Enhancing Startup Performance of Parallel Applications with SLURM

Sourav Chakraborty, Hari Subramoni, Adam Moody(1), Jonathan Perkins, and Dhabaleswar. K. Panda

Department of Computer Science & Engineering,
The Ohio State University
(1) Lawrence Livermore National Laboratory
Overview

• Introduction
• Challenges
• PMI Ring Extension
• Non-blocking PMI Extensions
• Conclusion
Current Trends in HPC

- Supercomputing systems scaling rapidly
  - Multi-core architectures and
  - High-performance interconnects
- InfiniBand is a popular HPC interconnect
  - 259 systems (51.8%) in top 500
- MPI and MPI+X programming models used by vast majority of HPC applications
- Job launchers for high performance middleware like MPI need to become more scalable to handle this growth!
Why is Fast Startup Important

**Developing and debugging**
- Developers spend a lot of time launching the application
- Reducing job launch time saves developer-hours

**Regression testing**
- Complex software have a lot of features to test
- Large number of short-running tests need to be launched

**System testing**
- Full-system size jobs to stress-test the network and software

**Checkpoint-restart**
- An application restart is similar to launching a new job
- Faster startup means less time recovering from a failure
Requirement for Out-of-band Startup Mechanisms in High-performance MPI Libraries

- InfiniBand is a low-latency, high-bandwidth network widely used in HPC clusters
- Lacks efficient hostname based lookup
- Requires some out-of-band communication before connection establishment
- Most MPI libraries use the Process Management Interface (PMI)\(^1\) as the out-of-band communication substrate

\(^1\) PMI: A Scalable Parallel Process-management Interface for Extreme-scale Systems; Balaji, Pavan and Buntinas, Darius and Goodell, David and Gropp, William and Krishna, Jayesh and Lusk, Ewing and Thakur, Rajeev; EuroMPI'10
Process Management Interface (PMI)

- Portable interface between middleware (e.g. MPI) and resource manager (e.g. SLURM, mpirun_rsh, Hydra)
- External process acts as the client, resource manager works as the server
- PMI provides these broad functionalities:
  - Creating/connecting with existing parallel jobs
  - Accessing information about the parallel job or the node on which a process is running
  - **Exchanging information used to connect processes together**
  - Exchanging information related to the MPI Name publishing interface
USE PMI-2!

Supported by most MPI libraries including MVAPICH2, OpenMPI
MVAPICH2

- High Performance open-source MPI Library for InfiniBand, 10Gig/iWARP, and RoCE
  - MVAPICH (MPI-1), Available since 2002
  - MVAPICH2 (MPI-2.2, MPI-3.0 and MPI-3.1), Available since 2004
  - MVAPICH2-X (Advanced MPI + PGAS), Available since 2012
  - Support for GPGPUs (MVAPICH2-GDR), Available since 2014
  - Support for MIC (MVAPICH2-MIC), Available since 2014
  - Support for Virtualization (MVAPICH2-Virt), Available since 2015
- Used by more than 2,450 organizations in 76 countries
- More than 285,000 downloads from the OSU site directly
- Empowering many TOP500 clusters (Jun‘15 ranking)
  - 8th ranked 519,640-core cluster (Stampede) at TACC
  - 11th ranked 185,344-core cluster (Pleiades) at NASA
  - 22nd ranked 76,032-core cluster (Tsubame 2.5) at Tokyo Institute of Technology and many others
- Available with software stacks of many IB, HSE, and server vendors including 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 (8th in Jun‘15, 462,462 cores, 5.168 PFlops)
Current PMI2 APIs

- PMI provides a **global key-value store** where each process can store or retrieve data from
  - PMI2_KVS_Put (key, value)
    - Store a new <key,value> pair
  - PMI2_KVS_Fence ()
    - Publish/synchronize the KVS across processes
    - Blocking operation, needs to be called by every process
  - PMI2_KVS_Get (... key, ...)
    - Lookup a <key,value> pair from the KVS
Use of PMI in High-performance MPI Libraries

• MPI libraries use the Put-Fence-Get operations to exchange their high-performance network endpoint addresses
• Each process Puts its own network endpoint address into the key-value store and calls Fence
• Each process does up to (Number of Processes – 1) Gets to look up the network endpoint address of remote processes
• Key-Value exchange over PMI takes more time as system size increases

• Other costs are relatively constant

• All numbers taken on TACC Stampede with 16 processes/node

• Based on MVAPICH2-2.0b & SLURM-2.6.5
Overview

- Introduction
- **Challenges**
  - PMI Ring Extension
  - Non-blocking PMI Extensions
  - Conclusion
Time Spent in Different PMI Operations

- One Put followed by a Fence and multiple Gets
- Put & Get are local operations and take negligible time
- Time taken by Fence is the bottleneck[2]

Time Spent in Different PMI Operations

- Time taken by Fence is determined by Data transferred
- Fence with no data movement is much faster
- Can we come up with other primitives to improve the performance?
Overview

• Introduction
• Challenges
• **PMI Ring Extension**
• Non-blocking PMI Extensions
• Conclusion
Using High Performance Networks for PMI

High Performance Cluster Middleware (MPI)

Communication Related to Job Startup

Exchange of End-point Information for High-performance Networks

- Complete Out-of-Band Exchange
  - Complete EP Information
- Partial Out-of-Band Exchange
  - Neighbor EP Information
  - Complete EP Information

Job Launcher

- Low Performance Communication Protocols (PMI2):
  - PUT
  - GET
  - FENCE
  - Barrier
  - Global Data Transfer

- High Performance Communication Protocols (IB Verbs):
  - Send
  - Recv
  - RDMA

Low Performance Networks

High Performance Networks
The PMI Ring Extension

```c
int PMIX_Ring ( const char value[], // IN – Own value int *rank, // OUT – Rank in ring int *size, // OUT – Size of ring char left[], // OUT – Value from rank-1 char right[], // OUT – Value from rank+1 int maxvalue // IN – Max length of values );
```

rank and size can be different from PMI size and rank
Already available in slurm-15.08.0 (thanks to Adam Moody)
Using PMI Ring Extension

Each process acquires its own InfiniBand address

PMIX_Ring – Exchange address with Left and Right neighbor processes

Form a Ring over InfiniBand using exchanged addresses

Perform Allgather operation over InfiniBand ring to gather addresses from all other processes
MVAPICH2 Startup with PMIX_Ring

- Amount of data transferred over TCP sockets reduced significantly
- Bulk of the data is exchanged over high-performance network (InfiniBand)
MPI_Init and Hello World with PMIX_Ring

- MPI_Init time reduced by 34%
- Time taken by Hello_World improved by 33% at 8,192 processes
Application Performance with PMIX_Ring

- NAS Parallel Benchmarks at 1,024 processes, class B data

- Up to 20% improvement in total execution time
Overview

• Introduction
• Challenges
• PMI Ring Extension
• **Non-blocking PMI Extensions**
• Conclusion
Non-blocking PMI Extensions

• Process manager (slurmd) is responsible for progressing the PMI exchanges. Can be overlapped with:
  • Different initialization related tasks, e.g.
    – Registering memory with the HCA
    – Setting up shared memory channels
    – Allocating resources
  • Any computation between MPI_Init and the first communication, e.g.
    – Reading input files
    – Preprocessing the input
    – Dividing the problem into sub-problems
Proposed Non-blocking PMI Extensions

```c
int PMIX_Allgather (const char value[],
        void *buffer);
```

- Each process provides an input value and an output buffer
- Values from each process are collected into the output buffer
- Values are ordered by their source rank

**PMIX_Request**

- Request objects are used to track completions of non-blocking operations
- Each non-blocking operation returns a handle to the request object
- Actual type of the object is determined by the implementation

```c
int PMIX_Wait (PMIX_Request request);
```

- Wait until the operation specified by the request object is complete
Proposed Non-blocking PMI Extensions

```c
int PMIX_Iallggather (  
    const char value[],  
    void *buffer,  
    PMIX_Request *request_ptr);
```

- Non-blocking version of the PMIX_Allgather
- Return does not indicate completion
- Output buffer will contain valid data only after successfully invoking the corresponding PMIX_Wait

```c
int PMIX_KVS_Ifence (PMIX_Request *request_ptr);
```

- Non-blocking version of the PMI2_KVS_Ifence

- All functions return 0 on success and an error code on failure
- PMI2_KVS_* can not be invoked between calling PMIX_KVS_Ifence and calling PMIX_Wait
Using Non-blocking PMI Extensions

Current

```c
MPI_Init() {
    PMI2_KVS_Put();
    PMI2_KVS_Fence();
    /* Do other tasks */
}

Connect() {
    PMI2_KVS_Get();
    /* Use values */
}
```

Proposed

```c
MPI_Init() {
    PMIX_Iallgather();
    /* Do other tasks */
}

Connect() {
    PMIX_Wait();
    /* Use values */
}
```
Design of PMIX_Allgather

- Put-Fence-Get combined into a single function
- Collective across all processes
- Optimized for symmetric data movement

```
int PMIX_Allgather (  
  const char value[], //UTF-8, NULL terminated  
  void *buffer       //size = NumProcs*MaxLength  
);
```

- Equivalent to Fence with rank used as the key
- Values are directly accessed from the result buffer
- Data from rank $r$ is available at $buffer[r\times MaxLength]$
- Further optimization by parameterizing $MaxLength$
Design of PMIX_Allgather

- Processes send the value to parent slurmd
- slurmd’s propagate the values (tagged with the source rank) to their parent
- srunc sends the aggregated data to children
- slurmd’s order the data by rank and sends to client processes
- More efficient packing/less data movement
- Avoids the expensive hash-table creation step

1. Send value to local slurmd
2. Forward values to parent slurmd
3. Forward values to parent slurmd
4. srunc sends gathered data to children
5. Forward gathered data to children
6. Order values by rank
7. Send ordered values to clients
Data Packing and Movement in Fence

Data packed for transfer between slurmd’s

Data from Process 1
Data from Process 2
Data from Process N

Data stored in Hash table in slurmd

Data sent to client process from slurmd

Data from Process 1
Data from Process 2
Data from Process N

286 KB @ 8K processes

Data Packing and Movement in Allgather

Data packed for transfer between slurmd’s

Data from Process 1
Data from Process 2
Data from Process N

Data sent to client process from slurmd

208 KB @ 8K processes

(27% less)
Performance of PMIX_Allgather

- Allgather performs 38% better than Fence at 16K processes
- Reduced data movement and processing overhead
- All numbers taken on TACC Stampede with 16 processes/node
- Based on MVAPICH2-2.0b & SLURM-2.6.5
Performance of MPI_Init with Non-blocking PMI

- Constant MPI_Init time using non-blocking PMI calls
- MPI_Init using Iallgather is 288% faster than using Fence at 16K processes
- Replacing the blocking Fence with blocking Allgather yields 21% benefit
Application Performance with Non-blocking PMI

- Sources of improvement
  - Overlap inside MPI_Init, depends on library and system size
  - Overlap outside MPI_Init, depends on application

- NAS Parallel Benchmarks
  - 4,096 processes
  - Class B data

- Improvements of up to 10% in total application run-time (as reported by the job launcher)

Overview

• Introduction
• Challenges
• PMI Ring Extension
• Non-blocking PMI Extensions
• Conclusion
Conclusion

- PMIX_Ring moves bulk of the PMI exchange over High-performance network like InfiniBand
- MPI_Init and Hello World is 33% faster @ 8K processes
- PMIX_Iallgather and PMIX_KVS_Ifence allows for overlap of PMI exchanges with library initialization and application computation
- MPI_Init can be completed in constant time at any scale using the proposed non-blocking PMI extensions (288% faster @ 16K)
- Total execution time of NAS benchmarks reduced by up to 20%
- Support for PMIX_KVS_Ifence is available since MVAPICH2-2.1
- SLURM support coming soon!
Thank you!

{chakrabs, subramon, perkinjo, panda}@cse.ohio-state.edu
moody20@llnl.gov

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