Job Startup at Exascale: Challenges and Solutions

Sourav Chakraborty
Advisor: Dhabaleswar K (DK) Panda

The Ohio State University
Current Trends in HPC

• Tremendous increase in system and job sizes

• Interconnects like InfiniBand and OmniPath is dominant

• Dense many-core systems like KNL are more common

• Hybrid MPI+PGAS models becoming popular

Fast and scalable job-startup is essential!
Why is Job Startup Important?

Development and debugging

Regression / Acceptance testing

Checkpoint - Restart
Towards Exascale: Challenges to Address

- Dynamic allocation of resources
- Leveraging high-performance interconnects
- Exploiting opportunities for overlap
- Minimizing memory usage
Challenge: Avoid All-to-all Connectivity

Connection setup phase takes 85% of initialization time with 4K processes.

The applications rarely require full all-to-all connectivity.

<table>
<thead>
<tr>
<th>Application</th>
<th>Processes</th>
<th>Average Peers</th>
</tr>
</thead>
<tbody>
<tr>
<td>BT</td>
<td>64</td>
<td>8.7</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td>10.6</td>
</tr>
<tr>
<td>EP</td>
<td>64</td>
<td>3.0</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td>5.0</td>
</tr>
<tr>
<td>MG</td>
<td>64</td>
<td>9.5</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td>11.9</td>
</tr>
<tr>
<td>SP</td>
<td>64</td>
<td>8.8</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td>10.7</td>
</tr>
<tr>
<td>2D Heat</td>
<td>64</td>
<td>5.3</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td>5.4</td>
</tr>
</tbody>
</table>
On-demand Connection Management

Main Thread
Put/Get (P2)
Create QP
QP→Init
Enqueue Send
Connection Manager Thread
Connection Established
Dequeue Send
Process 1

Main Thread
Connection Manager Thread
Connect Request
(LID, QPN)
(address, size, rkey)
QP→RTR
QP→RTS
Put/Get (P2)
QP→RTS
Connection Established

Process 2

Main Thread
Connection Manager Thread
Create QP
QP→Init
QP→RTR
Connect Reply
(LID, QPN)
(address, size, rkey)
Results - On-demand Connections

Performance of OpenSHMEM Initialization and Hello World

- Hello World - Static
- start_pes - Static
- Hello World - On-demand
- start_pes - On-demand

Execution time of OpenSHMEM NAS Parallel Benchmarks

- BT
- EP
- MG
- SP

Initialization – 29.6 times faster
Total execution time – 35% better
Challenge: Exploit High-performance Interconnects in PMI

- Used for network address exchange, heterogeneity detection, etc.
  - Used by major parallel programming frameworks

- Uses TCP/IP for transport
  - Not efficient for moving large amount of data
  - Required to bootstrap high-performance interconnects

PMI = Process Management Interface
PMIX_Ring: A Scalable Alternative

- Exchange data with only the left and right neighbors over TCP
- Exchange bulk of the data over High-speed interconnect (e.g. InfiniBand, OmniPath)

```c
int PMIX_Ring(
    char value[],
    char left[],
    char right[],
    ...)
```

Comparison of PMI operations

- Fence
- Put
- Gets
- Ring

**PMIX_Ring is more scalable**
Results - PMIX_Ring

Performance of MPI_Init and Hello World with PMIX_Ring

<table>
<thead>
<tr>
<th>Number of Processes</th>
<th>Time Taken (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td></td>
</tr>
<tr>
<td>128</td>
<td></td>
</tr>
<tr>
<td>256</td>
<td></td>
</tr>
<tr>
<td>512</td>
<td></td>
</tr>
<tr>
<td>1K</td>
<td></td>
</tr>
<tr>
<td>2K</td>
<td></td>
</tr>
<tr>
<td>4K</td>
<td></td>
</tr>
<tr>
<td>8K</td>
<td></td>
</tr>
</tbody>
</table>

- Hello World (Fence)
- Hello World (Ring)
- MPI_Init (Fence)
- MPI_Init (proposed)

33% improvement in MPI_Init

Total execution time – 20% better

NAS Benchmarks with 1K Processes, Class B Data

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Fence</th>
<th>Ring</th>
</tr>
</thead>
<tbody>
<tr>
<td>EP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SP</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Network Based Computing Laboratory

SC ’16
Challenge: Exploit Overlap in Application Initialization

- PMI operations are progressed by the process manager

- MPI/PGAS library is not involved

- Can be overlapped with other initialization tasks / application computation

- Put+Fence+Get combined into a single function - Allgater

```c
int PMIX_KVS_Ifence(
    PMIX_Request *request
)

int PMIX_Iallgather(
    const char value[],
    char buffer[],
    PMIX_Request *request
)

int PMIX_Wait(
    PMIX_Request request
)
```
Results - Non-blocking PMI Collectives

Performance of MPI_Init

- Fence
- Ifence
- Allgather
- Iallgather

Time Taken (Seconds) vs Number of Processes

Comparison of Fence and Allgather

- PMI2_KVS_Fence
- PMIX_Allgather

Near-constant MPI_Init at any scale

Allgather is 38% faster than Fence
Challenge: Minimize Memory Footprint

• Address table and similar information is stored in the PMI Key-value store (KVS)
• All processes in the node duplicate the KVS

• PPN redundant copies per node

PPN = Number of Processes per Node
Shared Memory based PMI

• Process manager creates and populates shared memory region

• MPI processes directly read from shared memory

• Dual shared memory region based hash-table design for performance and memory efficiency
Shared Memory based PMI

**Time Taken by one PMI_Get**

- Default
- Shmem

**PMI Memory Usage**

- Fence - Default
- Allgather - Default
- Fence - Shmem
- Allgather - Shmem

- **PMI Gets are 1000x faster**
- **Memory footprint reduced by O(PPN)**
Summary

- Near constant MPI/OpenSHMEM initialization at any process count
- 10x and 30x improvement in startup time of MPI and OpenSHMEM with 16,384 processes (1,024 nodes)
- $O(PPN)$ reduction in PMI memory footprint
Availability and Impact

• Tested at large-scale on Stampede and LLNL clusters

• All designs available as part of MVAPICH2 / MVAPICH2-X
  • MVAPICH powers Sunway TaihuLight - the #1 SuperComputer in the world!
  • 13th, 241,108-core (Pleiades) at NASA
  • 17th, 462,462-core (Stampede) at TACC

• Can be easily adopted by other MPI libraries and Resource Managers
  • Design of PMIX_Ring contributed to SLURM 15

• Other enhancements available as patches from MVAPICH2 website
  • Ongoing discussion to include them in future SLURM releases
Thank You!

http://go.osu.edu/mvapich-startup
http://mvapich.cse.osu.edu/

chakrabs@cse.osu.edu
panda@cse.osu.edu