Argonne NATIONAL LABORATORY

PMI: A Scalable Process-Management Interface for Extreme-Scale Systems

Pavan Balaji, Darius Buntinas, David Goodell, William Gropp, Jayesh Krishna, Ewing Lusk and Rajeev Thakur

U.S. DEPARTMENT OF ENERGY

Introduction

- Process management is integral part of HPC
- Scalability and performance are critical
- Close interaction between process management and parallel library (e.g., MPI) is important
  - Need not be integrated
- Separation allows
  - Independent development and improvement
  - Parallel libraries portable to different environments

PMI

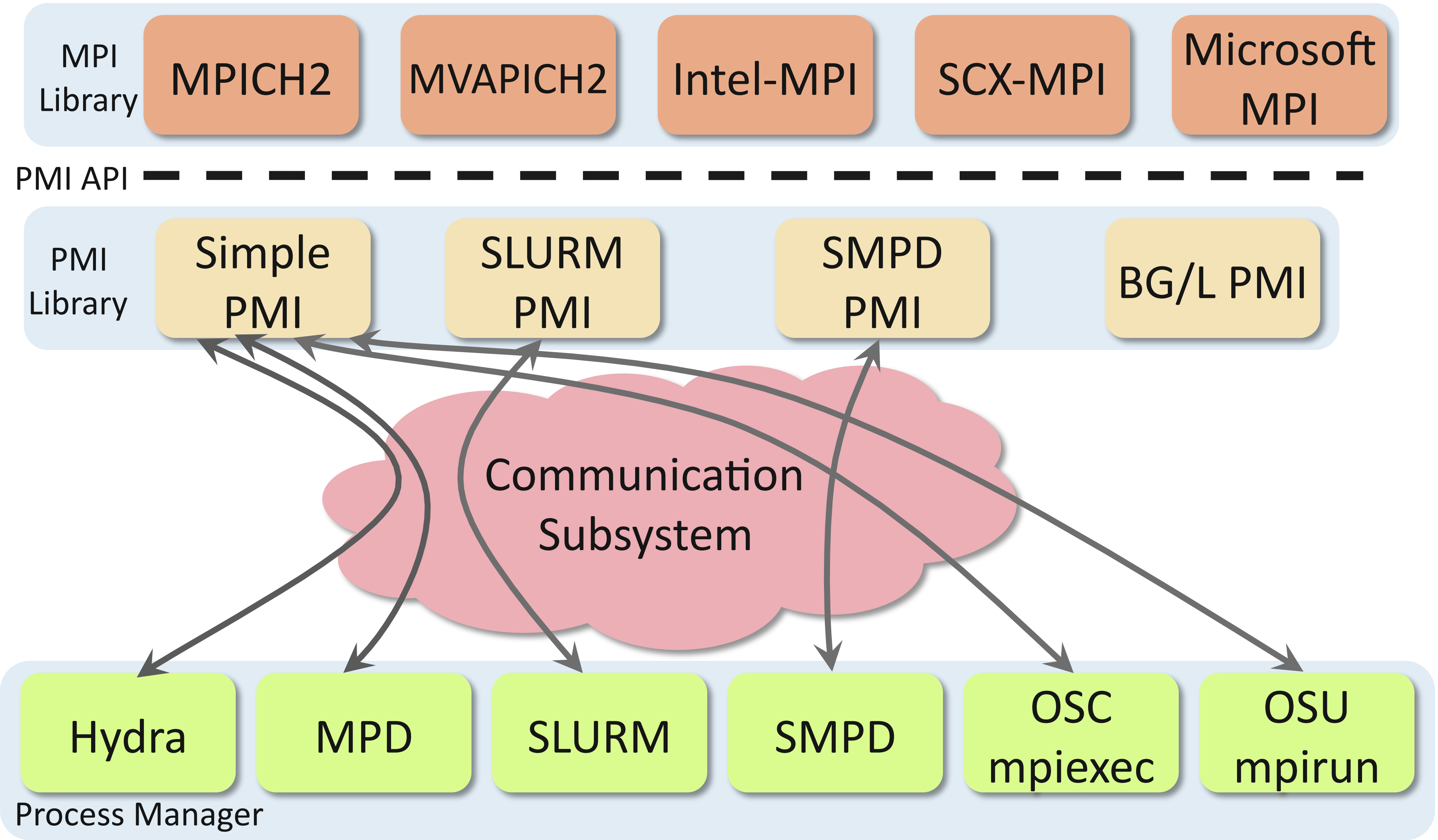
- Generic process management interface for parallel applications
- PMI-1 is widely used
  - MVAPICH, Intel MPI, Microsoft MPI
  - SLURM, OSC mpiexec, OSU mpirun
- Introducing PMI-2
  - Improved scalability
  - Better interaction with hybrid MPI+threads

PMI Functionality

- Process management
  - Launch and monitoring
    - Initial job
    - Dynamic processes
  - Process control
- Information exchange
  - Contact information
  - Environmental attributes

System Model

System Model



MPI Library

MPICH2

MVAPICH2

Intel-MPI

SCX-MPI

Microsoft MPI

PMI API

PMI Library

Simple PMI

SLURM PMI

SMPD PMI

BG/L PMI

Communication Subsystem

Hydra

MPD

SLURM

SMPD

OSC mpiexec

OSU mpirun

Process Manager

Process Manager

- Handles
  - Process launch
    - Start and stop processes
    - Forwarding I/O and signals
  - Information exchange
    - Contact information to set up communication
- Implementation
  - May be separate components
  - May be distributed
- E.g., PBS, Sun Grid Engine, SSH

PMI Library

- Provides interface between parallel library and process manager
- Can be system specific
  - E.g, BG/L uses system specific features
- Wire protocol between PMI library and PM
  - PMI-1 and PMI-2 have specified wire protocols
  - Allows PMI lib to be used with different PM
  - Note: wire protocol and PMI API are separate entities
    - PMI implementation need not have wire protocol

PMI API

- PMI-1 and PMI-2
- Functions associated with
  - Initialization and finalization
    - Init, Finalize, Abort
  - Information exchange
    - Put, Get, Fence
  - Process creation
    - Spawn

Information Exchange

- Processes need to exchange connection info
- PMI uses a Key-Value database (KVS)
- At init, processes *Put* contact information
  - E.g., IP address and port
- Processes *Get* contact info when establishing connections
- Collective *Fence* operation to allow optimizations

Connection Data Exchange Example

- At init
  - Proc 0 *Puts* (key="bc-p0", value="192.168.10.20;3893")
  - Proc 1 *Puts* (key="bc-p1", value="192.168.10.32;2897")
  - Proc 0 and 1 call *Fence*
    - PM can collectively distribute database
- Later Proc 0 wants to send a message to Proc 1
  - Proc 0 does a *Get* of key "bc-p1"
    - Receives value "192.168.10.32;2897"
  - Proc 0 can now connect to Proc 1

Implementation Considerations

- Allow the use of "native" process manager with low overhead
  - Systems often have existing PM
    - E.g., integrated with resource manager
  - Minimize async processing and interrupts
- Scalable data exchange
  - Distributed process manager
  - Collective *Fence* provides opportunity for scalable collective exchange

Second Generation PMI

New PMI-2 Features

- Attribute query functionality
- Database scope
- Thread safety
- Dynamic processes
- Fault tolerance

PMI-2 Attribute Query Functionality

- Process and resource managers have system-specific information
  - Node topology, network topology, etc.
- Without this, processes need to determine this themselves
  - Each process gets each other's contact-info to discover local processes
  - O(p^2) queries

PMI-2 Database Scope

- Previously KVS had only global scope
- PMI-2 adds node-level scoping
  - E.g., keys for shared memory segments
- Allows for optimized storage and retrieval of values

PMI-2 Thread Safety

- PMI-1 is not thread safe
  - All PMI calls must be serialized
    - Wait for request and response
  - Can affect multithreaded programs
- PMI-2 adds thread safety
  - Multiple threads can call PMI functions
  - One call cannot block the completion of another

PMI-2 Dynamic Processes

- In PMI-1 a separate database is maintained for each MPI_COMM_WORLD (process group)
  - Queries are not allowed across databases
  - Requires out-of-band exchange of databases
- PMI-2 allows cross-database queries
  - Spawned or connected process groups can now query each other's databases
  - Only process group ids need to be exchanged

PMI-2 Fault Tolerance

- PMI-1 provides no mechanism for respawning a failed process
  - New processes can be spawned, but they have a unique rank and process group
- Respawn is critical for supporting fault-tolerance
  - Not just for MPI but other programming models

Evaluation and Analysis

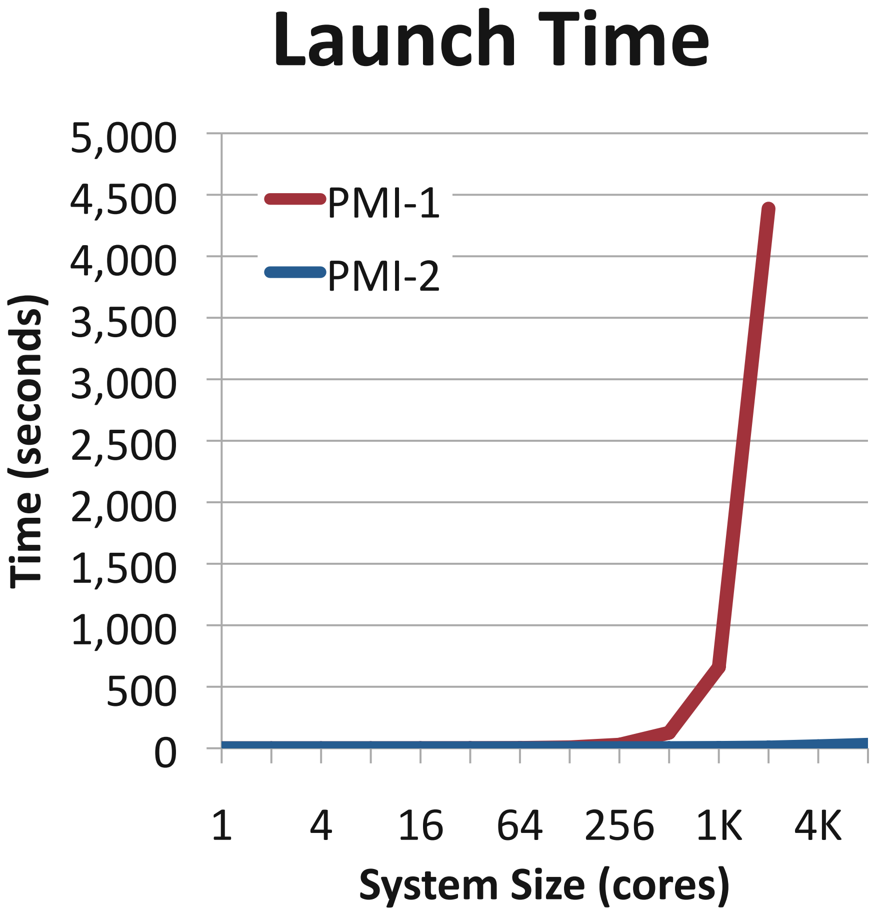
Evaluation and Analysis

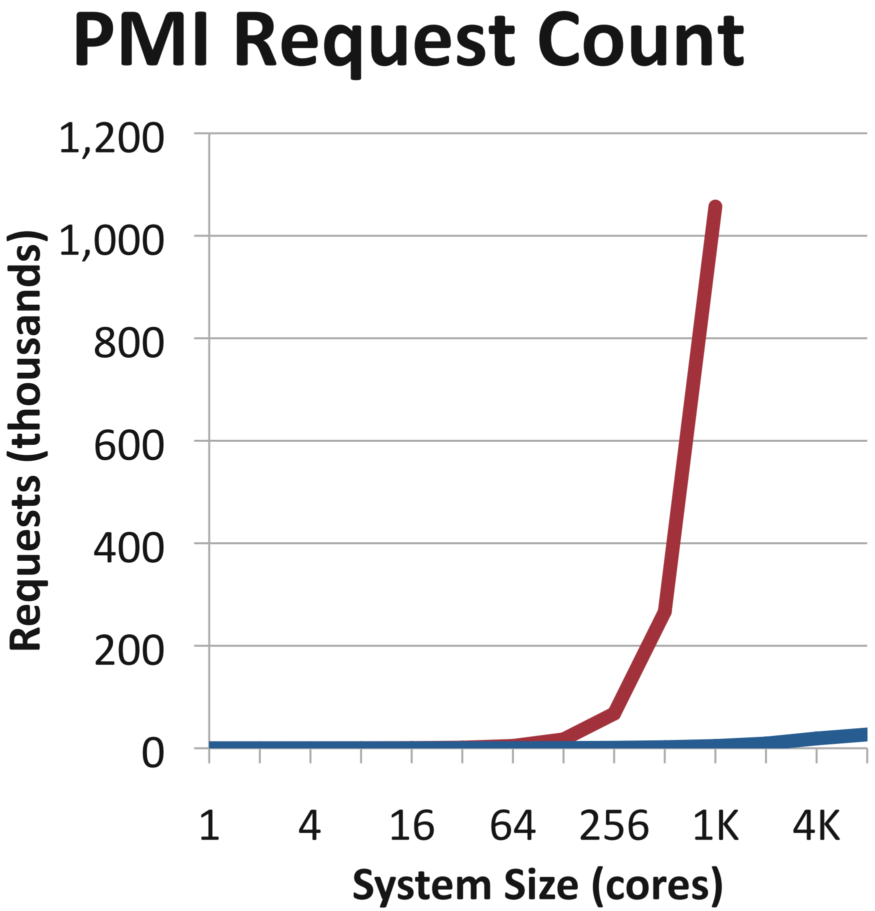
- PMI-2 implemented in Hydra process manager
- Evaluation
  - System information query performance
  - Impact of added PMI functionality over native PM
  - Multithreaded performance

System Information Query Performance

- PMI-1 provides no attribute query functionality
  - Processes must discover local processes
  - O(p^2) queries
- PMI-2 has node topology attribute
- Benchmark (5760 cores on SiCortex)
  - MPI_Init();MPI_Finalize();

Process Launch (5760-core SiCortex)


Launch Time

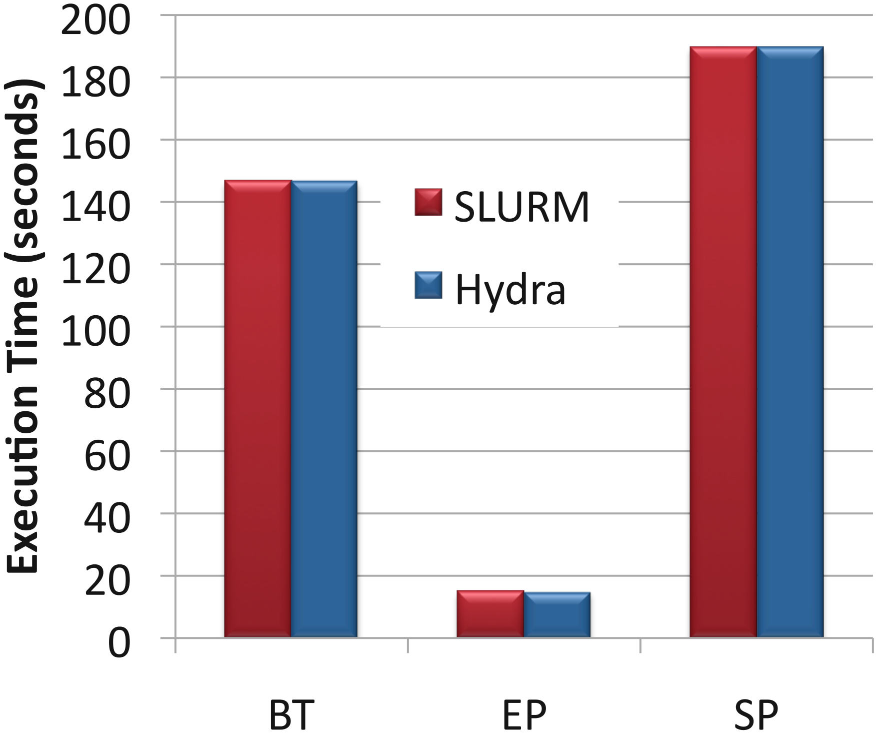

PMI Request Count

Impact of PMI Functionality

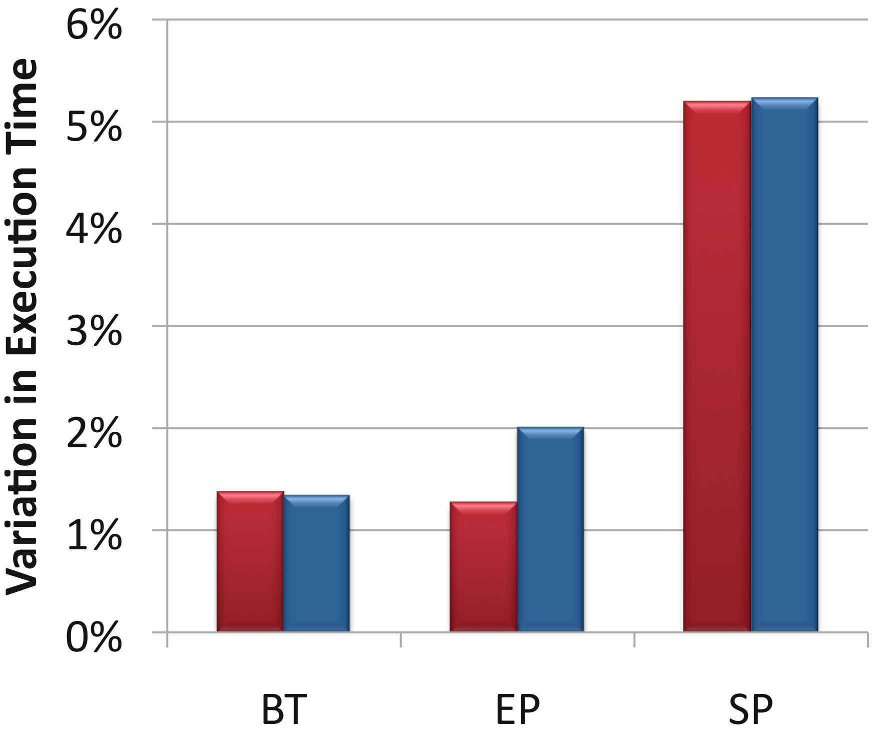
- Systems often have integrated process managers
  - Not all provide PMI functionality
- Efficient PMI implementation must make effective use of native process managers
  - Minimizing overhead
- Benchmark (1600 cores on SiCortex)
  - Class C BT, EP and SP
  - Using SLURM (which provides PMI-1)
  - Using Hydra over SLURM (for launch and management) plus PMI daemon

Runtime Impact of Separate PMI Daemons (1600 cores SiCortex)

Absolute Performance



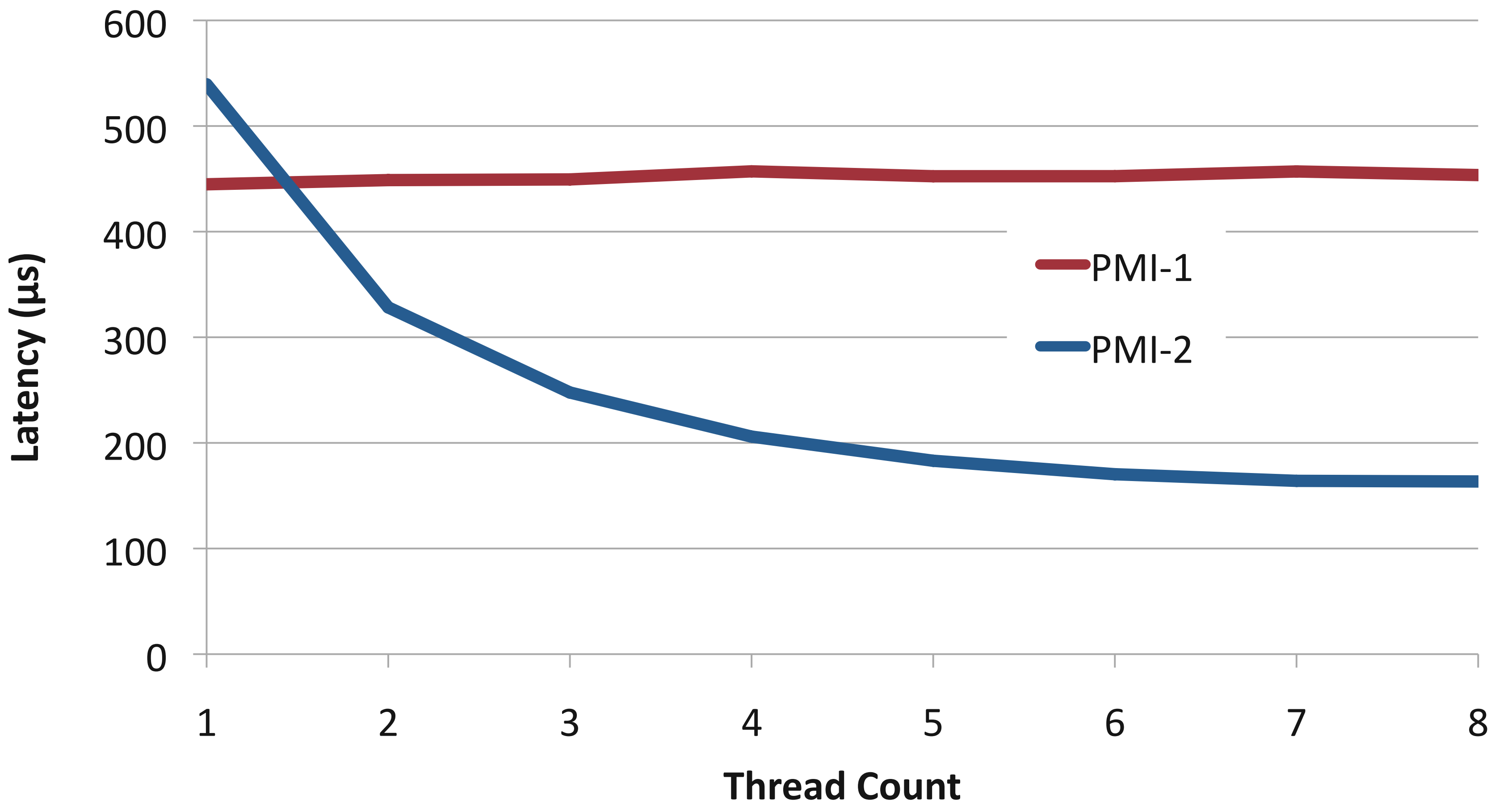
Percentage Variation



Multithreaded Performance

- PMI-1 is not thread safe
  - External locking is needed
- PMI-2 is thread safe
  - Multiple threads can communicate with PM
  - Hydra: lock only for internal communication
- Benchmark (8-core x86_64 node)
  - Multiple threads calling MPI_Publish_name(); MPI_Unpublish _name()
  - Work is fixed, number of threads increases

Multithreaded Performance



Conclusion

Conclusion

- We presented a generic process management interface PMI
- PMI-2: second generation eliminates PMI-1 shortcomings
  - Scalability issues for multicore systems
  - Issues for hybrid MPI-with-threads
  - Fault tolerance
- Performance evaluation
  - PMI-2 allows better implementations over PMI-1
  - Low overhead implementation over existing PM