Process Arrival Pattern and Shared Memory Aware Alltoall on InfiniBand

Ying Qian and Ahmad Afsahi

Department of Electrical and Computer Engineering
Queen's University
Kingston, ON, CANADA K7L 3N6
ying.qian@ece.queensu.ca ahmad.afsahi@queensu.ca

Abstract. Recent studies have shown that processes in real applications can arrive at the collective calls at different times. This imbalanced process arrival pattern can significantly affect the performance of the collective operations. MPI_Alltoall() is a communication-intensive collective operation that is used in many parallel scientific applications. Its efficient implementation under different process arrival patterns is critical to the performance of applications that use them frequently. In this paper, we propose novel RDMA-based process arrival pattern aware MPI_Alltoall() algorithms over InfiniBand clusters. We extend the algorithms to be shared memory aware for small to medium size messages. The micro-benchmark and application results indicate that the proposed algorithms outperform the native implementation as well as their non-process arrival pattern aware counterparts when processes arrive at different times.

Key words: Process Arrival Pattern, Collective Communications, MPI_Alltoall(), RDMA, InfiniBand

1 Introduction

Most scientific applications are developed on top of the *Message Passing Interface* (MPI) [1]. Such applications extensively use MPI collective communication operations. Most research on developing and implementing efficient collective communication algorithms assume all MPI processes involved in the operation arrive at the same time at the collective call. However, it has been recently shown that *process arrival patterns* (PAP) for collectives are adequately imbalanced that will adversely affect the performance of collective communications [2]. In addition, it has been found that different collective communication algorithms react differently to PAP [2]. In this regard, the authors in [3] have recently proposed PAP aware *MPI_Bcast()* algorithms and implemented them using MPI point-to-point primitives.

InfiniBand [4] has been introduced to support the ever-increasing demand for efficient communication, scalability, and higher performance in clusters. It supports *Remote Direct Memory Access* (RDMA) operations that allow a process

to directly access the exposed memory areas of a remote process. RDMA is a one-sided read, write, or atomic operation, offloaded to the network interface card. MPI implementations over RDMA-enabled networks such as InfiniBand are able to effectively bypass the operating system overhead and lower the CPU utilization.

In this paper, we take on the challenge to design and efficiently implement PAP aware *MPI_Alltoall*() collective on top of InfiniBand. In MPI_Alltoall(), each process has a distinct message for every other process. It is a communication-intensive operation that is typically used in linear algebra operations, matrix multiplication, matrix transpose, and multi-dimensional FFT. Therefore, it is very important to optimize its performance on emerging multi-core clusters in the presence of different process arrival patterns. Our research is along the work in [3], however it has a number of significant differences. First, the authors in [3] have incorporated control messages in their algorithms at the MPI layer to make the processes aware of and adapt to the PAP. These control messages incur high overhead, especially for short messages. Our proposed PAP aware MPI_Alltoall() algorithms instead is RDMA-based, and we use its inherent mechanism for notification purposes. Therefore, there are no control messages involved and thus there is no overhead. Secondly, while [3] is targeted at large messages, we propose and evaluate two RDMA-based schemes for small and large message. Thirdly, we propose an intra-node PAP and shared memory aware scatter operation to boost the performance for small messages.

Our performance results indicate that the proposed PAP aware MPI_Alltoall() algorithms perform better than the native MVAPICH [5] and the non-PAP aware algorithms when processes arrive at different times. Our RDMA-based PAP and shared memory aware algorithm is the best algorithm up to 256B messages and gains up to 3.5 times improvement over MVAPICH. The proposed RDMA-based PAP aware algorithm is the algorithm of choice for 512B to 1MB messages; it outperforms the native MVAPICH by a factor of 3.1 for 8KB messages.

The rest of the paper is organized as follows. In Section 2, we provide an overview of InfiniBand and ConnectX adapter from Mellanox Technologies [6], as well as the implementation of MPI_Alltoall() in MVAPICH. Section 3 discusses the motivation behind this work by presenting the performance of MPI_Alltoall() in MVAPICH when processes arrive at the collective at random times. Section 4 presents our proposed PAP and shared memory aware MPI_Alltoall() algorithms. In Sect. 5, the performance of the proposed algorithms on a four-node multi-core cluster is presented. Related work is discussed in Sect. 6. Section 7 concludes the paper.

2 Background

InfiniBand [4] is an I/O interconnection technology consisting of end nodes and switches managed by a central subnet-manager. End nodes use Host Channel Adapters (HCA) to connect to the network. InfiniBand verbs form the lowest level of software to access the InfiniBand protocol-processing engine offloaded to

the HCA. The verbs layer has queue-pair based semantics, in which processes post send or receive work requests to send or receive queues, respectively. InfiniBand supports both the channel semantics (send-receive) and the memory semantics (using one-sided RDMA operations). RDMA-based communication requires the source and destination buffers to be registered to avoid swapping memory buffers before the DMA engine can access them.

ConnectX [6] is the most recent generation of InfiniBand HCAs by Mellanox Technologies. Its architecture includes a stateless offload engine for protocol processing that improves the performance by having hardware schedule the packet processing directly. This technique allows the ConnectX to have a better performance for processing simultaneous network transactions, as used in our algorithms.

In MVAPICH [5], point-to-point and some MPI collective communications have been implemented directly using RDMA operations. However, MPI_Alltoall() uses the two-sided MPI send and receive primitives, which transparently uses RDMA. Different algorithms are employed in MPI_Alltoall() for different message sizes: the *Bruck* [7] algorithm for small messages, the *Recursive-Doubling* [8] algorithm for large messages and power of two number of processes, and the *Direct* algorithm for large messages and non-power of two number of processes.

3 Motivation

To show how the native MVAPICH MPI_Alltoall() perform on our platform under random PAP, we use a micro-benchmark similar to [3]. Processes first synchronize using an *MPI_Barrier()*. Then, they execute a random computation before entering the MPI_Alltoall(). The random computation time is bounded by a maximum value MIF (*maximum imbalanced factor* [3]) times the time it takes to send a message.

The experiment was conducted on a 4-node, 16-core ConnectX cluster, described in Sect. 5. To get the performance of MPI_Alltoall(), a high-resolution timer is inserted before and after the MPI_Alltoall() operation. The completion time is reported as the average execution time across all the processes. Figure 1 presents the performance of MVAPICH MPI_Alltoall() when MIF is 1, 32, 128 and 512, respectively. Clearly, the completion time is greatly affected by the increasing amount of random computation. The results confirm the findings in [2] that PAP can have significant impact on the performance of collectives. It is therefore crucial to design and implement PAP aware collectives to improve their performance and consequently the performance of the applications that use them frequently.

4 The Proposed Process Arrival Pattern Aware MPI_Alltoall()

In this section, we propose our PAP and shared memory aware algorithms for MPI_Alltoall(). The basic idea in our algorithms is to let the early-arrival pro-

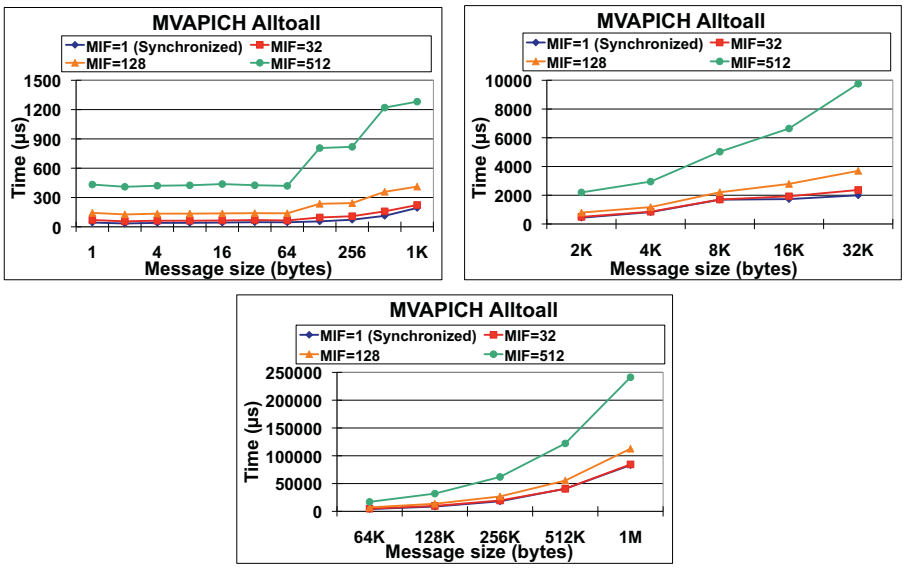

Fig. 1. Performance of MVAPICH MPI_Alltoall() for different process arrival patterns

cesses do as much work as possible. One critical issue in our PAP aware algorithms is how to let every other process know who has already arrived at the call. Previous work on PAP aware MPI_Bcast() [3] has introduced control messages that would add extra overhead, especially for small messages. However, in our work we do not send distinct control messages and instead we utilize the inherent features of RDMA-based communication to notify the arrival of a process.

4.1 Notification Mechanisms for Early-Arrival Processes

The basic idea in our PAP aware MPI_Alltoall() is for each process to send its distinct data to the already-arrived processes as soon as possible. It is therefore very important to have an efficient mechanism in place to inform others of the early-arrival processes. For this, we have devised two different notification mechanisms for *zero-copy* and *copy-based* schemes used in RDMA-based communications. These notification mechanisms do not incur any communication overhead.

In the zero-copy approach, where the cost of data copy is prohibitive for large messages, the application buffers are registered to be directly used for data transfer. However, for an RDMA Write message transfer to take place each source process needs to know the address of the remote destination buffers. For this, each process will advertise its registered destination buffer addresses to all other processes by writing into their pre-registered and pre-advertised control buffers. This inherent destination address advertisement mechanism can be interpreted as a control message to indicate a process has arrived at the MPI_Alltoall() call.

Therefore, processes can poll their control buffers to understand which other process has already arrived at the collective call.

To avoid the high cost of application buffer registration for small messages, the copy-based technique involves a data copy to pre-registered and pre-advertised intermediate data buffers at both send and receive sides. The sending process can copy its messages to the pre-registered intermediate destination buffers using RDMA Write. Therefore, the received data in the pre-registered intermediate destination buffer can be used as a signal that the sending process has already arrived at the site. This can be checked out easily by polling the intermediate destination buffer.

4.2 RDMA-based Process Arrival Pattern Aware Direct Algorithm

Our base algorithm is the Direct alltoall algorithm. Let N be the total number of processes involved in the operation. In this algorithm, at step i, process p sends its message to process $(p + i)$ *mod* N, and receives a message from process $(p - i)$ *mod* N. To implement this algorithm, each process p first posts its RDMA Writes to all other processes in sequence (after it receives the destination buffer addresses). It then polls the completion queues to make sure its messages have been sent to all other processes. Finally, it waits to receive the incoming messages from all processes.

To make this algorithm PAP aware using zero-copy scheme, each process p polls its control buffers for the advertised remote destination buffer addresses starting from process $(p + i)$ *mod* N. It then sends its distinct data to the final destination buffers of the early-arrived processes using RDMA Write. Subsequently, it waits for the remaining processes to arrive in order to send its messages to them. Finally, each process waits for all incoming messages by polling its own destination buffers. The beauty of this PAP aware algorithm over the non-PAP aware algorithm is that a sending process will never get stuck for a particular process to arrive in order to proceed with the next message transfer.

Under the copy-based scheme, each process p polls its intermediate destination buffers, starting from process $(p - i)$ *mod* N. Any received data indicates that the corresponding process has already arrived. The process p then copies its messages using RDMA Write to all early-arrived processes. It then sends its data to the rest of processes who have not yet arrived. All processes also need to wait to receive messages from all other processes into their intermediate buffers, and then copy them to their final destination buffers.

4.3 RDMA-based Process Arrival Pattern and Shared Memory Aware Direct Algorithm

Up to this point, we have utilized the RDMA features of InfiniBand along with the PAP awareness in an effort to improve the performance of MPI_Alltoall(). Previous research has shown that shared memory intra-node communication can improve the performance of collectives for small to medium size messages [9, 15, 16]. It is interesting to see how this might affect the performance under different

PAP. For this, we propose a shared memory and RDMA-based PAP aware Direct algorithm for MPI_Alltoall() that has the following three phases:

Phase 1: Intra-node shared memory gather performed by a master process
Phase 2: Inter-node process PAP aware Direct alltoall among the masters
Phase 3: Intra-node PAP and shared memory aware scatter by a master

A master process is selected for each node (without loss of generality, the first process in each node). Phase 1 cannot be PAP aware because the master process has to wait for all intra-node messages to arrive into a shared buffer before moving on to the PAP aware Phase 2. Phase 2 is the same as the algorithm proposed in Section 4.2, and is performed among the master processes. In Phase 3, an intra-node shared memory and PAP aware scatter is devised. Because the master processes may arrive in Phase 2 at different times, this awareness can be passed on to Phase 3 by allowing the intra-node processes to copy their destined data available in the shared buffer to their final destinations without having to wait for data from all other masters.

In a shared memory but non-PAP aware Phase 3, a master process waits to receive the data from all other master processes. It then copies them all to a shared buffer and sets a shared *done* flag. All other intra-node processes poll on this flag, and once set they start copying their own data from the shared buffer to their final destinations.

In the shared memory and PAP aware Phase 3, we consider multiple shared done flags, one for the data from each master process (four flags in our 4-node cluster). As soon as a master process receives data from any other master process, it copies it to the shared buffer and then sets the corresponding *done* flag. All other intra-node processes poll on all *done* flags, and as soon as any partial data is found in the shared buffer they copy them to their final destination buffers.

5 Experimental Results

The experiments were conducted on a 4-node dedicated multi-core cluster, where each node is a Dell PowerEdge 2850 server having two dual-core 2.8GHz Intel Xeon EM64T processors (2MB of L2 cache per core) and 4GB of DDR-2 SDRAM. Each node has a two-port Mellanox ConnectX InfiniBand HCA installed on an x8 PCI-Express slot. Our experiments were done under only one port of the ConnectX HCA operating in InfiniBand mode. The machines are interconnected through a Mellanox 24-port MT47396 Infiniscale-III switch. In terms of software, we used the OpenFabrics Enterprise Distribution, OFED-1.2.5 [9], installed over Linux Fedora Core 5, kernel 2.6.20. For MPI, we used MVAPICH-1.0.0-1625.

5.1 Micro-benchmark Results

In this section, we present the performance results of the proposed algorithms, the RDMA-based PAPaware Direct (*PAP_Direct*), and RDMA-based PAP and Shared-memory aware Direct (*PAP_Shm_Direct*), and compare them with the

non-PAP aware RDMA-based Direct (*Direct*) and RDMA-based and Shared-memory aware Direct (*Shm_Direct*) algorithms as well as with the native MVAPICH on our cluster.

We have evaluated the proposed algorithms using both copy-based and zero-copy techniques for 1B to 1MB messages. The results shown in this section are the best results of the two schemes for each algorithm. We use cycle-accurate timer to record the time spent in an MPI_Alltoall() (1000 iterations) for each process, and then calculate the average time across all processes.

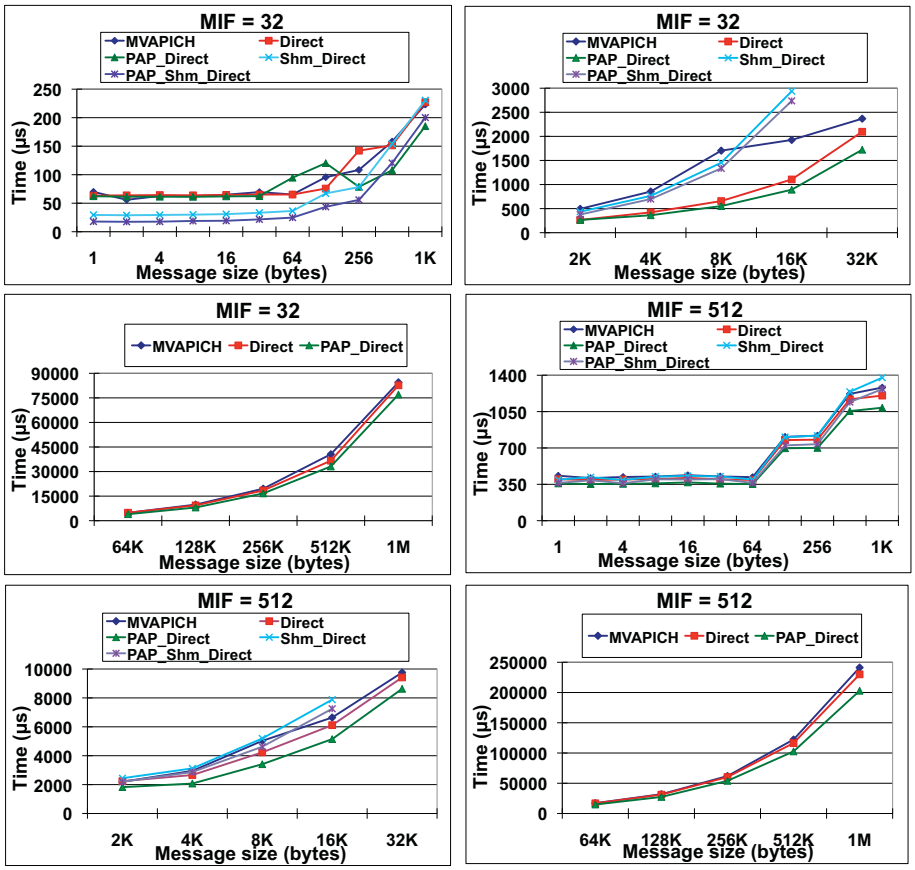

Fig. 2. Performance of MPI_Alltoall() running with 16 processes

Figure 2 compares our PAP aware algorithms with the native MVAPICH implementation and non-PAP aware versions, with MIF equal to 32 and 512. Clearly, the PAP aware algorithms, PAP_Direct and PAP_Shm_Direct, are better than their non-PAP aware counterparts for all message sizes. This shows that indeed such algorithms can adapt themselves well with different PAP. Our algo-

rithms are also superior to the native MVAPICH, with an improvement factor of 3.1 at 8KB for PAP_Direct and 3.5 at 4B for PAP_Shm_Direct, with MIF equal to 32. With a larger MIF of 512, the improvements are 1.5 and 1.2, respectively.

Comparing the PAP_Shm_Direct with PAP_Direct, one can see that the PAP_Shm_Direct is the algorithm of choice up to 256 bytes for MIF equal to 32. However, this is not the case for MIF of 512 where processes may arrive at the call with more delay with respect to each other. This shows that the shared memory version of our algorithm introduces some sort of implicit synchronization in Phase 1 that may degrade its performance under large maximum imbalanced factors.

To evaluate the scalability, we compare the performance of the PAP_Direct MPI_Alltoall() with those of MVAPICH and Direct for 4, 8, and 16 processes, as shown in Fig. 3 (shared memory algorithms are not shown due to limited data points). One can see that the proposed PAP aware algorithm has scalable performance and is always superior to the non-PAP aware algorithms. We have found similar results for other MIFs and messages sizes.

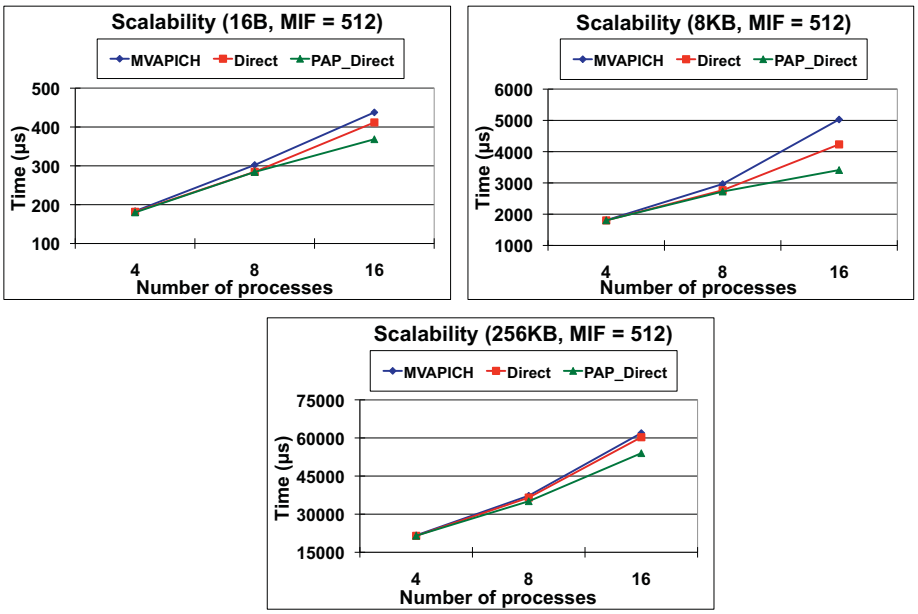

Fig. 3. MPI_Alltoall() scalability

In the previous micro-benchmark, the arrival time of each process is random. In another micro-benchmark, similar to [3], we control the number of late processes. In Figure 4, we present the results for MIF equal to 128 when 25% or 75% of processes arrive late. Our proposed algorithms are always better than their counterparts for the 25% case, and mostly better in the 75% case. The

PAP_Shm_Direct is always better than MVAPICH, although with a less margin in the 75% case.

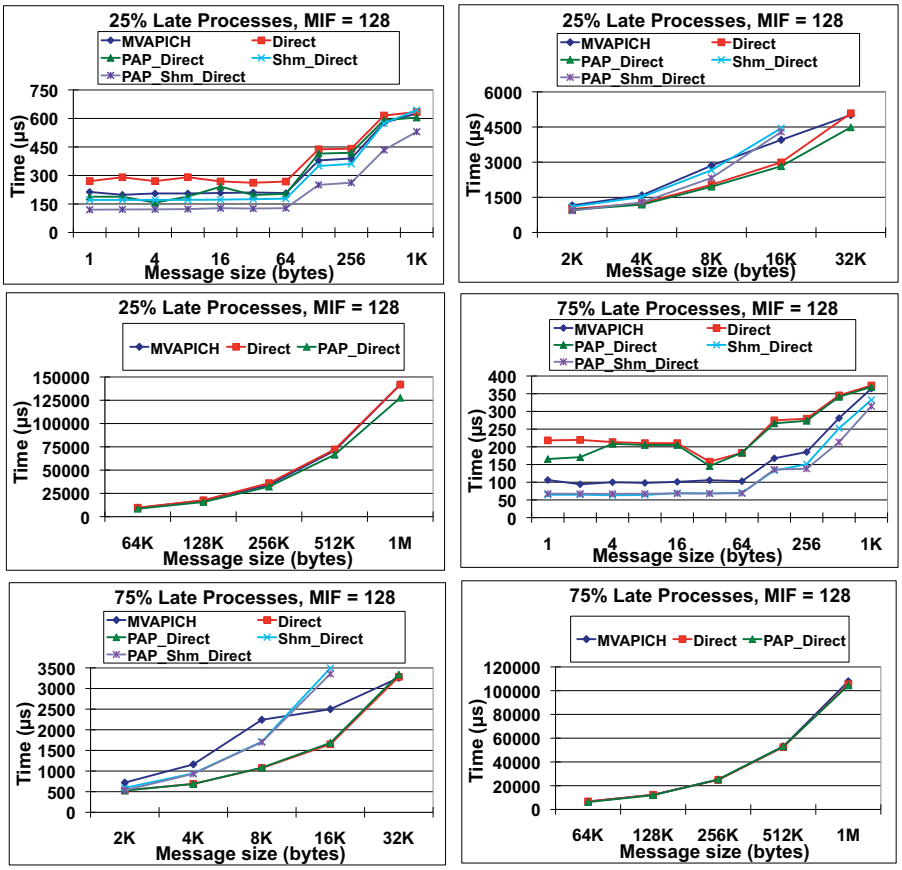

Fig. 4. Performance of MPI_Alltoall() with 25% and 75% late processes, 16 processes

5.2 Application Results

In this section, we consider the FT application benchmark from NAS 2.4 [11] to evaluate the performance and scalability of the proposed PAP aware MPI_Alltoall(). FT uses MPI_Allltoall() as well as a few other collectives. We have experimented with class B and C of FT, running with different number of processes, which use payloads larger than 2MB. Table 1 shows the PAP aware MPI_Alltoall() speedup over the native MVAPICH and the Direct algorithms for FT running with 4, 8, and 16 processes. Clearly, the proposed algorithm outperforms the conventional algorithms. The results also show that the PAP aware MPI_Alltoall() has modest scalability as speedup improves with increasing number of processes.

Table 1. PAP_Direct MPI_Alltoall() speedup over native MVAPICH and the Direct algorithms for NAS FT running with different number of processes and classes

	Speedup over native MVAPICH		Speedup over Direct algorithm	
	FT (Class B)	FT (Class C)	FT (Class B)	FT (Class C)
4 processes	1.08	1.01	1.16	1.04
8 processes	1.10	1.04	1.04	1.14
16 processes	1.14	1.17	1.42	1.63

6 Related Work

Study of collective communications has been an active area of research. Thakur and his colleagues [8] discussed recent collective algorithms used in MPICH [12]. They have shown some algorithms perform better depending on the message size and the number of processes involved in the operation.

Faraj and his associates [2] discussed the PAP in a set of MPI programs, which denotes the timing when different processes arrive at an MPI collective operation. They discovered that the time difference between the arrivals of each process at a collective call is usually large enough to affect the performance of the collective. An MPI broadcast across different PAP was proposed by Patarasul and Yuan [3], in which they inserted control messages in their algorithms to make the processes aware of and adapt to the PAP. Their algorithms were implemented at the MPI layer using MPI point-to-point operations.

Sur and others [13] proposed RDMA-based MPI_Alltoall() algorithms for InfiniBand clusters. Buntinas et al. [14] used different mechanisms to improve large data transfers in SMP systems. Tipparaju and others overlapped the shared memory and remote memory access communications in devising collectives [15]. Qian and Afsahi proposed efficient RDMA-based and shared memory aware allgather at the Elan-level over multi-rail QsNet^{II} clusters [9], and on InfiniBand ConnectX using its multi-connection capabilities [16].

7 Conclusions and Future Work

MPI_Alltoall() is one of the most communication-intensive primitives in MPI. Imbalanced PAP has an adverse impact on its performance. In this paper, we proposed RDMA-based and shared memory PAP aware MPI_Alltoall() algorithms without introducing any extra control messages.

The performance results indicate that the proposed algorithms perform better than their non-process arrival pattern counterparts when processes arrive at different times. They also outperform the native MVAPICH implementation by

a large margin, up to 3.1 times at 8KB for PAP_Direct and 3.5 times at 4B for PAP_Shm_Direct.

While this study was focused at MPI_Alltoall(), it can be directly extended to other collectives. The proposed techniques can be applied to other alltoall algorithms such as Bruck or recursive doubling. However, one has to bear in mind that due to synchronization between different steps of these algorithms they may not achieve the highest performance as in the Direct algorithm. We are currently investigating this. We also intend to experiment with a larger testbed and study other collectives to understand the true potential of PAP aware algorithms.

Acknowledgments. This work was supported in part by the Natural Sciences and Engineering Research Council of Canada Grant #RGPIN/238964-2005, and Canada Foundation for Innovation and Ontario Innovation Trust Grant #7154. We would like to thank Mellanox Technologies for the resources.

References

1. MPI: A Message Passing Interface standard (1997)
2. Faraj, A., Patarasuk, P., Yuan, X.: A Study of Process Arrival Patterns for MPI Collective Operations. International Journal of Parallel Programming, 36(6), 543–570 (2008)
3. Patarasuk, P., Yuan, X.: Efficient MPI_Bcast across Different Process Arrival Patterns. In: 22nd International Parallel and Distributed Processing Symposium (IPDPS) (2008)
4. InfiniBand Architecture, `http://www.infinibandta.org`
5. MVAPICH, `http://mvapich.cse.ohio-state.edu`
6. Mellanox Technologies, `http://www.mellanox.com`
7. Bruck, J., Ho, C.-T., Kipnis, S., Upfal, E., Weathersby, D.: Efficient Algorithms for All-to-all Communications in Multiport Message-passing Systems. IEEE Transactions on Parallel and Distributed Systems, 8(11), 1143-1156 (1997)
8. Thakur, R., Rabenseifner, R., Gropp, W.: Optimization of Collective Communication Operations in MPICH. International Journal of High Performance Computing Applications, 19(1), 49–66 (2005)
9. Qian, Y., Afsahi, A.: Efficient Shared Memory and RDMA based Collectives on Multi-rail QsNet^{II} SMP Clusters. Cluster Computing, Journal of Networks, Software Tools and Applications, 11(4), 341–354 (2008)
10. OpenFabrics Alliance Homepage, `http://www.openfabrics.org`
11. NAS Benchmarks, version 2.4, `http://www.nas.nasa.gov/Resources/Software/npb.html`
12. MPICH, `http://www.mcs.anl.gov/research/projects/mpich2`
13. Sur, S., Jin, H.-W., Panda, D.K.: Efficient and Scalable All-to-all Personalized Exchange for InfiniBand Clusters. In: 33rd International Conference on Parallel Processing (ICCP), pp. 275–282 (2004)
14. Buntinas, D., Mercier, G., Gropp, W.: Data Transfers Between Processes in an SMP System: Performance Study and Application to MPI. In: 35th International Conference on Parallel Processing (ICPP), pp. 487–496 (2006)

15. Tipparaju, V., Nieplocha, J., Panda, D.K.: Fast Collective Operations using Shared and Remote Memory Access Protocols on Clusters. In: 17th International Parallel and Distributed Processing Symposium (IPDPS), (2003)
16. Qian, Y., Rashti, M.J., Afsahi, A.: Multi-connection and Multi-core Aware All-Gather on InfiniBand Clusters. In: 20th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS), pp. 245–251 (2008)