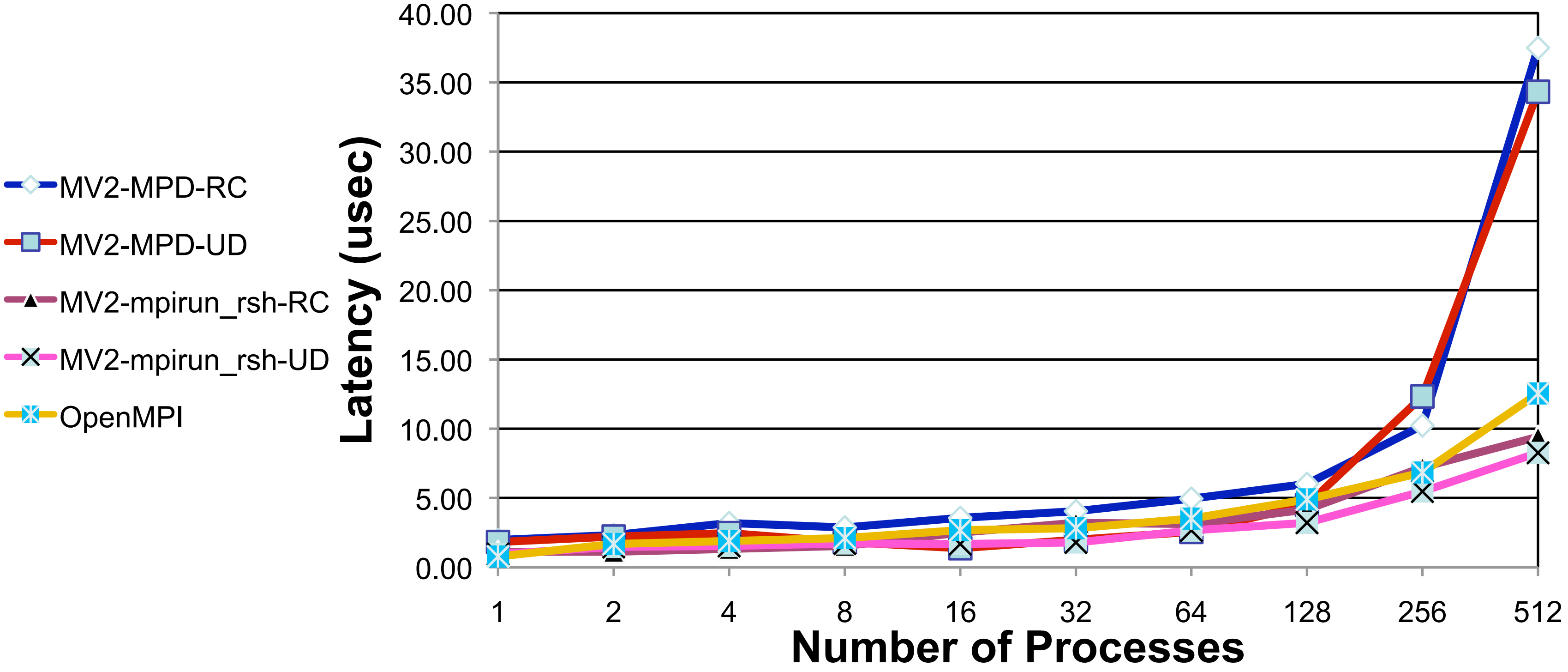
Spawn Latency Benchmark



Cyclic Rank Allocation

- UD design shows benefit beyond job size of 32
- MPD startup mechanism is faster than mpirun_rsh for small job size, however mpirun_rsh performs better as job size increases
- Up to 128 processes, MV2-mpirun_rsh-RC and OpenMPI perform similarly
- For > 128 processes, MV2-mpirun_rsh-UD performs the best

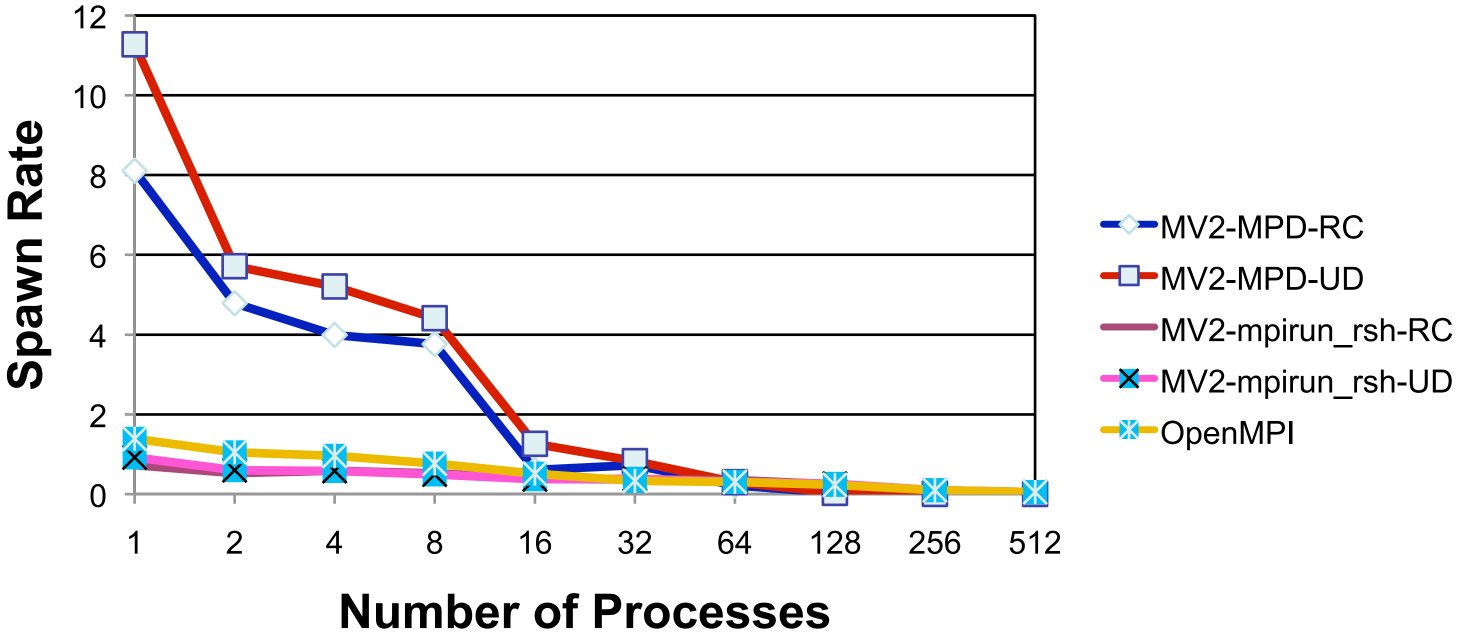
Spawn Latency Benchmark



Block Rank Allocation

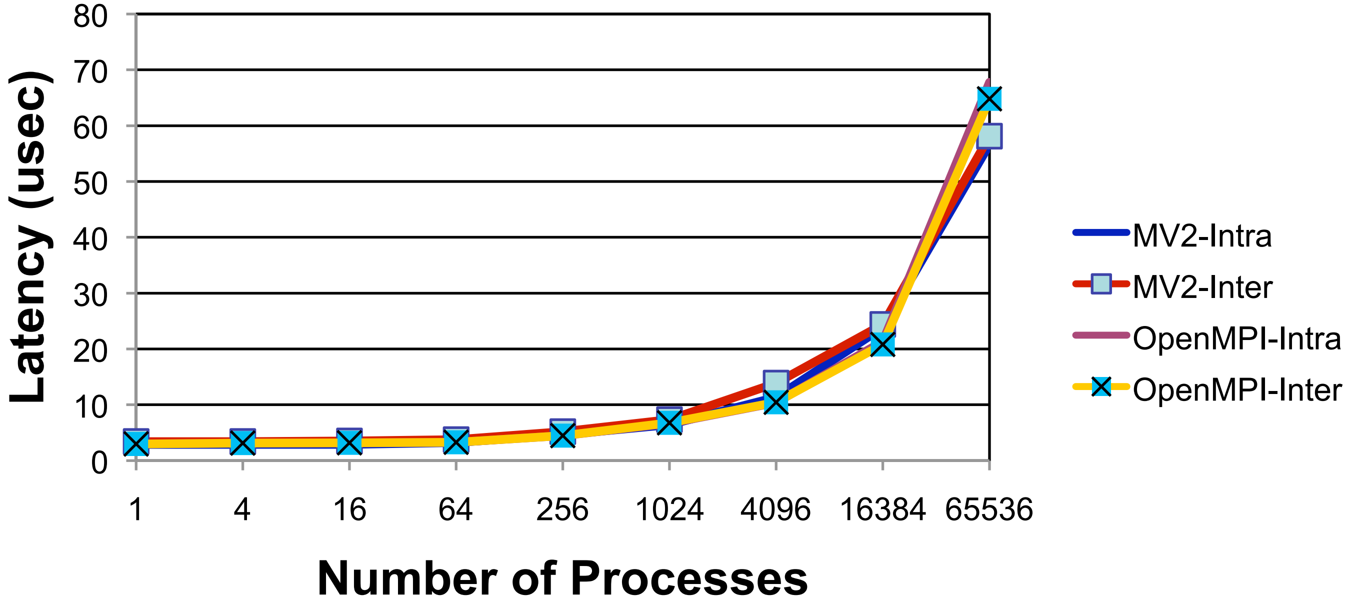
- Block allocation of ranks shows the effect of HCA contention on spawn time
- The UD-based design performs better due to lesser overhead
- MV2-mpirun_rsh-UD design performs the best

Spawn Rate Benchmark



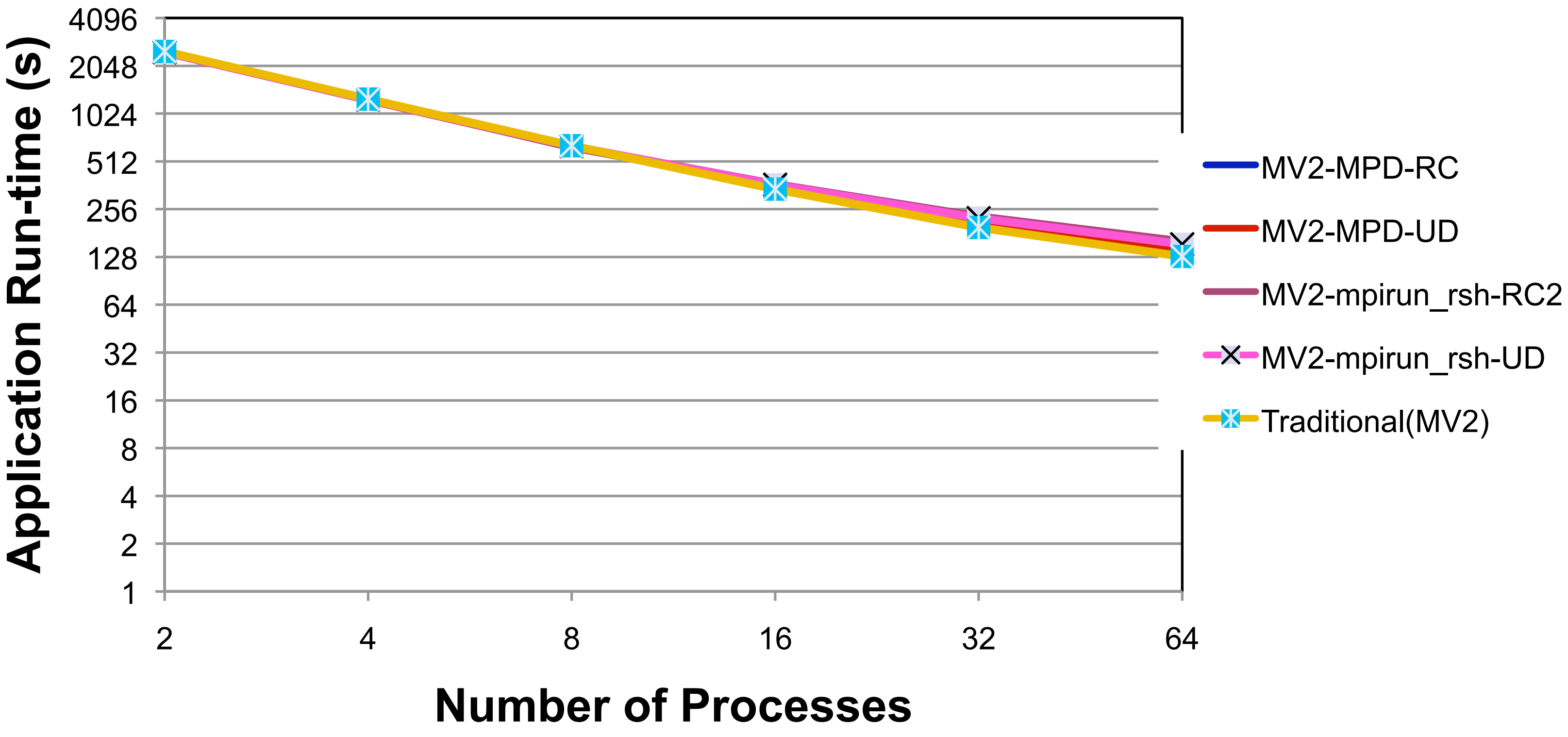
- UD designs provide better spawn rates than RC ones because of the higher cost of creating and destroying RC queue pairs
- MPD designs provide higher spawn rates than mpirun_rsh for small jobs due to the higher initial overhead in the later case
- Mpirun_rsh scales very well and maintains a steady spawn rate with increasing job size.

Inter-Communicator Point-to-Point Latency Benchmark



- Performance is very similar for small messages
- Performance differs in the medium message length (depends on rendezvous threshold values)
- For large messages (64K), MV2 delivers better performance

Parallel POV-Ray Evaluation



- Re-designed a dynamic process version of the POV-Ray application
- Render a 3000x3000 glass chess board with global illumination
- The dynamic process framework adds very little overhead

Software Distribution

- The new DPM support is available with MVAPICH2 1.4
  - Latest version is MVAPICH2 1.4RC2
  - Downloadable from http://mvapich.cse.ohio-state.edu
- Micro-benchmarks will be available as a part of OSU MPI Micro-benchmarks (OMB) in the near future

Conclusions & Future Work

- Presented alternative designs for DPM interface on InfiniBand
- Proposed new benchmarks to evaluate DPM designs
- MPD based framework is suitable for frequent small spawns
- Mpirun_rsh based startup is recommended for large infrequent spawns
- DPM interface has very little overhead on the application performance

Future Work:

- Explore a hybrid model that switches between UD and RC modes based on job size
- Evaluate the performance of collectives and one-sided routines for the dynamic process interface

Thank you !

MVAPICH

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

{gangadha, koop, panda}@cse.ohio-state.edu

Network-Based Computing Laboratory