Job Startup at ExaScale: Challenges and Solutions

Sourav Chakraborty, Dhabaleswar K. Panda (Advisor)
Dept. of Computer Science and Engineering,
The Ohio State University
Email: {chakrabs, panda}@cse.ohio-state.edu

***Abstract*—We identify the major performance bottlenecks and memory constraints in bootstrapping MPI and PGAS applications and propose scalable solutions for them. We introduce on-demand connection management to OpenSHMEM to reduce startup time. We also propose extensions to the PMI (Process Management Interface) standard to reduce the data transferred over the network as well as overlap the communication with other initialization tasks. Finally, we introduce a shared memory based design for PMI to reduce its memory footprint by a factor of processes per node (PPN). Our evaluation shows that with sufficient overlap, near-constant initialization time can be achieved at any process count for MPI and hybrid MPI+PGAS applications. Time taken for MPI_Init is reduced by 2.88 times at 16,384 processes. Initialization time of OpenSHMEM is improved by 29.6 times at 8,192 processes. Estimated memory footprint for PMI is reduced by nearly 1GB with 1 million processes and 16 PPN.**

I. Introduction

As high-performance computing clusters continue to increase in size, fast and scalable startup of parallel applications is becoming more important. As shown in Figure 1(a), the most time-consuming step while initializing OpenSHMEM is establishing $O(N^2)$ connections. However in most applications, a given process only communicates with a small subset of peers. Based on this, we introduce on-demand connection management for OpenSHMEM.

As shown in Figure 1(b), the major bottleneck in bootstrapping MPI libraries with support for on-demand connection management is the exchange of netwrok addresses over PMI. PMI provides a global key-value store and the following basic operations - *Put* (adds a key-value pair to the store), *Get* (retrieves the value for a given key), and *Fence* (distributes the key-value pairs to all processes). We identify and propose solutions for several inefficiencies in the current PMI protocol.

II. Proposed Designs

A. On-demand Connection Management

In the proposed design, connection between two processes are established only when they try to communicate for the first time. Each process maintains a thread communicating over the connectionless but unreliable UD protocol. Information about memory segments are serialized and sent to the remote peer with the first message. This information is used to establish reliable RC connections [1].

B. PMI Ring Extension

Currently, inter-node PMI communication goes over TCP/IP and do not take advantage of high-speed networks like InfiniBand. We propose a PMI operation, PMIX_Ring, to establish a ring structure by exchanging key-value pairs with the left and right neighbors. Once the ring is established, the the high speed network is used to exchange bulk of the data. Consequently, the total time is reduced significantly as time taken by the Ring operation is nearly independent of process count [2].

C. Non-blocking PMI Collectives

While PMI operations are being progressed by the process manager, the MPI processes cannot perform anything useful due to their blocking nature. We introduce a non-blocking collective called *Ifence* which allows processes to perform useful tasks while the PMI operations progress in the background. We also introduce *Allgather*, and its non-blocking variant, *Iallgather*, which combine the functionality of *Put*, *Fence* and *Get* and reduce data movement by using the rank as the implicit key[3].

D. Shared Memory Based PMI Get/Allgather

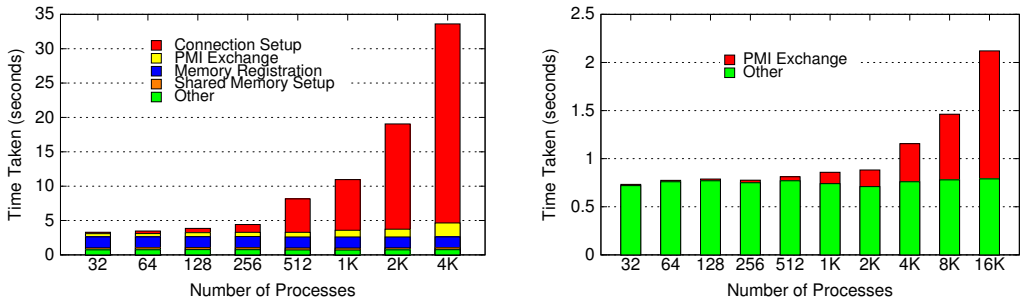
In the current design, intra-node PMI communication is performed over UNIX sockets which causes poor latency as number of concurrent clients increases. Furthermore, the key-value store is replicated PPN+1 times on each node. We propose a design where the key-value store is exposed using shared memory regions. This improves Get performance and reduces PMI memory footprint per node by PPN times [4].

III. Results

The TACC Stampede system was used for experimental evaluation. Each compute node is equipped with two Intel SandyBridge eight-core sockets @2.70 GHz, 32 GB RAM and FDR ConnectX-3 HCAs (56 Gbps). SLURM-2.6.5 and MVAPICH2-2.1 were used to implement the proposed designs.

A. On-demand Connection Management

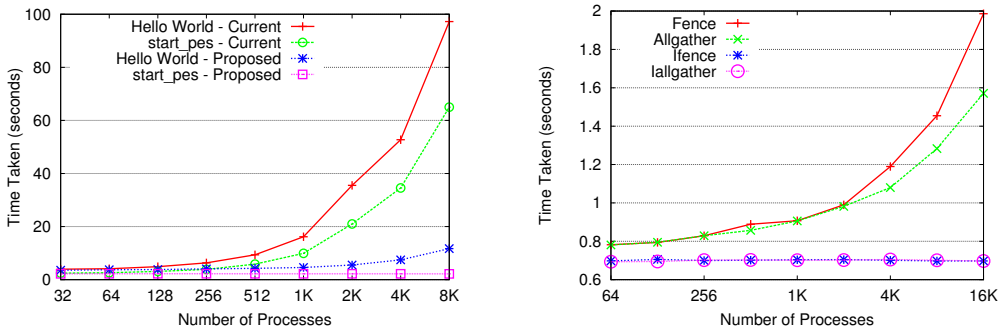
Figure 2(a) shows the improvements from using on-demand connection management. At 8,192 processes, OpenSHMEM initialization is 30 times faster. Total execution time of NAS Parallel Benchmarks [5] is reduced by up to 35% with 256 processes.



(a) OpenSHMEM Initialization (start_pes)

(b) MPI Initialization (MPI_Init)

Fig. 1. Breakdown of time taken for initializing OpenSHMEM and MPI



(a) OpenSHMEM initialization and Hello World

(b) MPI_Init

Fig. 2. OpenSHMEM and MPI initialization times with proposed designs

B. PMI Ring Extension

The Ring extension improves performance of MPI_Init by up to 34% and reduces the time taken by Hello_World by up to 33%. Total execution time for NAS Parallel Benchmarks with 1,024 processes is improved by up to 20%.

C. Non-blocking PMI Collectives

Replacing the blocking *Fence* operation with blocking *Allgather* yields 20% benefit at 16K processes. Using the non-blocking PMI extensions results in a constant MPI_Init time independent of the number of processes. At 16,384 processes, MPI_Init with *Iallgather* is 2.88 times faster, as shown in Figure 2(b).

D. Shared Memory based PMI

With the shared-memory design, PMI Gets take 0.25ms compared to the default 250ms with 32 clients (1,000x improvement). With 1 million processes and 16 ppn, nearly 1GB per node can be saved.

IV. Future Work

The proposed designs have been adopted into latest versions of MVAPICH2 and SLURM. We plan to extend support for on-demand connection management to UPC, CAF and other PGAS languages. We also plan to identify and propose solutions to minimize other bottlenecks in job-startup.

References

[1] Sourav Chakraborty, Hari Subramoni, Jonathan Perkins, Ammar A Awan, and Dhabaleswar K Panda. On-demand Connection Management for OpenSHMEM and OpenSHMEM+MPI. In *Parallel & Distributed Processing Symposium Workshops (IPDPSW), 2015 IEEE International*. IEEE, 2015.

[2] S. Chakraborty, H. Subramoni, J. Perkins, A. Moody, M. Arnold, and D. K. Panda. PMI Extensions for Scalable MPI Startup. In *Proceedings of the 21st European MPI Users' Group Meeting*, EuroMPI/ASIA '14, pages 21:21–21:26, New York, NY, USA, 2014. ACM.

[3] Sourav Chakraborty, Hari Subramoni, Adam Moody, Akshay Venkatesh, Jonathan Perkins, and Dhabaleswar K Panda. Non-Blocking PMI Extensions for Fast MPI Startup. In *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on*, pages 131–140. IEEE, 2015.

[4] Sourav Chakraborty, Hari Subramoni, Jonathan Perkins, and Dhabaleswar K Panda. SHMEMPMI: Shared Memory Based PMI for Improved Performance and Scalability. In *Cluster, Cloud and Grid Computing (CCGrid), 2016 16th IEEE/ACM International Symposium on*. IEEE, 2016.

[5] D H Bailey, E Barszcz, J T Barton, D S Browning, R L Carter, L Dagum, R A Fatoohi, P O Frederickson, T A Lasinski, R S Schreiber, et al. The NAS Parallel Benchmarks. *IJHPCA*, 5(3):63–73, 1991.