Evaluation of Topology-Aware All-Reduce Algorithm for Dragonfly Networks

Junchao Ma, Dezun Dong(✉), Cunlu Li, Ke Wu, and Liquan Xiao

National University of Defense Technology, Changsha, China
{majunchao,dong,cunluli,wuke13,xiaoliquan}@nudt.edu.cn

Abstract. Dragonfly is a popular topology for current and future high-speed interconnection networks. The concept of gathering topology information to accelerate collective operations is a very hot research field. All-reduce operations are often used in the research fields of distributed machine learning (DML) and high-performance computing (HPC), because All-reduce is the key collective communication algorithm. The hierarchical characteristics of the dragonfly topology can be used to take advantage of the low communication delay of adjacent nodes to reduce the completion time of All-reduce operations. In this paper, we propose g-PAARD, a general proximity-aware All-reduce communication on the Dragonfly network. We study the impact of different routing mechanisms on the All-reduce algorithm, and their sensitivity to topology size and message size. Our results show that the proposed topology-aware algorithm can significantly reduce the communication delay, while having little impact on the network topology.

Keywords: All-reduce operation · Dragonfly topology · Collective communication

1 Introduction

Dragonfly are typically deployed in many high-performance computer systems, including the Cray Cascade system [7], Titan [6] and Trinity [3]. The Cray Slingshot network is designed for continuous computation and also uses dragonfly topology [14].

Recently, it is popular to continuously improve the performance of collective operations so that it can run efficiently on various hardware and software platforms. All-reduce aggregates the values of all processes and then sends the values back to all nodes. All-reduce simplifies a complex set of point-to-point communications, making it easier for programmers to perform parallel and distributed programming. In addition, All-reduce operations can effectively separate application and interface developers, and contribute to the portability of functions and performance between applications and interfaces.

Published by Springer Nature Switzerland AG 2022
C. Cérin et al. (Eds.): NPC 2021, LNCS 13152, pp. 243–255, 2022.
https://doi.org/10.1007/978-3-030-93571-9_19

There are already many All-reduce implementation algorithms in the mpi library. According to the number of processes and the size of the message, different algorithm choices can speed up the efficiency of communication. The most popular All-reduce algorithms are the ring algorithm and the halving and doubling algorithm. Ring algorithm [15] is an algorithm that makes full use of bandwidth. However, it is mainly suitable for tasks consisting of long messages because the number of steps executed increases linearly as the number of processes increases. Conversely, the halving and doubling algorithm [16] performs the least communication steps and achieves the least delay. While delay-sensitive and tasks consisting of small messages often use the halving and doubling algorithm, it suffers from considerable bandwidth overhead. There are also several hybrid methods, which complete the operations by combining different types of sub-step operations, including reduce-scatter and All-gather, as well as reducing and broadcasting [13].

G-PAARD is designed as a topology-aware algorithm to accelerate the All-reduce communication. The communication mode of the existing acceleration algorithm implemented in the dragonfly network may cause network congestion and waste of link resources[1]. The key idea behind g-PAARD is to decompose All-reduce operations into reduce-scatter and All-gather modes in a topology-aware way. The All-reduce operation with g-PARRD can be completed in just 6 steps. Therefore, g-PAARD utilizes local communication with neighbors to minimize global communication and reduce overhead. Experimental results show that g-PAARD is superior to these state-of-the-art solution in a particular context.

The main contributions of this design can be summarized as follows:

- We have shown that existing algorithms are not well suited for Dragonfly networks, resulting in poor performance and limited overall network throughput.
- We propose g-PAARD, which permits the scheduling of an end-to-end solution to alleviate congestion.
- A comprehensive evaluation of the proposed algorithm has been performed and demonstrated that higher performance can be achieved in a particular context.

2 Motivation

2.1 Dragonfly

Kim et al. [10] introduced Dragonfly networks, it has become one of the most popular topologies in high cardinality HPC interconnect networks. Dragonfly has the advantages of high scalability, small diameter and low cost. The standard Dragonfly topology employs a two-tier structure, as shown in the Fig. 1.

[1] For example, using standard algorithms, both nodes require at least 3 hops, and in some cases up to 6 hops, to facilitate communication at each step of the topology.

The routers are interconnected using a fully connected intra-group topology to create a group. A palmtree for $a = 4$ is included in Fig. 1. The number of links connecting each router to the local compute node is p; the number of routers in each group is a; the number of global links each router connects to routers in other groups is h; and the number of groups be g.

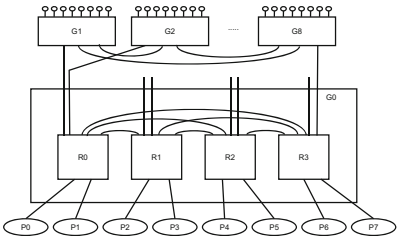


Fig. 1. An example Dragonfly topology with 9 groups and each group contains 4 routers. DF($p = 2$, $a = 4$, $h = 2$, $g = 9$)

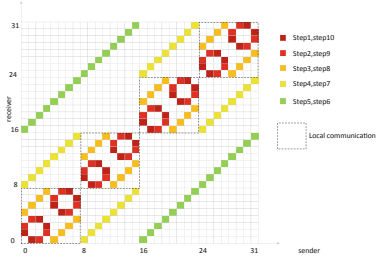


Fig. 2. Communication pattern of HD for 32 nodes in DF(2,4,2) network.

In the dragonfly topology, data packets are routed along the minimum or non-minimum path. The minimum path is any path from the source node to the target node, and contains at most one global link. The smallest path takes one hop in the source group (from the source switch to the switch with the global link to the destination group), then travel through the global link to the destination group, and finally selects the local link to the destination at the target group. Depending on the source location and target location, the minimum path may have fewer hops.

The Minimal routing (MIN) scheme routes packets only through minimal paths, thus minimizing resource usage. MIN works well for traffic patterns where the load can be evenly distributed, such as random uniform traffic. However, as the number of links between each pair of groups is small, MIN routing performs poorly for adversarial traffic, particularly where most communication arises between two groups.

Generally, the delay of the global link is greater than the delay of the local link. In the dragonfly topology, there are some pairs of nodes connected by global links between different groups. These node pairs can communicate at the same time to maximize the utilization efficiency of the global link. In this article, we will consider the characteristics of Dragonfly to design an efficient All-reduce algorithm on the Dragonfly network. This can significantly reduce communication time by coordinating communication between adjacent nodes and completing the All-reduce operation.

2.2 Existing All-Reduce Algorithms

– Ring Algorithm (Ring)

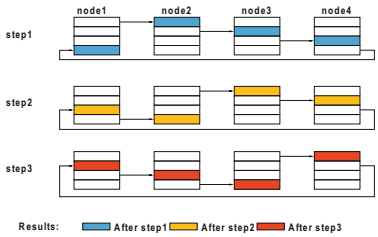


Fig. 3. Ring algorithm for all-reduce

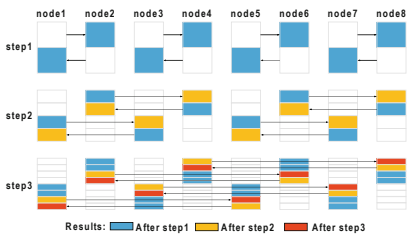


Fig. 4. Halving-doubling algorithm for all-reduce

Figure 3 present a ring algorithm for All-reduce. Utilizing chunked data transfer and reduction can optimize the ring algorithm. The entire ring process is divided into two major steps: the first step is scatter-reduce, and the second step is All-gather. First we have n nodes, then we divide the data (equal) on each node into n blocks, and assign each node its left and right neighbors (the left neighbor of node 1 is node 0, and the right neighbor is node 2...), and then start to perform $n-1$ operations. At the i-th operation, the node j will send its $(j-i)\%n$-th block of data to node $j+1$, accept the $(j-i-1)\%n$ block of data from node $j-1$, and perform a reduce operation on the received data. When $n-1$ operations are completed, the first scatter-reduce step of ring-allreduce has been completed. At this time, the $(i+1)$-th node of the i-th node % n blocks of data have collected the $(i+1)\%n$ block of all n nodes.

Then, perform All-gather again to complete the algorithm. The second phase for All-gather is very simple. It is to pass the $(i+1)\%$ n-th block of the i-th node to other nodes through $n-1$ passes, and it is also in the i-th pass. Node j sends its $(j-i-1)\%n$-th block of data to the right neighbor, and accepts the $(j-i-2)\%n$-th data of the left neighbor, but the received data does not need to be like the first step to reduce, but directly replace its own data with the received data.

The ring algorithm [15] can make full use of bandwidth and is suitable for large message tasks. However, as the number of nodes increases, the number of steps executed by the algorithm increases linearly. This greatly increases the delay time of small message communication. In the dragonfly topology, each step of the ring algorithm needs to use the global link for communication, which will further increase the complete time.

– Halving-doubling Algorithm

Figure 4 gives an example to show the All-reduce process of the HD algorithm. The entire HD process is divided into two major steps, the first stage is scatter-reduce, and the second stage is All-gather. In the first step, node 0 and the node with a distance of 1 (node 1) exchange half of the data aggregated in the previous step (the first step) and aggregate. In step $\log_2 P$, node 0 and the node at distance 2^{n-1} exchange half of the data aggregated in the previous step $\log_2(P\text{-}1)$ and aggregate. The second stage is the reverse process of the first stage. In the first step, node 0 exchanges aggregated data with a node at a distance of 2^{n-1}. In the second step, node 0 exchanges updated aggregated data with the node at a distance of 2^{n-2}. The node 0 and the node with a distance of 1 (node 1) exchange half of the aggregated data. Finally, all nodes get aggregated data, and the Allreduce operation is completed.

The HD algorithm is suitable for small message tasks and the number of nodes is a power of 2. At this time, the HD method greatly reduces the communication delay because of the few communication steps ($2\log_2 P$). However, in the dragonfly topology, since the distance between nodes may be 3 hops, each step of communication will have additional global link delay overhead. This will bring performance degradation. At the same time, the size of dragonfly topology is usually not a power of 2, which will add an extra step of communication overhead.

As shown in Fig. 2, it is a communication pattern of HD for 32 nodes in DF(2,4,2) network. Horizontal axis represents the sending node, and the vertical axis represents the receiving node. Different colors represent different steps. Communications carried out at the same time has been marked with the same color. It can be seen that nearly half of the communication is adversarial traffic, where nodes in one group should communicate with nodes in another group. These communications take multiple hops and add network contention.

3 The g-PAARD Design

3.1 g-PAARD Algorithm in Dragonfly

Notation: In the following we use the term *global node* to refer to the node connecting with global link directly. We call the *local node* to refer to the node connecting with global link indirectly.

Node Placement: The algorithms presented hereafter are based on the assumption that some information about the topology and the node placement can be obtained. Each node, router, and group is assigned a unique node, router, and group *id*, respectively. Any process can have access to the node *id* of any other process belonging to the same application.

g-PAARD Algorithm consist of six steps. First, the data of the node is evenly divided into g(which is the size of group) parts. *Local node* send specific $1/g$ data to *global node*. *Global node* send specific $1/g$ data to other *global node* and, *Global node* receive and reduce data from *local node* and other *global node* simultaneously. In the second step, *global node* in different group communication

with each other through the directly connected global link. *Global node* send specific data to the pair *global node*. Simultaneously, every *global node* receive and reduce the specific data. In the third step, the aggregation data in second step is evenly divided into $p \times a$ (which is the size of nodes in a group) parts. Nodes in the same group communicate with each other. Node send specific data to other nodes in the same group. Simultaneously, every node receive and reduce specific data from other nodes in the same group. After third step, every node aggregate the $\frac{1}{g \times p \times a}$(which is the number of all nodes). The reduce-scatter phase is completed. The next three step is reversed to complete the All-gather phase.

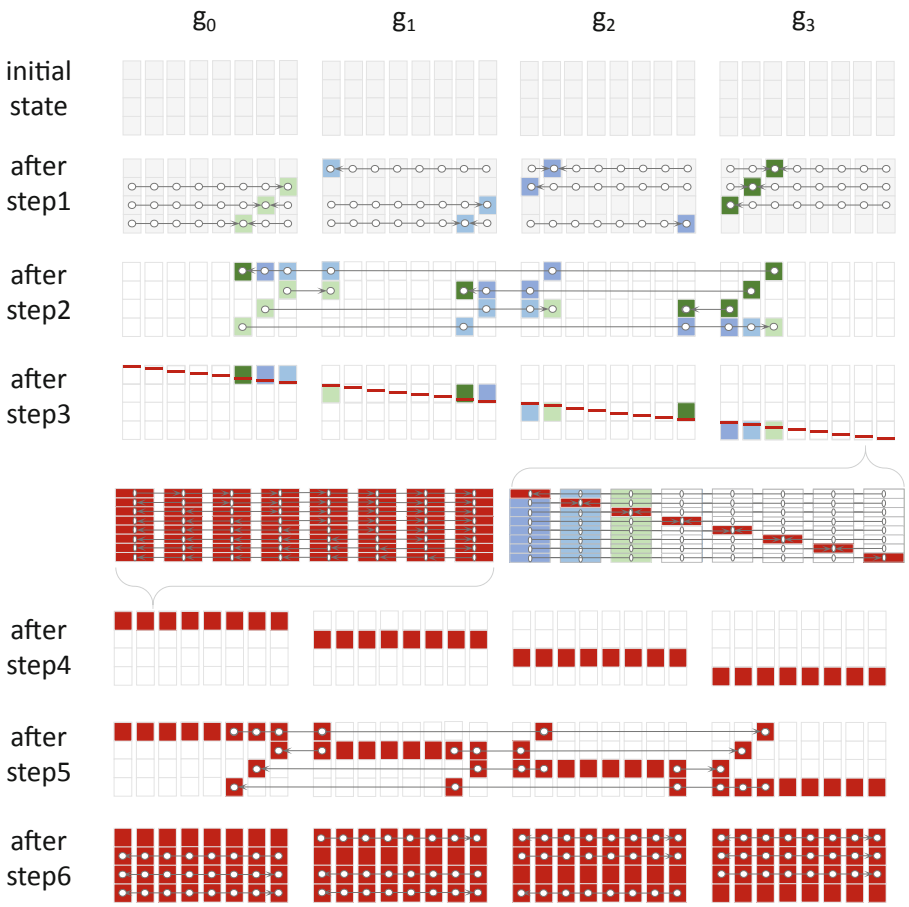


Fig. 5. g-PAARD for a 32 nodes in DF(2,4,2) networks

3.2 An Example of the g-PAARD Algorithm for 32 Nodes in DF(2,4,2) Network

As shown in Fig. 5, it is an example of the g-PAARD algorithm for 32 nodes in DF(2,4,2) network. We assume that 32 nodes are allocated in four groups, there are 8 nodes in a group. And a global palmtree arrangement is deployed [4].

g-PAARD requires two processing phases, each with several communication steps. As shown in 5, 8 columns per group represent data in 8 nodes, and 32 columns are given to represent 32 nodes. Due to the existence of four groups, the data in each node is divided into four parts (marked as a first to fourth data set). At first, each node n has its own local data.

In Fig. 5, we take nodes in $g0$ as an example to show the All-reduce process of g-PAARD. In the first step, for sender, *local node*(data are marked as blank square) sends the data set to *global nodes*(some data are marked as colored square) in $g0$, and *global nodes* sends the data set to other *global nodes* in $g0$. For the receiving end, the *global nodes* receive and aggregate data sets from all other nodes simultaneously. Remarkably, this kind of communication between nodes is direct and requires only one hop. After step 1, one quarter of the data in each node has not been transferred because it will be aggregated in step 2.

In the second step, communication takes place between the global groups. There are 6 global links between groups, so that 6 pairs of nodes can communicate at the same time. These communications can be accomplished with only a hop at the same time. Take the *global nodes* in $g0$, which sends aggregated data after step 1 to their counterparts (data marked green in other groups). At the same time, *global nodes* receive and aggregate data sets from their pairs. After step 2, all four groups received their specific data. Therefore, another step needs to be taken to complete the reduce-scatter phase.

Because there are eight nodes in a group, the data aggregated after step 2 can be divided into eight groups in step 3. Taking the node in $g0$ as an example again, the first set of data for each node in $g0$ is divided into eight parts (1, i (1∼8)), where node 1 sends (1, i) data to node i in $g0$. At the same time, node 1 receives and aggregates data from other nodes in $g0$. After the third stage, the reduce-scatter phase of the data was completed, with 32 nodes collecting the data respectively. The All-gather phase is similar to the process leading to the final result. As a result, the g-PAARD algorithm reduces the dependent length and hops per step compared to the Ring algorithm and the HD algorithm, as shown in Table 1.

The next three step is a reversed phase to complete All-reduce operation. In the fourth step, Taking nodes in $g0$ as an example again, node 1 sends the aggregates data after step 3(marked as red) to other nodes in $g0$. Simultaneously, node 1 receives the aggregated data from other nodes in $g0$. After step 4, every node has 1/4 aggregated data.

In fifth step, *global nodes* in $g0$ send the aggregated data(marked as red) to their pair in other group, simultaneously *global nodes* receive aggregated data from *global nodes* in other group.

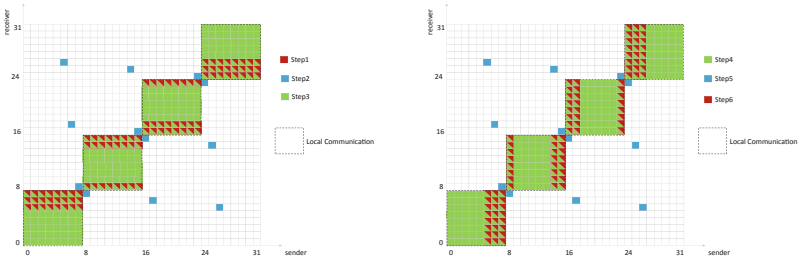
In sixth step, it is similar with step 1. Taking nodes in $g0$ as an example, *global nodes* send the data received in step 5 to other nodes in $g0$. After step 6, all

nodes get the completed aggregated data. The All-reduce operation finish. While our example uses 32 nodes, the g-PAARD algorithm is applicable for arbitrary nodes counts.

As shown in Fig. 6, it is communication pattern of g-PAARD for 32 nodes in DF(2,4,2), horizontal axis represents the sending node, and the vertical axis represents the receiving node. Different colors represent different steps. Communications carried out at the same time has been marked with the same color. All the communication are proximity, most of communications occur in intra-group. thus, it reduce the global link latency compare to HD algorithm. And, The communications marked with blue occur in inter-group, and only take one hop. So, it can alleviate the global link contention.

Table 1. g-PAARD achieves good tradeoffs by minimizing hops and length of dependency

Algorithm	Ring	Halving doubling	g-PAARD
Minimum hop	1	3	1
No. of dependant steps	2(P−1)	2logP	6



(a) (b)

Fig. 6. Communication pattern of g-PAARD for 32 nodes in DF(2,4,2) network

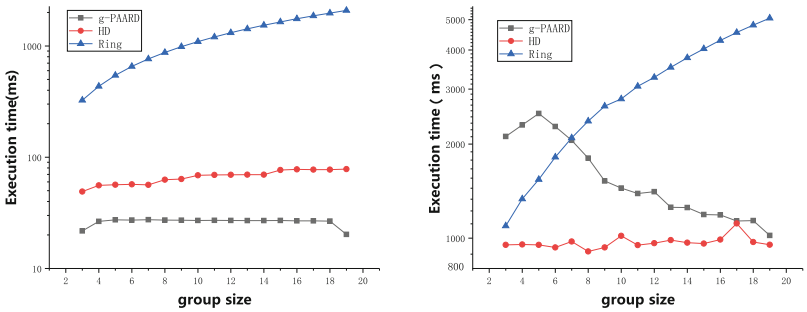
4 Evaluation

4.1 Evaluation Methods

Our simulator is based on the Booksim router model, which emphasizes the precise simulation of the cycle of hardware components. In our evaluation, the accuracy of network simulations was at the microchip level. We expanded Booksim to support parallel distributed simulation for large scale networks to overcome the drawbacks of Booksim serial execution. In addition, we extend it to support parallel distributed emulation and model a complete MPI library (such as an SST emulator) to evaluate the full emulation of the entire process from MPI Call to generate network traffic.

4.2 Evaluation Results

A. Node Allocation

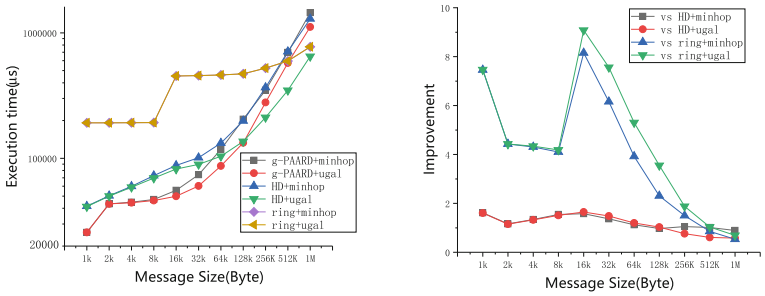


(a) Message size = 1KB (b) Message size = 1MB

Fig. 7. Execution time when All-reduce 1 KB and 1 MB on a (3,6,3) 342 nodes Dragonfly network with minhop routing strategies. The job runs alone on the network

Figure 7 show the execution time when All-reduce 1 KB and 1 MB on a (3,6,3)342 nodes Dragonfly network with minhop routing strategies. When message size is 1 KB, g-PAARD perform better than HD and Ring algorithm. However, when message size is 1 MB, g-PAARD perform worse in small group size. g-PAARD relies more on a complete network to play its performance advantages. When group size is small, the global link decrease. g-PAARD lost the advantage of using all global links at the same time.

B. Router Algorithm



(a) Execution time (b) Improvement

Fig. 8. Performance of three algorithms for 32 nodes in DF(2,4,2) 72 nodes

The performance of three algorithms is evaluated using minhop routing algorithm and UGAL routing algorithm. The topological scale is 32 nodes of Dragonfly(2,4,2) 72 nodes, and the result is shown in Fig. 8. When the message size

is between 1 k and 256 k, the performance of the g-PAARD algorithm is significantly better than the HD algorithm or the ring algorithm, because link latency has a greater weight in the total time cost. However, when the size of a message is large, the gap decreases because communication latency becomes a major factor in the overall time cost. For g-PAARD and HD algorithms, the choice of routing algorithm also affects execution time, while UGAL routing algorithm is beneficial to improve execution speed. This is because the side effects of network congestion increase with the size of messages. Because of its long dependence chain, the ring algorithm has the worst performance, which increases startup delay.

C. Topology Scale

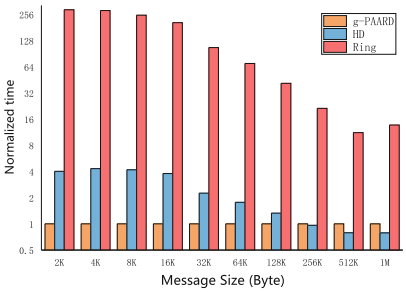


Fig. 9. Normalized time of three algorithm with minhop route algorithm in DF(4,8,4) 1056 nodes

Figure 9 illustrates the normalized time of the three algorithms using the minhop route algorithm in DF(4,8,4) 1056 nodes. The data size was varied only from 2 K to 1 MB in order to ensure the ring algorithm could be run. Figure 9 presents the normalized time of the three algorithms. Here, the g-PAARD displays a 4.33× speed increase over the halving-doubling algorithm at 4 kB and a 287.34× speed increase over the ring algorithm at 2KB with the minhop route algorithm. As compared with both the halving-doubling algorithm and the ring algorithm, Fig. 9 illustrates the normalized time of three algorithms using minhop routing in DF (4,8,4) 1056 nodes. The data size ranges from 2 K to 1 MB to ensure that the ring algorithm can run. Figure 9 shows the normalized time of the three algorithms. Here, g-PAARD shows accelerated growth 4.33× over recursive doubling algorithm and 287.34× over ring. Compared to HD and ring algorithms, g-PAARD reduces the time required more as the topology scale increases, as the cost of link latency has a higher weight when node count is larger. Nevertheless, when data sizes are particularly large, the performance of the three algorithms becomes generally similar, as the communication cost has a higher weigh. When the topology scale is larger, the lengthy of the dependency chain has greater weight in terms of the cost.

5 Related Works

The approach of gathering topology information and leveraging this knowledge to design better algorithms has been applied to many systems in the past. For example, HAN [11] build upon the existing collective communication infrastructure, reuse these existing algorithms as submodules, and combine them to perform efficient and hierarchical collective operations. MagPIe [9] is another MPI system designed to construct collective operation trees in heterogeneous communication environments. MPICH2 [1] groups processes by nodes to limit the number of inter-node communication. Previous work [12] use trees that stratify the network deeper than two layers for further optimization. In work [8], it propose a topology-aware collective algorithms for large scale supercomputing systems that are tightly coupled through inter-connects such as InfiniBand. Work [2] propose a scalable network discovery service for InfiniBand available at the user level. With this service, it design a network-topology-aware MPI library to provide a topology aware mapping of the MPI processes. [5] address the application of pipelining to optimize a blocking All-reduce, for very large messages, of up to 1 GB, in the context of distributed DNN training on clusters of computer nodes. However, we utilize locality in Dragonfly interconnect topology to propose a topology-aware algorithm to minimize the communication latency in computing All-reduce.

6 Conclusion

The scale of deployment of supercomputers has increased year by year. These clusters provide a lot of computing resources for application developers. The demand for topology-aware collective operations is also increasing. This paper thus presented g-PAARD for use in Dragonfly systems. Essentially, g-PAARD designs each step of peer-to-peer communication to create a six-step process, regardless of node allocation. This ensures that at each step, each node communicates with its partner node through at most one hop. Therefore, gPAARD achieves the goal of minimum hops and short dependency chain more effectively than the previous All-reduce operation scheme. According to the simulation results, compared with the halving and doubling algorithm and the ring algorithm, the speed of g-PAARD is increased up to 4.33× and 287.34×. While, With today's supercomputer sizes and people relying more on group collectives, optimization of communication in group can be further researched. And, straggler processors is a potential performance bottleneck to collective operation, it is interesting to explore the sensitivity of straggler processors.

Acknowledgment. We thank the anonymous reviewers for their insightful comments. We gratefully acknowledge members of Tianhe interconnect group at NUDT for many inspiring conversations. The work was supported by the National Key R&D Program of China under Grant No. 2018YFB0204300, the Excellent Youth Foundation of Hunan Province (Dezun Dong), and the National Postdoctoeral Program for Innovative Talents under Grant No. BX20190091.

References

1. Hierarchical collectives in mpich2. Springer-Verlag (2009). https://doi.org/10.1007/978-3-642-03770-2_41
2. Design of a scalable infiniband topology service to enable network-topology-aware placement of processes. In: SC 2012: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (2013)
3. Archer, B.J., Vigil, B.M.: The trinity system. Technical report, Los Alamos National Lab. (LANL), Los Alamos, NM (United States) (2015)
4. Camarero, C., Vallejo, E., Beivide, R.: Topological characterization of hamming and dragonfly networks and its implications on routing. ACM Trans. Archit. Code Optim. (TACO) **11**(4), 1–25 (2014)
5. Castelló, A., Quintana-Ortí, E.S., Duato, J.: Accelerating distributed deep neural network training with pipelined MPI allreduce. Cluster Comput. **24**(4), 3797–3813 (2021). https://doi.org/10.1007/s10586-021-03370-9
6. De, K., et al.: Integration of panda workload management system with titan supercomputer at OLCF. J. Phys. Conf. Ser. **664**, 092020. IOP Publishing (2015)
7. Faanes, G., et al.: Cray cascade: a scalable hpc system based on a dragonfly network. In: SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. pp. 1–9. IEEE (2012)
8. Kandalla, K.C., Subramoni, H., Vishnu, A., Panda, D.K.: Designing topology-aware collective communication algorithms for large scale infiniband clusters: case studies with scatter and gather. In: IEEE International Symposium on Parallel & Distributed Processing (2010)
9. Kielmann, T., Hofman, R., Bal, H.E., Plaat, A., Bhoedjang, R.: MagPIe: MPI's collective communication operations for clustered wide area systems. In: ACM SIGPLAN Notices (1999)
10. Kim, J., Dally, W.J., Scott, S., Abts, D.: Technology-driven, highly-scalable dragonfly topology. In: 2008 International Symposium on Computer Architecture, pp. 77–88. IEEE (2008)
11. Luo, X., et al.: HAN: a hierarchical autotuned collective communication framework. In: 2020 IEEE International Conference on Cluster Computing (CLUSTER), pp. 23–34 (2020). https://doi.org/10.1109/CLUSTER49012.2020.00013
12. Lusk, E., de Supinski, B.R., Gropp, W., Karonis, N.T., Bresnahan, J., Foster, I.: Exploiting hierarchy in parallel computer networks to optimize collective operation performance. In: Parallel and Distributed Processing Symposium, International, p. 377. IEEE Computer Society, Los Alamitos, CA, USA, May 2000. https://doi.org/10.1109/IPDPS.2000.846009
13. Rabenseifner, R.: Optimization of collective reduction operations. In: Bubak, M., van Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds.) ICCS 2004. LNCS, vol. 3036, pp. 1–9. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24685-5_1
14. Sensi, D., Girolamo, S., McMahon, K., Roweth, D., Hoefler, T.: An in-depth analysis of the slingshot interconnect. In: 2020 SC20: International Conference for High Performance Computing, Networking, Storage and Analysis (SC), pp. 481–494. IEEE Computer Society (2020)

15. Thakur, R., Gropp, W.D.: Improving the performance of collective operations in MPICH. In: Dongarra, J., Laforenza, D., Orlando, S. (eds.) EuroPVM/MPI 2003. LNCS, vol. 2840, pp. 257–267. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-39924-7_38
16. Thakur, R., Rabenseifner, R., Gropp, W.: Optimization of collective communication operations in MPICH. Int. J. High Perform. Comput. Appl. **19**(1), 49–66 (2005)