Collective Framework and Performance Optimizations to Open MPI for Cray XT Platforms

Joshua S. Ladd, Manjunath Gorentla Venkata, Pavel Shamis, Richard L. Graham
Oak Ridge National Laboratory (ORNL)
Oak Ridge, TN
U.S.A.
{laddjs, manjugv, shamisp, rlgraham}@ornl.gov

The performance and scalability of collective operations plays a key role in the performance and scalability of many scientific applications. Within the Open MPI code base we have developed a general purpose hierarchical collective operations framework called Cheetah, and applied it at large scale on the Oak Ridge Leadership Computing Facility's Jaguar (OLCF) platform, obtaining better performance and scalability than the native MPI implementation. This paper discuss Cheetah's design and implementation, and optimizations to the framework for Cray XT 5 platforms. Our results show that the Cheetah's Broadcast and Barrier perform better than the native MPI implementation. For medium data, the Cheetah's Broadcast outperforms the native MPI implementation by 93% for 49,152 processes problem size. For small and large data, it out performs the native MPI implementation by 10% and 9%, respectively, at 24,576 processes problem size. The Cheetah's Barrier performs 10% better than the native MPI implementation for 12,288 processes problem size.

1 Introduction

Previous research has shown that collective operations are important for the performance and scalability of HPC applications. The profiling of the HPC applications show that a significant amount of execution time is spent in the collective operations. The projections of exa-scale systems predict that the systems will contain million or more processor cores. As the HPC systems increase in size the amount of application time spent in collectives only increases.

To improve the performance of MPI collectives for recent and upcoming HPC systems, MPI implementations should support asynchronous progress, be optimized for the deep cache hierarchies, support arbitrary network communication hierarchies, and take advantage of collective capabilities of modern network interfaces [4], [7], [5]. However, current Open MPI implementation does not take advantage of modern hardware architectures to improve the performance of MPI routines [2]. The Open MPI's current infrastructure does not provide the flexibility to implement customized routines for each of these communication architectures. Its algorithms and implementation does not take into account the latency of communication hierarchies. Further, it does not support the non-blocking collective operations that help improve overlap characteristics of HPC applications.

In this paper, we present the Cheetah framework, a general purpose collectives framework, implemented with the Open MPI, and the optimizations for the Cray XT platforms. The various communication network hierarchies are expressed as subgroups or hierarchies, and a collective primitive is implemented for each of these hierarchy. The coordinated execution of these primitives achieve a global collective operation. The grouping of processes and algorithm implementation are decoupled, as a result the framework can choose any algorithm for each of these communication hierarchies providing customized implementation for each of these hierarchies, also increasing the code reuse. Further, the framework supports asynchronous progress of collectives, and provides scalable memory usage mechanisms.

In the rest of paper, we present the Cheetah design in Section 3, and discuss the optimizations for Cray XT platforms. Then present results in Section 5 and conclude in Section 6.

2 Related Work

To improve the performance of collectives for modern hardware architecture, most of the research has followed the approach of optimizing algorithms for the shared memory architectures [10], [9], [3]. Mamidala et. al optimize the collectives by considering data layouts for the L2 and L3 caches, and minimizing communication over the network

between MPI processes [7]. Tipparaju et al. replace point to point communications either by shared memory or remote memory protocols in collective operations for SMP clusters [10].

There is another body of research that takes the approach of improving the performance of collectives by optimizing them for various homogeneous architectures. Almasi et al. have developed collectives for the BG/L platform taking advantage of the hardware support for the collectives [1]. Ritzdorf et al. developed collectives for NEC's SX platforms [8]. This research [6] took advantage of the topology of switches and chassis of InfiniBand cluster for *Scatter* and *Gather* collectives. This research [2] however concentrated on selecting collectives for a architecture from the available implementations.

The Cheetah framework provides a system for implementing customized collectives over arbitrary communication network topologies. Though we present an optimized framework for Cray XT platforms here, the platform is general to implement collectives over Cray XT platforms, InfiniBand, CORE-*Direct* and Ethernet systems. It supports non-blocking collectives and helps achieve more than 90% communication and computation overlap on CORE-*Direct* systems [11].

3 Design

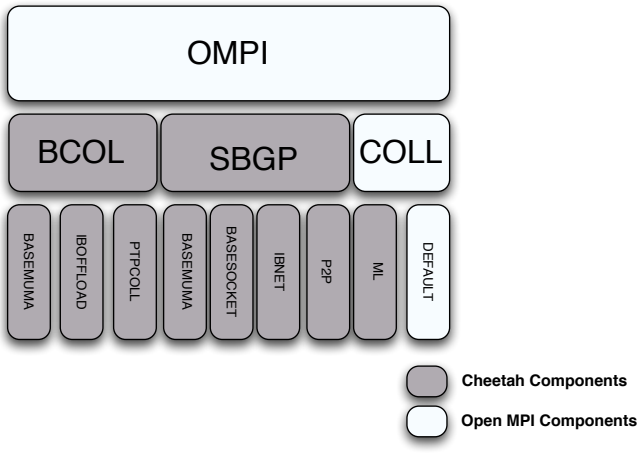

Figure 1. The components of the Cheetah framework along with Open MPI components.

Figure 1 shows the Cheetah components along with the Open MPI components. We provide a brief overview of the design, architecture and implementation in this section, and more detailed discussion can be found in Graham et al. [4]

The collective routines in the Cheetah framework are implemented as a combination of collective routines over the communication hierarchies. We call this collective routines - collective primitives. The framework provides the components to build, implement, and execute the hierarchical collective primitives and collective routines. The main components of the Cheetah framework are BCOL, SBGP, and ML components.

The BCOL component is responsible for implementing the collective primitives that are optimized for different communication architectures. For Cray XT platforms, the collective primitives are implemented in PTP BCOL component. It uses Open MPI portals BTL for message transfer. Since PTP BCOL component uses PML for message transfer, which supports network interfaces such as InfiniBand, Ethernet besides Cray XT, the PTP BCOL collective primitives can be used over these networks. Besides Cray XT, InfiniBand, and Ethernet, there are specialized BCOL components for shared memory and CORE-*Direct*.

The SBGP component groups the processes into subgroups based on the communication hierarchy. Currently, the framework supports UMA, SOCKET, IBNET and P2P subgroups. For example, the processes on the same CPU socket are grouped together into SOCKET subgroup, and processes on a same node can be grouped together into a UMA subgroup.

The ML component coordinates the BCOL's collective primitives across all the subgroups, provides data and control buffers for the collective operations, and provides a mapping from MPI collective semantics to collective primitives. The small data transfers on the Cray XT platforms use mmaped memory bank for shared memory communication, and for the large data transfers it depends on portals library, particularly it uses *PtlMDMDCopy* copy mechanism.

The collective primitives that are combined to achieve a global collective operation and the order of its execution is defined by a schedule. The schedule takes into account the source of the data for the collective, if any. The progress engine based on the schedule starts the collective primitives. Using a wait list and active list, the progress engine manages the progress of the routines until the completion of the collective.

4 Collective Algorithms

We have implemented Barrier and Broadcast Collectives optimized for Cray XT platforms in the Cheetah framework.

4.1 Open MPI Portals BTL

Open MPI portals BTL provides a message transfer layer for Cray XT platforms. It uses an eager protocol for small message data and rendezvous protocol for large message

data. The eager protocol sends the message from the sender to the receiver without any flow control. The BTL layer uses the portals acknowledgement mechanism to learn the successful delivery of the data. For large message data, the sender sends a RTS (Request To Send) message to the receiver, and the receiver sends the CTS (Clear To Send) message to signal the readiness of the receiver to receive the data.

In the above protocols, every portal message is acknowledged by the receiver and the sender uses the acknowledgement as a successful transmission of the message. However, the Cray XT 5 platforms' SeaStar network interface uses BEER (Basic End to End Reliability) protocol for message transfer. The protocol provides an end-to-end reliability between two endpoints, it handles failures such as packet loss, receive not read, and resource shortages.

In the Cheetah framework, message transfers on the Cray XT platforms take advantage of the BEER protocol. The portal message transfer does not generate an acknowledgment in the case of small message protocol, or for RTS and CTS messages in the case of large message data protocol.

4.2 Broadcast

The hierarchical Broadcast is implemented as a Broadcast operation split over multiple subgroups or hierarchies. Each of these split routines are implemented using different algorithms that are customized for a network communication hierarchy. A compile-time or run-time schedule of split operations provides the order at which these operations are executed. A progress engine which is part of the ML component is responsible for executing these operations. The order of execution of these operations can be arbitrary, sequential order, or order defined based on the source of the data.

For Cray XT platforms, we implement the algorithms as three split Broadcast operations over SOCKET, UMA, and P2P subgroups. For example, a three level hierarchy Broadcast that has split collective primitives over SOCKET, UMA, and P2P subgroups. The processes on a CPU socket are grouped into a SOCKET subgroup, and a process among them is selected as a group leader. The group leader is responsible for any communication between processes of any two subgroups. The leaders of the SOCKET subgroup are combined to form a UMA subgroup, and the leaders of the UMA subgroup form a P2P subgroup. The SOCKET and UMA subgroups use shared memory BCOL, and the Broadcast is implemented as a recursive doubling algorithm, and P2P subgroups use PTPCOLL BCOL, and the Broadcast is implemented as a scatter-allgather algorithm.

The ML component coordinates the data and control between these split broadcasts. It also provides and manages the control and data buffers required for the collective operations.

We have various implementations of Broadcast collectives in the Cheetah framework. They either vary in the amount of concurrent progress allowed for the collective primitives, or the tree used for data distribution. The *k-nomial Knownroot* algorithm uses *k-nomial* tree for data distribution, and all independent collective primitives over different hierarchies can progress concurrently. The sequential algorithm is similar to *Knownroot* except that the collective primitives are progressed sequentially, and on a node no two primitives progresses at a given time. The N-ary algorithm is similar to *Knownroot* except that it uses N-ary tree for data distribution.

4.3 Barrier

The hierarchical Barrier like hierarchical Broadcast is implemented as split collective primitives. For Cray XT platforms, the hierarchical Barrier is implemented as collective primitives over SOCKET, UMA, and P2P subgroups. The shared memory BCOL barrier routine is implemented as a fanin algorithm, and the PTPCOLL BCOL barrier routine is implemented as a recursive doubling barrier primitive. Say if we have N subgroups (hierarchies) in the collective routine execution, the first $N-1$ subgroup levels use FANIN and FANOUT routines and the last subgroup use recursive doubling routine. The collective routine over P2P BCOL uses portals BTL for message transfer. The shared memory collective routine uses a mmaped backed memory buffer which is a lockless memory sharing mechanism in Cheetah for small message transfer. For large message transfer, the Broadcast routine uses a portal library call.

5 Evaluation

In this section, we first provide an overview of experimental test-bed and then provide the results showing the performance characteristics of Open MPI point-to-point operations, and Cheetah's Barrier and Broadcast collective operations.

5.1 Experimental Setup

We ran all experiments on the OLCF's Jaguar, a Cray XT5 system. It has 18,688 compute nodes, each one containing two 2.6 GHz AMD Opteron (Istanbul) processors, 16 GB of memory, and a SeaStar 2+ router. The routers are connected in a 3D torus topology. Each AMD Opteron processor has six computing cores and three levels of cache memory. The compute nodes run Compute Node Linux micro-kernel.

5.2 Results

5.2.1 Point-to-point Message Latency and Bandwidth

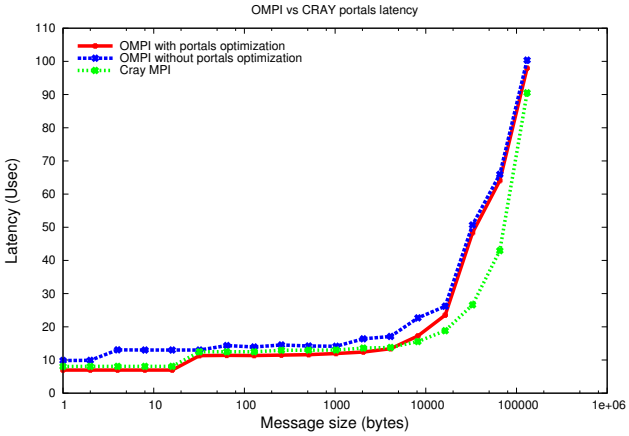

Figure 2. The point-to-point message latency with and without optimizations in Open MPI compared to Cray MPI.

Figures 2 shows the latency of message with and without the portals optimization compared to the latency of message that is using Cray MPI. The one byte latency of Open MPI is 6.97 $usecs$ which is 15% better than the Cray MPI. The latency of Open MPI point-to-point messages are better for small messages (up to 2 KB). However, for larger message size the Cray MPI message is marginally better.

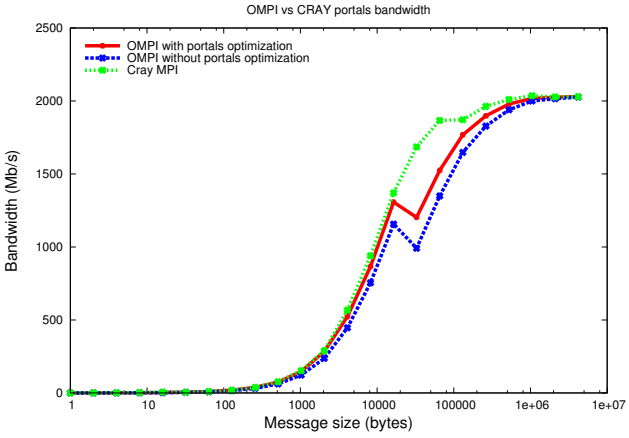

Figure 3. The message bandwidth with and without optimizations in Open MPI compared to Cray MPI.

Figure 3 shows the bandwidth of Open MPI with and without the portals optimization compared to the bandwidth of Cray MPI. Though the bandwidth for protocols in Open MPI improved when the optimizations were applied, it is only similar to Cray MPI protocols. They both achieve a peak bandwidth of 2 Gbp/s for 4 MB message size.

5.2.2 Latencies of Broadcast Implementations

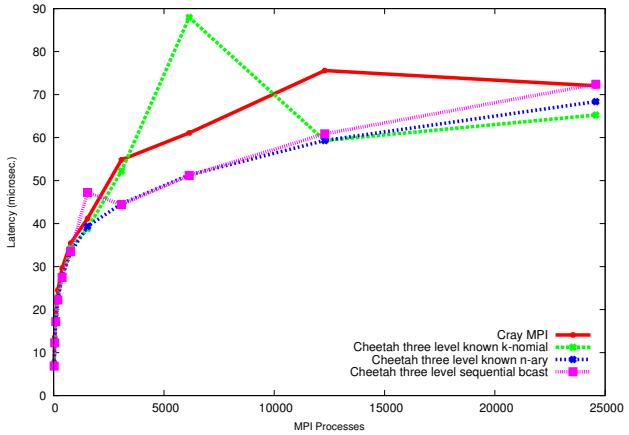

Figure 4. The latency of 8 byte Cheetah Broadcast with three levels of hierarchy compared to the Cray MPI.

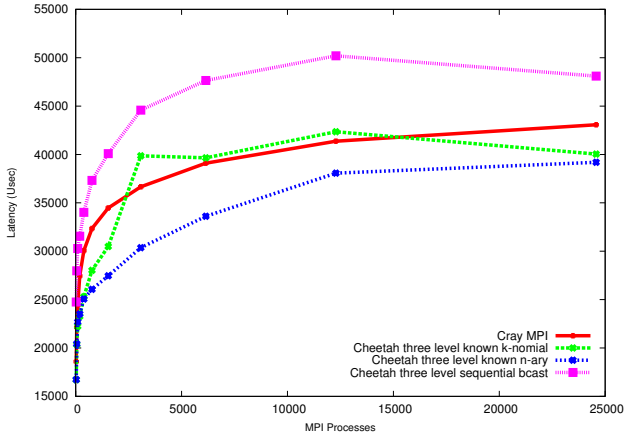

Figure 5. The latency of 4 MB Cheetah Broadcast with three levels of hierarchy compared to the Cray MPI.

Figures 4, 6, and 5 show results comparing the latencies of Cheetah's Broadcast with Cray MPI for small, medium and large data.

For small data as seen in Figure 4 for 24,576 processes problem size, the latencies of Cheetah's *Knownroot* and N-ary algorithms are 65.26, and 68.38 $usecs$, respectively,

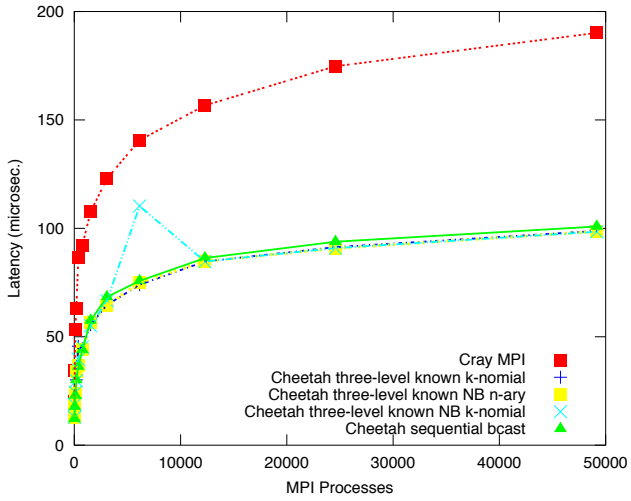

Figure 6. The latency of 4 KB Cheetah Broadcast with three levels of hierarchy compared to the Cray MPI.

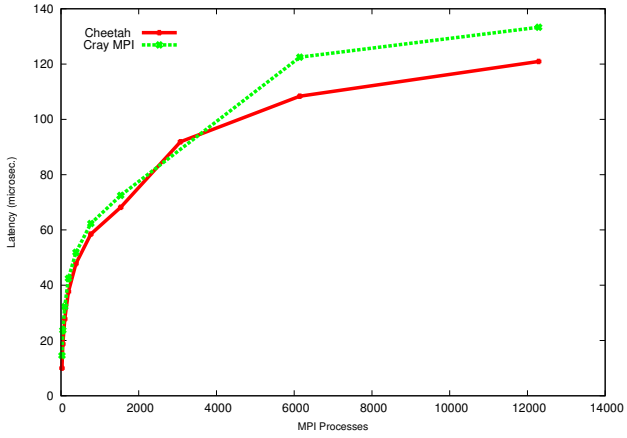

Figure 7. The latency of Barrier with three levels of hierarchy compared to the Cray MPI.

which is 10% and 5% better than the Cray MPI. The performance of sequential Broadcast is however similar to the Cray MPI. For large data as seen in Figure 5 for 24,576 processes problem size, the Cheetah's *Knownroot* and N-ary algorithms outperform the Cray MPI by 7% and 9%, and the sequential Broadcast algorithm performs 11% worse.

Figure 6 show the performance of 4 KB Cheetah broadcast compared to the Cray MPI. The Cheetah's algorithms, blocking *Knownroot*, nonblocking *Knownroot*, N-ary, and sequential algorithm all perform better than the Cray MPI. The latency of blocking *Knownroot*, N-ary, nonblocking *Knownroot*, sequential and Cray MPI are 98.79, 98.58, 98.391, 100.82, and 190.154 $usecs$, respectively, which are 92%, 92%, 93% and 88% better than the Cray MPI. From the graph we can observe that Cheetah's algorithms out perform Cray MPI for all problem sizes.

5.2.3 Latencies of Barrier Implementations

Figure 7 shows the performance of Cheetah barrier compared to the Cray MPI. For 12,288 processes problem size, the Cheetah's barrier is 120.94 $usecs$ which is 10% better than the Cray MPI. The Cheetah's Barrier performance though, lags behind the Cray MPI for smaller problem sizes, it out performs the Cray MPI after 3,072 processes problem size.

6 Conclusion

The performance of Barrier and Broadcast demonstrate the potential of Cheetah framework to take the advantage of hardware capabilities provided by the Cray XT platforms. Further, the optimizations to the Open MPI portal BTL decreased the latency of point-to-point message protocols and collectives.

In Figure 7, we observe that the performance of Cheetah's barrier is better than the Cray MPI for problem size greater than 3,072 processes, and it out performs the Cray MPI by 10% for 12,288 processes problem size. For smaller problem size, the overhead of hierarchy synchronization is adding to the latency of Barrier, degrading the performance. However at the large problem size, this overhead is negated by the performance improvement achieved because of concurrent progress of collective primitives and reduced point-to-point message transfer latency. In our previous paper [4], we have shown that by configuring Barrier with two level hierarchy the overhead of hierarchy synchronization can be reduced while improving the performance.

Figure 4, 6, and 5 show that the Cheetah's *Knownroot* and N-ary Broadcast algorithms outperforms the Cray MPI. For small and large data and at 24,576 processes problem size, the Cheetah's *Knownroot* algorithm outperforms the Cray MPI by 10% and 7%, respectively, and the N-ary algorithm outperforms the Cray MPI by 5% and 9%, respectively. For medium data and at 49,152 proccesses, the *Knownroot* algorithm outperforms the Cray MPI by 92%. We can also observe that all Cheetah's algorithms outperform the Cray MPI by 90% on average for a 4 KB message size. However for large data, the sequential algorithm performs worse than the Cray MPI by 11%. This, all algorithms outperforming the Cray MPI and sequential

algorithm not performing well, demonstrates the potential of Cheetah's design – concurrent progressing of collective primitives, customizing the collective for the communication hierarchy and arbitrary communication topology, and reduced point-to-point message latency – to improve the performance characteristics of collective implementations.

References

[1] G. Almási, P. Heidelberger, C. J. Archer, X. Martorell, C. C. Erway, J. E. Moreira, B. Steinmacher-Burow, and Y. Zheng. Optimization of mpi collective communication on bluegene/l systems. In *Proceedings of the 19th annual international conference on Supercomputing*, ICS '05, pages 253–262, New York, NY, USA, 2005. ACM.

[2] E. Garbriel, G. Fagg, G. Bosilica, T. Angskun, J. J. D. J. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. Castain, D. Daniel, R. Graham, and T. Woodall. Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, 2004.

[3] R. L. Graham and G. Shipman. MPI support for multi-core architectures: Optimized shared memory collectives. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 130 – 140, 2008.

[4] R. L. Graham, M. G. Venkata, J. S. Ladd, P. Shamis, I. Rabinovitz, V. Filipov, and G. Shainer. Cheetah: A framework for scalable hierarchical collective operations. In *In Proceedings of the 11th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, USA Newport Beach, CA, USA, May 2011 (To appear).

[5] T. Hoefler and G. Zerah. Optimization of a parallel 3d-FFT with non-blocking collective operations. Invited presentation at the 3rd International ABINIT Developer Workshop, Liege, Belgium, 01 2007.

[6] K. Kandalla, H. Subramoni, A. Vishnu, and D. Panda. Designing topology-aware collective communication algorithms for large scale infiniband clusters: Case studies with scatter and gather. In *Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, pages 1 –8, april 2010.

[7] A. Mamidala and et al. MPI collectives on modern multi-core clusters: Performance optimizations and communication characteristics. CCGRID, 2008.

[8] H. Ritzdorf and J. L. Träff. Collective operations in nec's high-performance mpi libraries. In *Proceedings of the 20th international conference on Parallel and distributed processing*, IPDPS'06, pages 100–100, Washington, DC, USA, 2006.

[9] S. Sistare, R. vandeVaart, and E. Loh. Optimization of MPI collectives on clusters of large-scale SMP's. In *Supercomputing, SC '99: Proceedings of the 1999 ACM/IEEE conference on Supercomputing (cd-rom)*, page 23, New York, New York, USA, 1999. ACM.

[10] V. Tipparaju, J. Nieplocha, and D. Panda. Fast collective operations using shared and remote memory access protocols on clusters. International Parallel and Distributed Processing Symposium (IPDPS), April 2003.

[11] M. G. Venkata, R. L. Graham, J. S. Ladd, P. Shamis, I. Rabinovitz, V. Filipov, and G. Shainer. Connectx-2 core-direct enabled asynchronous broadcast collective communications. In *In Proceedings of the 25th IEEE International Parallel & Distributed Processing Symposium (IPDPS), CASS Workshop*, Anchorage, Alaska, USA, May 2011 (To appear).