Low-Latency Collective Communication for High-Performance Computing Networks

by

PHAM TRUNG KIEN

Doctor of Philosophy

Department of Informatics
School of Multidisciplinary Sciences

The Graduate University for Advanced Studies, SOKENDAI

March 2026

Abstract

Network link bandwidth has increased to 400Gbps or higher through high-density optical integration, e.g., Co-Packaged Optics (CPO) and onboard silicon-photonics transceivers. The high bandwidth enables the design of high-radix interconnection networks, reducing their network diameter to as few as two or three in parallel computer systems, including cloud data centers and supercomputers. This dissertation focuses on Kautz network topology with a diameter of two and the two-dimensional fully connected (2dfc) network topology, which also has a diameter of two. Connecting the maximum number of nodes for a given degree while maintaining a diameter is an open graph problem, known as the degree-diameter problem (DDP). Among the well-known strong candidates for this problem, we select the Kautz graph. The Kautz graph has $d^{k-1}(d+1)$ nodes, where k and d denote the diameter and degree, respectively. We apply the Kautz graph as the interconnection network topology. This dissertation presents multi-port message-combine collective communications to fully exploit the large number of links in Kautz networks. Our main idea is the design of "multi-port" operations, that utilize all outgoing links of a source node in collective communication. We evaluate the multi-port message-combine collective communication using version 3.28 of the SimGrid framework. In the simulation, the startup latency for each communication and link bandwidth are set to 1μs and 100 Gbps, respectively. Then, we measure the execution time for various messages sizes. SimGrid evaluation results illustrate that the proposed algorithm achieves substantial performance gains over an existing competitor, being up to 11× faster in alltoall benchmarks.

Although the proposed multi-port message-combine collective communication algorithm performs well on the Kautz network topology, existing large parallel

computer systems have not employed uni-directional interconnection networks, including Kautz. This limitation arises from practical concerns, such as the absence of efficient switch-by-switch flow control. To address this issue more practically, this dissertation also presents multi-port message-combine collective communication in a bi-directional 2dfc network topology. In a 2dfc, the number of dimensions equals two. In each dimension, all nodes are fully connected. When the number of dimensions equals two, the topology corresponds to HyperX and Generalized Hyper Cube (GHC). Thus, the proposed multi-port message-combine operation on a 2dfc can be leveraged, further enhancing its applicability to state-of-the-art parallel computer systems. SimGrid evaluation results demonstrate that the proposed multi-port message-combine operations improve throughput by up to 1.38x compared with a competitor. Although the proposed multi-port message-combined collective communication can, in principle, be applied to other high-radix network topologies, the latency overhead of the message-combine operation is significant at each network interface. Therefore, the proposed multi-port message-combine collective communications particularly well suited for diameter-two topologies such as Kautz and 2dfc network topologies.

Acknowledgments

I would like to express my thanks to those who supported me and helped me make this dissertation possible.

First and foremost, I would like to thank Prof. KOIBUCHI Michihiro who is the most supportive advisor I have ever known. His strong scientific and personal support made it possible for me to complete this dissertation. Throughout my work, I have been persistently impressed by his analytical skills and understanding of a very broad range of scientific problems. I am also very impressed about his determined working style, especially his patience and consistency. Recently, when I talked to him more, I have realized that I am still just a kid. I regret not talking with him more earlier. From the bottom of my heart, I would like to thank him, who took the greatest care of me during my PhD studies in Japan.

Second, I would like to thank Dr. Truong Thao Nguyen (AIST) who actively helps me in life and work from the beginning days when I came to Japan. I have learned from him a lot about the research: writing skill, how to do the experiment, etc. persistence and concentration are two things that I want to learn from him. He and his family always silently looking at my progress and give me comments. I do not know how to thank him.

I also would like to express my sincere gratitude to my other advisors, Professor AIDA Kento, Professor FUKUDA Kensuke, Professor GOSHIMA Masahiro, Professor YAMAKI Hayato and Professor TAKEFUSA Atsuko whose encouragement, insightful comments and hard questions helped me improve the dissertation a lot. I also like to acknowledge the NII international support team for the endless effort to provide a good research environment.

I would like to show my warm thanks to all Koibuchi lab members for the friendly

atmosphere which made my every working day be comfortable, and for the thoughtful comments in the group meetings which helped improving my research.

I am deeply grateful to Vietnamese guys in NII who made my days full of joys. Especially, I would like to thank Le Van An san for enduring me in daily life, sharing me about the investment, drinking with me when I get bored, encouraging me when I get stuck in the research. And Dinh Thi Ha Ly san for holding parties, caring about me when I got sick. Le Tien Thanh san who always by my side for every aspects of life. I have learned from him a lot.

Last but not least, from the bottom of my heart, I would like to send the warmest thanks to my family, especially my Parents, my sister for the understanding and sacrifice they always give me.

Finally, my deepest thanks go to my wife, who has always cared for me and stayed by my side in moments of joy and sorrow. She has always shared my difficulties with deep empathy and given me thoughtful advice when I faced important choices. I love you so much.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Overview

High-performance computing (HPC) systems, such as supercomputers or cloud data centers, have become a critical infrastructure of our digital society. The increasing scale and computational complexity of HPC applications (e.g., big data analytics and deep learning) demand continuous performance improvements, especially in collective communications. Several large scale systems have been designed with these workloads in mind. For example, ABCI and NVIDIA Selene target deep learning applications, while Fugaku supports AI, big data analysis, and scientific computing. These systems have hundreds of thousands of nodes to solve a common big problem. When the size of systems increase, efficient use of system resources has become a crucial challenge.

In parallel with the development of cutting-edge high-throughput and low-latency hardware components, interconnection networks and communication algorithms play a crucial role in determining system performance. An interconnection network is the infrastructure that connects computing components within a system. It represents the

physical connections between devices. A well-designed interconnection network significantly contributes to the performance of an HPC system. A clear example is that if a ring interconnection were used for El Capitan (currently the largest supercomputer [1]), running applications on this hypothetical system would be nearly impossible due to excessively high latency, with hop counts reaching tens of thousands.

While an interconnection network provides the physical connections between processing components, communication algorithms determine the logical connections among these components. These algorithms decide the placement and ordering of packet transmissions within the system. Applying an inappropriate communication algorithm to a given topology can drastically degrade performance. In particular, collective-communication performance becomes more crucial in the system. Collective communication involves a group of nodes and require coordination among all members of the group, such as reduce, scatter, and all-to-all. A large-scale analysis of two years of log data from the Mira supercomputer [2], involving over 100,000 production jobs, was reported in [3]. The results indicate that collective communications play a more prominent role than point-to-point communications in real supercomputer workloads. Interestingly, the applications that consume the most core-hours frequently exhibit high message-passing time fractions. About half of all applications spend one third of their execution time in collective communications on the supercomputer. Further, 15% of jobs spend more than 60% of their time in MPI. These observations suggest that improving the efficiency of collective communication can shorten execution time by reducing the amount of time processes spend waiting for communication to complete.

Many HPC systems in the Top500 [1] have used Fat-tree [4] network topology. However, recent work shows the performance limitation of Fat-tree when optimizing the collective communication algorithm for emergent application's workloads, whose performance is frequently dominated by the collective communication [5, 6]. For example, Dragonfly [7] and HyperX [8] network topology together with customized communication algorithms can challenge to outperform traditional Fat-Trees [5, 9]. Borrowing the theoretical foundations of graph theory to design interconnection networks is a promising approach. Slim Fly [10] is a representative system that uses theoretical results from graph theory. Slim Fly is built on the McKay–Miller–Šir'aň (MMS) graph, which is a well-known construction in graph theory.

Kautz graphs are well known in the study of degree–diameter problems [11, 12].

These graphs are attractive for network design because of their low diameter, high number of nodes per degree, and regular structure. To fully exploit the potential of the Kautz graph, we need to design a communication library that efficiently supports collective communication patterns on the underlying topology. In this dissertation, with the aim of optimizing resource utilization in HPC systems, we leverage the properties of the Kautz graph to build an efficient communication library for Kautz graph.

Kautz graphs share limitations with many theoretically attractive network topologies, such as Jellyfish [13], expander-based networks [14], and Xpander [15]. Although these topologies offer excellent theoretical properties, evaluating their performance at large scale is challenging due to the significant financial cost and engineering complexity involved in building such systems. This dissertation focuses on validating the underlying principle of optimizing resource utilization in collective communication algorithms.

In this dissertation, we introduce Kautz graphs as an interconnection network for HPC systems and develop a collective communication library that exploits their structural properties. To further validate the proposed ideas and avoid topology-specific conclusions, we apply the same resource-aware communication principles to the two-level fully connected (2dfc) topology, a well-known and widely used network topology in practical HPC systems.

1.2 Challenges and Motivation

The primary challenge addressed in this dissertation is the efficient utilization of available network resources. In high-radix network topologies, a large number of links are available at each node, but exploiting this parallelism efficiently remains difficult. At the same time, collective communication must maintain low end-to-end latency. Our goal is to improve end-to-end communication performance by designing collective communication operations that explicitly consider resource utilization. In particular, we focus on pre-scheduled communication schemes that enable a source node to concurrently leverage multiple outgoing links. This leads to the concept of “multi-port” collective communication operations, in which all available links of a node are utilized to increase communication parallelism and reduce overall latency.

1.2.1 Kautz Network Topology

Low latency in HPC networks strongly depends on low hop counts, yet many existing network topologies have hop counts much larger than the theoretical lower bounds, known as Moore' s bound. To alleviate this problem, network topologies based on Kautz graphs are attractive.

The Kautz graph is one of the best-known solutions to the degree–diameter problem [12]. In other words, given a uniform degree of switches and a fixed diameter, the Kautz graph is the largest graph that has been found. Given a fixed degree and diameter, the degree–diameter problem seeks to construct the largest possible graph. These constraints are defined by the degree and the diameter of the graph.

For a graph with degree d and diameter k, the number of vertices is upper bounded. This upper bound, known as the Moore bound, is given by $1 + d \sum_{i=0}^{k-1} (d-1)^i$ for bidirectional graphs and $1 + \sum_{i=0}^{k} d^i$ for unidirectional graphs. Kautz graphs achieve a high percentage of the Moore bound. For example, with degree $d = 8$ and diameter $k = 4$, a Kautz graph reaches 98% of the Moore bound with 4,608 vertices. This property makes Kautz graphs particularly attractive for designing low-diameter, high-radix interconnection networks for large-scale HPC systems.

Despite their attractive structural properties, Kautz graphs are directed networks. This unidirectional nature makes them difficult to directly adopt in HPC systems, where collective communication typically assumes bidirectional communication capabilities. As a result, Kautz graphs have not been widely used as interconnection networks in practical HPC environments. This motivates the need for novel collective communication algorithms that can effectively operate on directed network topologies.

1.2.2 Two-Dimensional Fully Connected (2dfc) Network Topology

Uni-directional Kautz network topology has attractive statistics of low diameter, low average shortest path length, and regularity. However, existing HPC systems have rarely adopted uni-directional interconnection networks because of their practical concerns, such as the lack of efficient switch-by-switch flow control. From a practical perspective, we therefore target a 2dfc network topology. In a 2dfc network topology,

the number of dimensions equals two. In each dimension, all nodes are fully connected, thus the network diameter is two. When the dimension count is two, HyperX and Generalized Hyper Cube (GHC) [16] variants become isomorphic to 2dfc under appropriate link configurations. The 2dfc structure therefore serves as a useful baseline in state-of-the-art HPC systems.

A 2dfc offers the following advantages for building a supercomputer inherited from its super-sets (Dragonfly and HyperX). Firstly, it boasts a low-latency attribute because of its diameter-two nature, ensuring low-latency data communication. Secondly, because each node has a direct connection to other nodes in all dimensions, a 2dfc provides high path diversity. This property aids in load balancing and fault tolerance by providing multiple data transmission routes [8, 6]. Finally, it shares an isomorphic relationship with existing topologies like Dragonfly and HyperX. This means that the research and development of collective communications efforts done in these network topologies can be leveraged, further enhancing the appeal of this network for state-of-the-art HPC systems.

In this dissertation, we first analyze classical collective communication algorithms commonly used on 2dfc. By examining their communication patterns and resource utilization, we identify inefficiencies in how available links are exploited. Based on this analysis, we propose an improved collective communication algorithm that explicitly leverages the multi-port capability of the 2dfc topology. This approach allows us to demonstrate how careful algorithm design can further improve performance, even on a well-studied and widely deployed network topology.

1.3 Contributions

This dissertation contributes not only new collective communication algorithms, but also a systematic design perspective that emphasizes maximizing network resource utilization through topology-aware, multi-port communication. Based on this perspective, we make the following contributions.

Proposal of multi-port message-combine collective communication for Kautz topology We design topology-aware, multi-port message-combine collective algorithms that fully exploit the multiple outgoing links of each node in diameter-two

Kautz networks. SimGrid evaluation results illustrate that the proposed algorithm achieves substantial performance gains over an existing competitor, being up to 11× faster in alltoall benchmarks.

Multi-port message-combine collective communication for 2dfc topology We adapt the proposed multi-port message-combine approach to the 2dfc network, which is widely used in state-of-the-art HPC systems such as HyperX and Dragonfly variants. The proposed multi-port message-combine operations were shown to improve throughput by up to 1.38x over a competitor at 32MB data size, confirming its practical applicability.

Comprehensive performance evaluation Using the SimGrid simulation framework (v3.28), we evaluate the proposed algorithms under micro-benchmarks (Allgather, Alltoall, Allreduce). The results demonstrate significant reductions in execution time and improved scalability compared with existing competitor algorithms.

1.4 Dissertation Organization

The remainder of the dissertation is organized as follows. Chapter 2 presents background information. Chapter 3 proposes multi-port message-combine collective communications on Kautz network topology, while Chapter 4 extends the multi-port message-combine collective communications to 2dfc network topology. Chapter 5 draws our conclusions with a summary of our findings.

Figure 1.1 illustrates the outline of this dissertation.

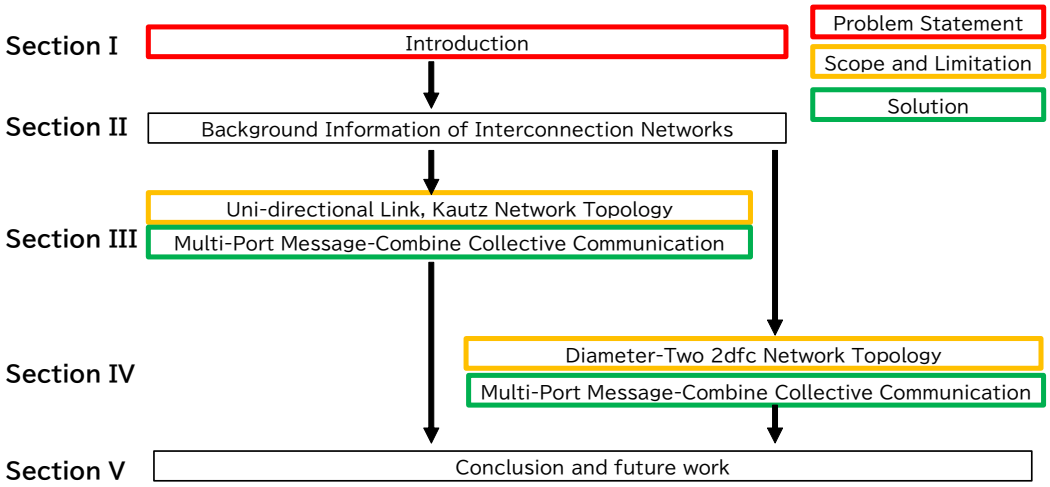


Figure 1.1: Outline of the Dissertation.

2

Background Information and Related Work

2.1 Interconnection Networks

An interconnection network is the infrastructure that connects computing elements within a system. It determines how data flows between devices and significantly impacts network performance, scalability, and reliability. There are four performance factors: network topology, routing, switching, and arbitration. Network topology refers to the static arrangement of channels and nodes in an interconnection network, as stated in a textbook[17]. The remaining network performance factors, routing, arbitration, and switching, rely heavily on the chosen network topology [18]. Routing provides a function to determine the allowable (valid) paths for packets. Arbitration provides a function to determine paths available for packets. Switching provides a function to determine paths allocated to packets.

In this subsection, we focus on network topology. We first introduce the concept

and classification of network topology. Then, we review related works on network topology for current HPC systems. Finally, we pick up two network topologies used in this dissertation: the Two-Dimensional Fully Connected (2dfc) network and the Kautz network.

2.1.1 Categories of Interconnection Networks

Many network topologies have been proposed in the literature. However, only a few of them have ever been implemented in supercomputers. A network topology connects a number of endpoints which can be a processor, a server, or a switch node. If every node plays both roles as an endpoint and forward point (switch), it is known as *direct network*. On the other hand, if at least a switch has no endpoints, it is called *indirect network*. In this dissertation, we consider only the direct network for the sake of simplicity.

A direct network can be expressed by a bi-directional or uni-directional graph, in which vertices and edges correspond to endpoints and network links, respectively. To understand the statics of a direct network, graph property is helpful. In this dissertation, two terms *graph* and *direct-network topology* are interchangeable.

Definition 2.1.1 (Graph)**.** We state graph G and G' as follows.

- A bi-directional graph $G(V, E)$ has a vertex set V and a bi-directional edge set E. Two vertices $u, v \in V$ are adjacent if there is an edge $e(u, v) \in E$.
- A uni-directional graph $G'(V', E')$ has vertices set V' and uni-directional edge set E'. Two vertices $u', v' \in V'$ are adjacent if there is an edge $e(u', v')$ or an edge $e(v', u') \in E'$.

The number of links connecting to/from a given vertex is the *degree* of that vertex, which can refer to the radix of a switch.

Definition 2.1.2 (Degree and Regular Graph)**.** We state the degree and regular graph G and G' as follows.

- Degree of a vertex u, $deg(u)$, is the number of neighbors of u. In the case of a unidirectional graph, $deg_{out}(u)$ is equal to the number of outgoing links from u, $deg_{in}(u)$ equals the number of incoming links to u.

- Degree of a graph G, $deg(G)$, is the maximum value of vertices' degree, $\max_{u \in V}(deg(u))$.
- G is a (bi-directional) regular graph if $deg(u)\ \forall u \in V$ is constant. If $deg_{out}(u') = deg_{in}(u') = constant\ \forall u' \in V'$, G' is a (uni-directional) regular graph.

The network topology was created with the aim to connect the nodes inside the network (or vertices inside the graph). Typically, there are more than one path $p_i(u,v)$ between a pair of nodes (u,v). The length of a shortest path between (u,v) is considered as the *distance* between them. In this dissertation, we consider regular graphs because existing supercomputers usually use regular or semi-regular graphs.

The following definitions are used as characteristics of a network topology.

Definition 2.1.3 (Path, Length, and Distance)**.** We state graph's path, length, and distance as follows.

- A *path* $p_i(u,v)$ between a pair of nodes (u,v) is a sequence of consecutive edges, $p_i(u,v) = e_1(u,v_1), e_2(v_1,v_2), ..., e_n(v_{n-1},v)$
- The *length* or *hop count* of a path $p(u,v)$ is the number of edges inside the path, $len(p_i(u,v)) = |p_i(u,v)|$.
- *Distance* between vertices u and v, dist(u,v), is the shortest joining u and v, $dist(u,v) = min_{i \in N}(len(p_i(u,v)))$.
- Diameter of a graph G is the maximum distance between any pair of vertices, $max_{u,v \in V}(dist(u,v))$.

Definition 2.1.4 (Cut, Bisection, Bisection Bandwidth)**.** We state Cut, Bisection, and Bisection Bandwidth as follows.

A *cut* of a network, $C(N_1,N_2)$, is a set of channels (links) that partitions the network into two disjoint sets, N_1 and N_2 [17]. The number of channels in the cut is $|C(N_1,N_2)|$, therefore the total bandwidth of the cut is

$$B(N_1,N_2) = \sum_{c \in C(N_1,N_2)} b_c$$

in which b_c is the bandwidth of each channel. A bisection of a network is a cut that partitions the entire network nearly in half, such that$|N_2| \leq |N_1| \leq |N_2| + 1$.

The channel bisection of a network, B_C, is the minimum channel count over all bisections of the network [17].

$$B_C = \min_{\text{bisections}} |C(N_1, N_2)|$$

The *bisection bandwidth* of a network, denoted by B_B, is the minimum bandwidth over all bisections of the network [17]:

$$B_B = \min_{\text{bisections}} B(N_1, N_2).$$

For networks with uniform channel bandwidth b, $B_B = bB_C$

In the next sub-section, we will present a survey of popular network topologies and their basic properties.

2.1.2 Existing Network Topologies

As mentioned in Section 1.1, network topology is a critical topic to research especially for SAN[1]. Research on LAN and WAN tends to be more ad hoc due to long distance and connecting across different communication subnets.

Many recent efforts [4, 20, 7, 8, 21, 22, 23, 24, 25, 26, 27, 28, 13, 29, 30, 15, 31] have been made to create an efficient network topology for data centers and HPC systems. Recent HPC systems frequently use fat-tree network topologies [4]. In the fat tree network topology, all the endpoints are connected at the bottom level of the trees. The number of links going down to its siblings is equal to the number of links to its parent in the upper level. Therefore, the link becomes "fatter" towards the top of the tree. The design of fat tree ensures all nodes in the network communicate with the same bandwidth and slightly different latency (for nodes at different distances). It was reported that fat tree was the cost burden for a big system with up to 33% of the system cost [20].

To mitigate the cost problem, some network topologies [20, 7, 8] have been proposed thanks to the emergence of high radix switch. However low radix networks, such as

[1]The network domain is classified follow *appendix F* of the textbook [19]. There are four categories. OCN: On-Chip Network. SAN: System/storage Area Network, LAN: Local Area Network, WAN: Wide Area Network. The classification depends on the distance and the number of devices in the network.

k-ary n-cubes, were unable to take the advantage of high radix technology. Thus, there are some recent studies on high-radix network topologies.. There are some researches to reduce the hardware cost while maintaining the performance of fat tree network. *Flattened Butterfly* [20] was proposed based on an observation that fat tree has a building cost nearly double butterfly network while butterfly has no path diversity. Thanks to high-radix switch, *Flattened Butterfly* can achieve low hop count and construction cost compared to fat tree. In addition, it can achieve higher path diversity, and improves the performance of adversarial traffic, when compared to conventional butterfly.

More recently, optical technology has had breakthrough improvements leading to the development of network equipment. A number of topologies based on photonics equipment have been born [21, 22, 23, 24, 25, 26, 27]. Optical technologies, such as Reconfigurable Optical Add-drop Multiplexers (ROADMs), can add or drop wavelength between data channels without optical-to-electrical conversion. This provides an opportunity to flexibly adjust bandwidth for different data sources. In other words, bandwidth for each data channel and the route between source and destination can be reconfigured.

Proteus [21] and OSA [22] use Micro-Electro-Mechanical Switch (MEMS) to reconfigure (bandwidth and topology) between Top-of-Rack switches (ToR). MEMS adjusts the angle of micromirrors to alter the connection relationship. Flexfly [23] can reconfigure either bandwidth or topology based on Silicon Photonic (SiPh). Bandwidth in Flexfly is redirected from idle links to the links with larger traffic. This method can provide more transfer capacity to under-utilized links and increase the communication bandwidth. Flexspander [24], 3D-Hyper-FleX-LION [25], Opera [26], Sirius [27] are some recent works about the network topology that are based on the advantage of technology, which is in conjunction with a network topology from graph-theory field. Flexspander applies xpander [15] network in the context of the new reconfigurable switch. 3D-Hyper-FleX-LION proposes a topology for multi-dimension connectivity for ToR switches. Opera extends Rotornet [32] and research on the context of expander graph rotation [14].

More recently, OCSes are used in real systems, such as Google TPUv4 supercomputers and datacenters. The former uses OCSes as improving job embeddability, while the latter uses OCSes as high expandability and low communication latency. Both of

them take a single interconnection network, which use OCSes and EPSes.

Optical technologies with reconfigurable functionality hold great promise for the future of HPC networks. However, at present, they suffer from high reconfiguration latency and significant costs, making them unsuitable for building supercomputers [33].

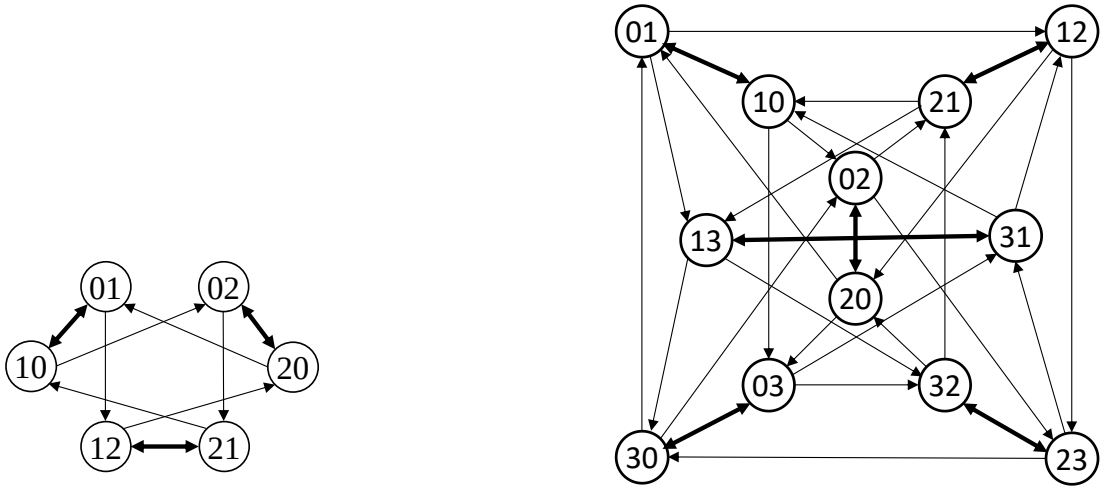
Besides technology, graph theory evolves independently and gains interesting achievements. Scientists in computer science also take this achievement to network topology [28, 13, 29, 10, 30, 15, 31]. Koibuchi et.al. [28] first introduced random topology to the HPC network. Singla et.al [13] introduced random topology to data center network. The advantage of random topology is that they have a larger number of nodes, and smaller average shortest path length when using the same amount of hardware as non-random topologies. The studies also show that random topology is easy to extend. Following Jellyfish of Singla et.al., Long hop networks [29] and Slimfly [10] were proposed based on expander graphs, despite that they use different build methods. Long hop networks are built based on Cayley graphs [34] while Slim Fly is based on a group of graphs called McKay–Miller–Širáň (MMS) [35]. 2015, Valadarsky et.al. found the commonality of a good random graph and called it "expander" [15].

Random topologies have theoretical advantages. However, they are not used in current supercomputers [1] because their randomness makes implementation difficult. Slimfly [10] is a topology based on MMS graphs from graph theory. It has been tested at scale. The results show that it performs better than non-blocking fat-tree networks for deep neural network (DNN) workloads. It also reduces costs [36].

2.1.3 Target Network Topology: Kautz

In this dissertation, we focus on yet another network topology: uni-directional Kautz graphs [37] and 2dfc.

Kautz graph (see Figure 2.1) was invented by W.H. Kautz in [37, 38]. Kautz graph is characterized by degree d and diameter k, denoted $K(d,k)$. Vertices of $Kautz(d,k)$ are labeled with words of length k $(x_1, x_2, ..., x_k)$ $where$ $x_i \in [0..d]$. Two continuous letters in $(x_1, x_2, ..., x_k)$ cannot be identical $x_i \neq x_{i+1} \forall i \in [1..d-1]$. There is an (unidirectional) edge from a vertex x to vertex y iff the last $k-1$ letters of x are identical to the first $k-1$ letters of y. Some properties of $K(d,k)$, for a Kautz graph degree d, diameter k:



(a) Kautz(2,2) **(b)** Kautz(3,2)

Figure 2.1: Examples of Kautz graph.

- There are $(d+1) \times d^{k-1}$ vertices.
- There are d disjoints paths from any node x to any other node y (prove can be found in [39]).
- Because one node connects to d neighbors, there are $d \times ((d+1) \times d^{k-1}) = (d+1) \times d^k$ edges.

In the field of graph theory, scientists try to create a graph as large as possible - degree/diameter problem. The problem can be stated is that: find the largest graph $G(V,E)$ $(max(|E|))$ that $\forall v \in V, deg(v) \leq d$ and $diam(G) \leq k$, in which d and k are given degree and diameter respectively. The upper bound for this problem was first mentioned in [40]. The bound is as follows. For a given graph $G(V,E)$ has degree d and diameter k

$$|E| = \begin{cases} 2k+1 & if\ d = 2 \\ 1 + d\frac{(d-1)^k - 1}{d-2} & if\ d > 2 \end{cases} \tag{2.1}$$

In the case of a unidirectional graph:

$$|E| = \begin{cases} k+1 & if\ d = 1 \\ \frac{d^{k+1}-1}{d-1} & if\ d > 1 \end{cases} \tag{2.2}$$

Kautz graph is famous for the fact that it is the best-known solution for degree/diameter problem. When k and d are large enough Kautz can achieve over 99% of Moore bound. For example, in the case of diameter 2, the ratio between the number of vertices of Kautz and Moore bound is as follows:

$$\frac{|V_{Kautz}|}{Moore\ Bound} = \frac{d(d+1)}{\frac{d^{2+1}-1}{d-1}} = \frac{d(d+1)(d-1)}{d^3-1} = \frac{d^2+d}{d^2+d+1} \tag{2.3}$$

The difference between theoretical bound and the Kautz is just $(d^2+d+1)-(d^2+d) = 1$ vertex. In this dissertation, we target to diameter two Kautz graphs. We use the following notations and definitions:

Definition 2.1.5 (Path, in_neighbor, out_neighbor)**.** Let $K(d, 2)$ be a Kautz graph with degree d and diameter two. Choose two arbitrary nodes from K. They are source and destination nodes, denoted $a = \overline{xy}$ and $b = \overline{zt}$ respectively.

- A path $P(a, b)$ is a sequence $E = \langle e_1, .., e_n \rangle$ of letters, where $e_i \in [0..d]\ \forall i \in [0..n], e_i \neq e_{i+1}\ \forall i \in \{0..n\}$. First two letters of E are equal to source $\overline{xy}$. Last 2 letters of E are equal to destination $\overline{zt}$. Each pair of consecutive letters from the second letter in E is an intermediate node from the source a to the destination b. For example, the shortest path between $\overline{10}$ and $\overline{32}$ in Figure 2.1b is $P(\overline{10}, \overline{32}) = \langle 1032 \rangle$ i.e. $\overline{10} \rightarrow \overline{03} \rightarrow \overline{32}$. One other alternative path having length 3 is $P(\overline{10}, \overline{32}) = \langle 01232 \rangle$ or $\overline{10} \rightarrow \overline{12} \rightarrow \overline{23} \rightarrow \overline{32}$

- The intermediate node whose pair of letters starts from the second letter of $P(a, b)$ is called $out\_neighbor\ on_P$ of source a. Similarly, The node whose pair of letters ends before the last letter of $P(a, b)$ is called $in\_neighbor\ in_P$ of b.

In addition to the shorter diameter, Kautz graph has some unique interesting properties such as:

- **High connectivity and Fault tolerance**: because a node $a = \overline{xy}$, in a degree d diameter 2 Kautz graph, guarantees d disjoint paths from/to a.

- **Regular and Symmetric Structure**: the symmetry of Kautz graph simplifies the design of routing algorithms and communication protocols, because each node holds uniform connectivity characteristics. We can easily prove the symmetry of

Kautz graph by changing the notation of each vertex (e.g., swap two symbols). After the changing, the structure of the graph remains unchanged.

- **Deterministic Construction**: Kautz graphs are derived from numerical definition in graph theory. Therefore, it can be constructed deterministically. The deterministic structure enables for precise performance analysis and implementation. This is beneficial for designing scalable and efficient HPC interconnection networks.

The three properties make Kautz graph a promising topology for HPC interconnection networks but it is not widely used in network topology. One reason we can explain is that there is no efficient communication algorithm for it. In this dissertation, we will present a contention-less collective communication on Kautz graph diameter two.

2.1.4 Target Network Topology: Two-Dimensional Fully Connected (2dfc)

A **Two-Dimensional Fully Connected (2dfc) Network Topology** is expressed by a bidirectional graph, consisting of N nodes, represented as a Cartesian product:

$$N = m_2 \times m_1, \tag{2.4}$$

where m_2 and m_1 represent the sizes along the Y-axis and X-axis, respectively.

Each node is addressed by a 2-tuple:

$$A = (a_2 a_1), \quad \text{where } 0 \leq a_i < m_i, \forall i \in \{1, 2\}. \tag{2.5}$$

A node $A = (a_2 a_1)$ is connected to all other nodes along its two coordinate axes in a 2dfc network topology. That is, node A connects to all nodes $A' = (a'_2 a'_1)$ satisfying

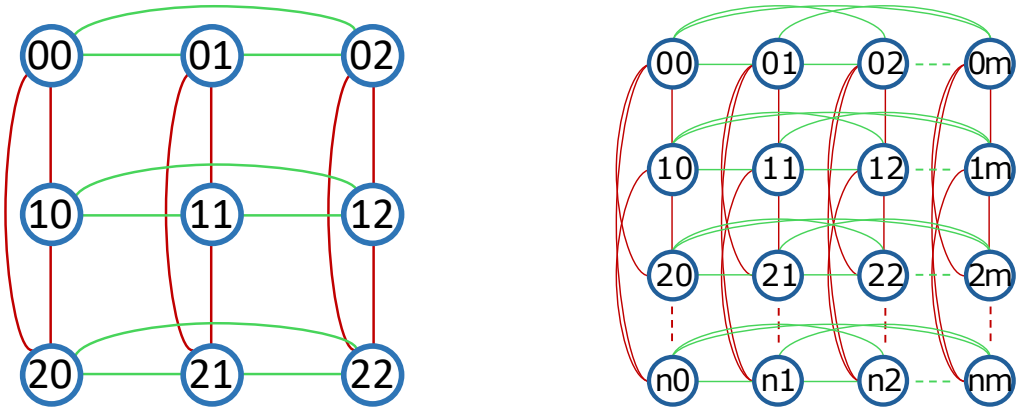
$$a'_i \in \{0, 1, \ldots, m_i - 1\} \setminus \{a_i\}, \quad \forall i \in \{1, 2\}. \tag{2.6}$$

Totally, each node has $(m_1 - 1) + (m_2 - 1)$ neighboring nodes and the edge set is

defined as:

$$E = \{(A, A') \mid A, A' \in V, \exists i \in \{1, 2\} \text{ such that } a'_i \neq a_i \text{ and } a'_j = a_j \text{ for } j \neq i\}. \quad (2.7)$$

Figure 2.2 illustrates both a 3×3 and an $n \times m$ 2dfc network. In the topology, each node is directly connected to all other nodes along the same coordinate axis. For instance, in Figure 2.2a, node (00) is directly connected to nodes (01) and (02) along the **X-axis**. Similarly, node (00) has direct connections to nodes (10) and (20) along the **Y-axis**.



(a) 3×3 2fdc **(b)** $m \times n$ 2dfc

Figure 2.2: Two examples of 2dfc.

Each node is connected to all other nodes along its coordinate axes:

$$d(A) = (m_1 - 1) + (m_2 - 1) = m_1 + m_2 - 2. \quad (2.8)$$

Each node contributes $m_1 + m_2 - 2$ edges, but each edge is counted twice (once for each endpoint). Thus, the total number of unique edges in the network is:

$$|E| = \frac{N(m_1 + m_2 - 2)}{2} = \frac{m_1 m_2 (m_1 + m_2 - 2)}{2}. \quad (2.9)$$

Since each node can reach directly any other node in the same row or column, the **diameter** of the network is always two.

Table 2.1: Summary of properties for (3×3) and ($m \times n$) 2dfc network topologies.

	3×3 **2dfc**	$m \times n$ **2dfc**
Number of Nodes	9	$N = m \times n$
Node Degree	4	$d(A) = (m-1) + (n-1)$
Total Edges	18	$\|E\| = \frac{mn(m+n-2)}{2}$
Network Diameter	2	2

Table 2.1 lists the properties and characteristics of 3×3 2dfc and $m \times n$ 2dfc.

Bisection bandwidth of 2dfc

Consider a "line" of switches along dimension m, in this "line" there are m_i switches. Divide the switches on this "line" into two groups, each group will have $m_i \div 2$ switches (assuming that m_i is even, bandwidth of each link is equal and equal to 1). The number of links that cross this channel bisection within this single "line" is

$$\frac{m_i}{2} \times \frac{m_i}{2} = \frac{m_i^2}{4} \tag{2.10}$$

Entire 2dfc network can be thought as $\frac{m_2 \times m_1}{m_i}$ such "line" of switches, each fully connected along dimension m. The total channel bisection B_C for dimension m is

$$B_{Cm} = \frac{m_2 \times m_1}{m_i} \times \frac{m_i^2}{4} = \frac{m_2 \times m_1 \times m_i}{4} \tag{2.11}$$

Therefore the channel bisection bandwidth of the network is the min B_{Cm}, equal to:

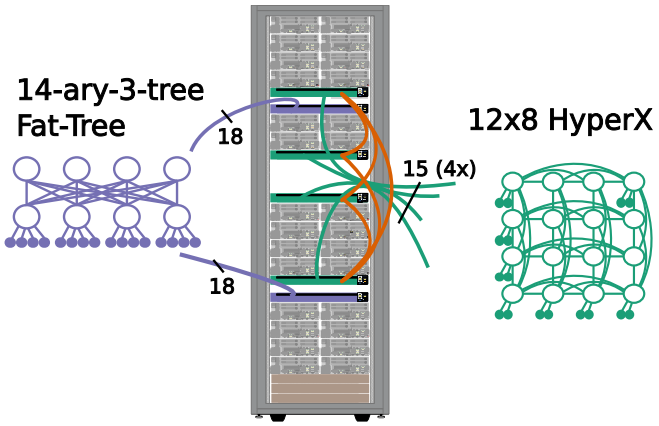
$$B_C = \frac{m_2 \times m_1 \times \min(m_2, m_1)}{4} \tag{2.12}$$

Consider a 2dfc with T terminal per node, a 2dfc network needs at least half the number of terminal links ($T \times m_2 \times m_1/2$) crossing any bisection B_{Cm} to be non-blocking. Therefore, the bisection bandwidth of 2dfc is

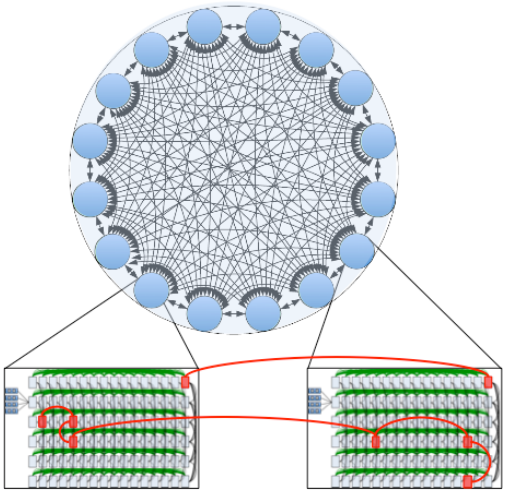
$$B_B = \frac{B_C}{T \times m_2 \times m_1/2} = \frac{\min(m_2, m_1)}{2T} \tag{2.13}$$

Comparison with Similar Direct Network Topologies

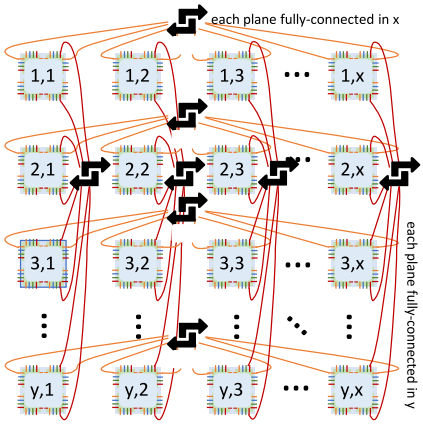
The 2dfc can be regarded as a family of HyperX, Dragonfly, and Hammingmesh network topologies. Here, we describe their difference in detail.



(a) HyperX network in a dual planes system (Figure 2c from [9]).



(b) Cray Aries network (page 7 From [41]).



(c) Hammingmesh network (Figure 3 from [42]).

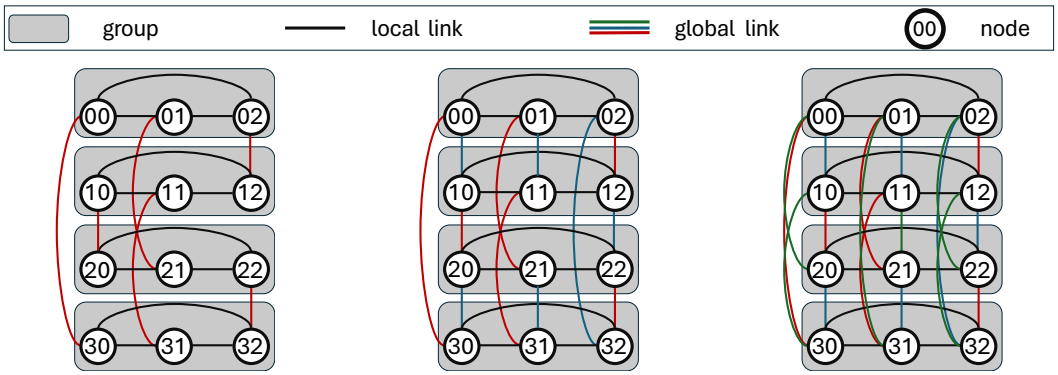
Figure 2.3: The widespread presence of 2dfc in HPC Networks.

Figure 2.3 shows the 2dfc topology used in HPC systems. Many interconnection applications in HPC systems use this structure.

It can serve as the primary interconnection topology for an entire system [8, 9, 6]. For instance, Figure 2.3a depicts the network topology of a supercomputer at the Tokyo Institute of Technology, which utilizes a 12×8 HyperX topology, which is isomorphic to a 2dfc network. HyperX considers a floor-layout and a rack-layout with high-radix network switches. By contrast, 2dfc does not care about the floor layout. This is a main difference between HyperX and 2dfc network topologies.

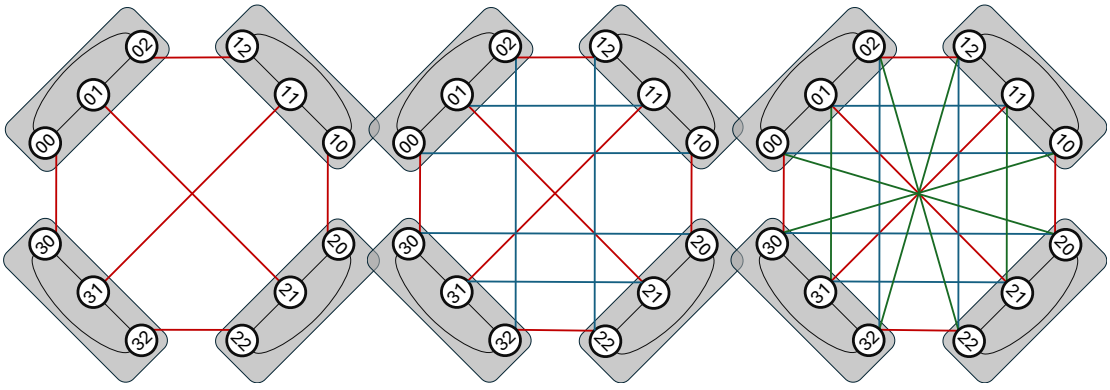
Alternatively, 2dfc can be used to connect local node groups, as seen in the Cray Aries interconnect [43]. This interconnect is deployed in the Piz Daint [44] and Trinity [45] supercomputers. Both Piz Daint and Trinity utilize a 16×6 2dfc topology for local interconnects.

Furthermore, 2dfc also serves as a global topology for linking multiple node groups, as demonstrated in HammingMesh [42]. In this case, HammingMesh employs a 2dfc structure to interconnect $n \times n$ mesh networks. HammingMesh considers a two-dimensional floor-layout for a machine-learning parallel computer. By contrast, 2dfc does not care about the floor layout. This is a main difference between HammingMesh and 2dfc.



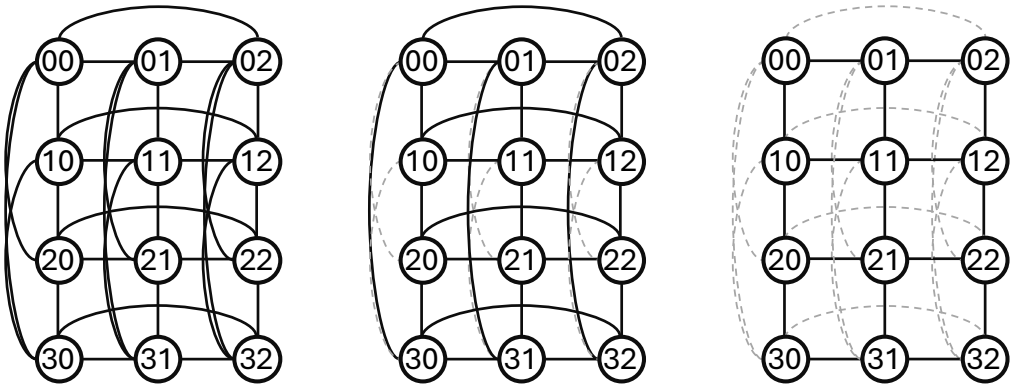
(a) Dragonfly $a = 3, g = 4, h = 1$ **(b)** Dragonfly $a = 3, g = 4, h = 2$ **(c)** Dragonfly $a = 3, g = 4, h = 3$

Figure 2.4: Dragonfly Network with Varying Global Link Configurations ($a = 3, g = 4, h = \{1..3\}$). Nodes are organized in a two-dimensional layout.



(a) Dragonfly $a = 3, g = 4, h = 1$ **(b)** Dragonfly $a = 3, g = 4, h = 2$ **(c)** Dragonfly $a = 3, g = 4, h = 3$

Figure 2.5: Dragonfly with different global link configurations. ($a = 3, g = 4, h = \{1, 2, 3\}$)



(a) 3 × 4 2dfc **(b)** 3 × 4 Torus **(c)** 3 × 4 Mesh

Figure 2.6: The 3 × 4 2dfc, Torus and Mesh network. By selective removing some certain links of 2dfc, we can obtain Torus and Mesh.

Compared with the other topologies, the 2dfc network maintains a similar fundamental structure. By selectively removing specific links from a 2dfc, different HPC topologies can be derived, such as Dragonfly, Torus, and Mesh networks. Figures 2.4 and 2.5 show various global link configurations of a 3 × 4 Dragonfly network. Figure 2.4(a)-(c) are isomorphic to Figure 2.5(a)-(c), respectively. The Dragonfly network with parameters $a = 3, g = 4, h = 3$ (shown in Figure 2.5c) is isomorphic to the 3 × 4 2dfc network. Additionally, by removing more links from a 2dfc, we can obtain Torus and Mesh topologies, as shown in Figure 2.6.

2.2 Collective Communication

2.2.1 Outline

After selecting the network topology, the next critical component in optimizing communication performance is the *communication algorithm*. It determines how messages are traveled across the network. Currently, *Message Passing* is the widely used programming model to exchange messages inside distributed systems. This can be categorized into two types:

- Point-to-Point (P2P) communication, where messages are exchanged between exactly two processes.

- Collective communication, where communication involves a group of processes simultaneously.

P2P communication serves as the fundamental building block, with efforts focused on achieving performance close to the raw bandwidth and latency of the hardware. Collective communication, in contrast, involves all processes within a communicator. Typical operations include broadcast, reduce, allreduce, gather, scatter, and barrier, each designed for a specific communication pattern. For instance, broadcast distributes data from one process to all others, while allreduce combines values from all processes and then shares the result with every process in the communicator.

Due to their global scope, the performance of collective operations is influenced by multiple factors such as network conditions, network topology, and the collective-communication algorithm. In large-scale HPC systems, the execution time of collective operations often dominates overall application performance.

Two factors mainly influencing collective performance are latency (time to send zero byte message) and the bandwidth [46].

To model the performance of collective communication, the well-known α–β **model** (or *Hockney model*) is commonly employed [47]. The total communication time is estimated as:

$$T = \alpha + \beta \cdot m$$

where α represents the fixed latency overhead (e.g., connection setup, synchronization), β is the inverse of the available bandwidth, and m is the message size in bytes. Communication algorithms are typically designed to minimize T by reducing the number of communication rounds (to minimize latency) and optimizing data paths (to maximize bandwidth usage).

As the increasing complexity of network topologies in modern supercomputers, achieving low latency and high bandwidth in collective communication becomes more challenging. A naive algorithm that ignores the topology can result in significant performance degradation due to traffic contention and unbalanced communication schedules. In terms of using topology information, we can divide the communication algorithms into two categories:

- **Topology-unaware**, which assume uniform communication cost between any pair of nodes, and are thus portable but suboptimal on modern systems.

- **Topology-aware**, which co-design the communication schedule and the topology, minimizing long-distance traffic and improving locality.

In this section, we provide a comprehensive review of communication algorithms for collective operations in HPC systems, with a focus on those designed for or adapted to Kautz graph or 2dfc.

2.2.2 Collective Communication Pattern

In this section, we will introduce the purpose, the communication pattern of common used collective communications. The **Message Passing Interface (MPI)** [48, 49, 50] is the most widely use standardized system [51] that supports the Message Passing paradigm. It enables communication between processes running in parallel across distributed memory systems. This dissertation borrows the terms of collective communication operations in the MPI document [50] to call collective operations.

Collective communication refers to communication operations that involve a group of processes working together. This group is organized into a structure called a *communicator*. All processes within a communicator must call the same collective operation simultaneously.

In this section, we revisit seven important collective operations commonly used in HPC applications. These operations can be grouped into two main categories:

- **One-to-All**: A designated root process either sends data to or receives data from all other processes in the communicator.
- **All-to-All**: All processes in the communicator exchange data symmetrically with each other, and all have equal roles.

The following presents the communication patterns of these seven collective operations.

Broadcast

One process sends the same data to all processes in the same communicator, as shown in Figure 2.7a.

Scatter

One process sends different data to all other processes in the communicator, as shown in Figure 2.7b.

Gather

All other processes in the communicator send different data to one process. as shown in Figure 2.7b.

Reduce

It combines data from all processes in a communicator using a reduction operation (e.g., sum, max) and returns the result to a root process, as shown in Figure 2.7c.

Allgather

It is the same as Gather except that all processes in the communicator receive the result, as shown in Figure 2.7d.

Alltoall

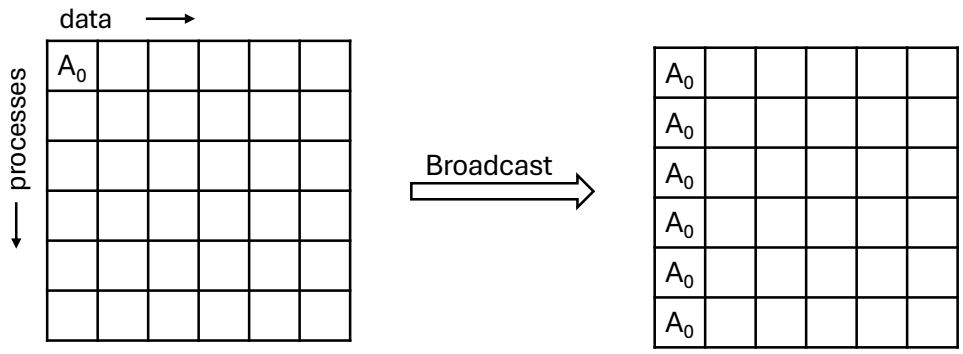
Scatter/Gather data from all processes/to all processes of a communicator. Each process have a distinguish data to send to other process, as shown in Figure 2.7e.

Reduce-Scatter

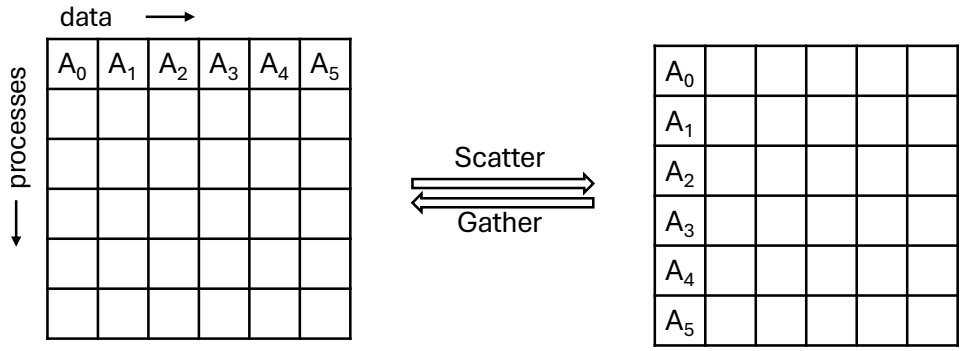
It is a combination operation of reduction and scatter, as shown in Figure 2.7f.

Allreduce

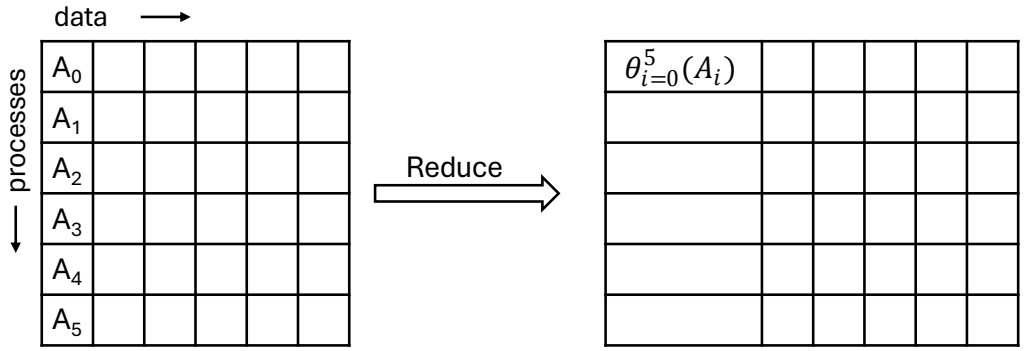
It is a series of a reduction operation (such as sum, max, min or user-defined functions) where the result is returned to all processes of the communicator, as shown in Figure 2.7g. We can obtain the result of Allreduce operation by doing the Reduce-Scatter follow by an Allgather (same as Rabenseifner algorithm [52]).



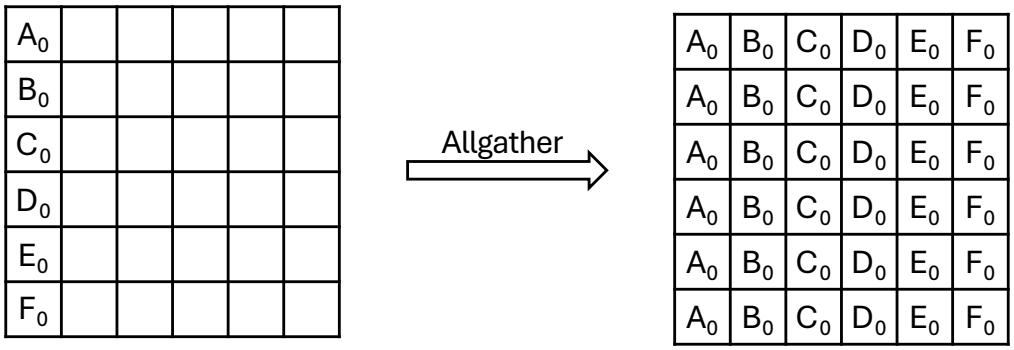
(a) Broadcast. Data A_0 is originated from one process and broadcasted to all 6 processes.



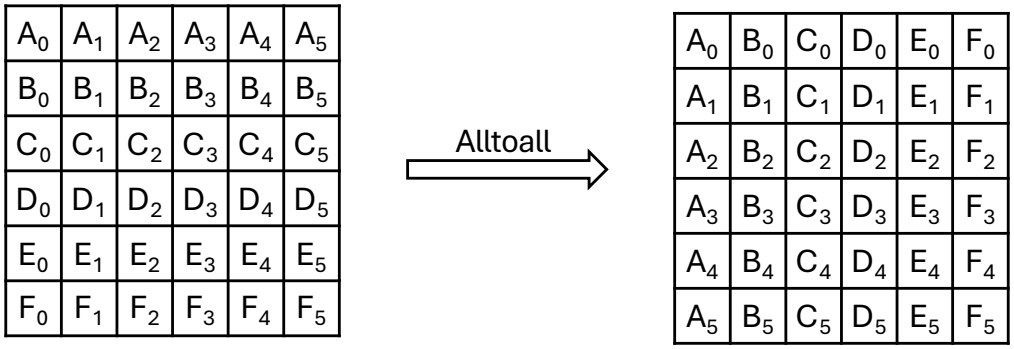
(b) Scatter and Gather. From the left to the right: scatter data from one process to 6 processes. From the right to the left gather data from six processes to one process.



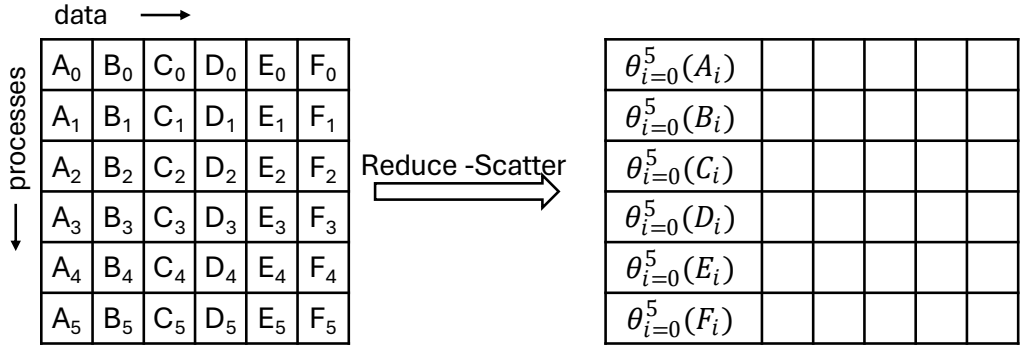
(c) Reduce: Data is reduced from six processes to one process. θ shows the reduce operator.



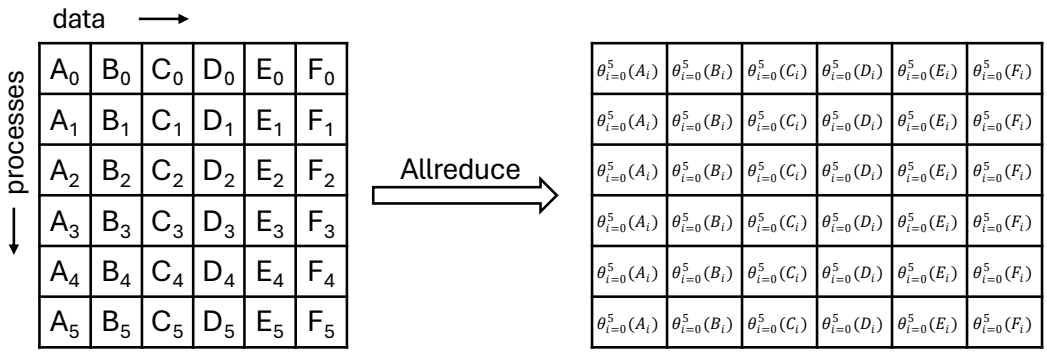
(d) Allgather: All processes in the communicator receive the data. Process i-th sends data to the destination process and places the data at position i-th in the receive buffer.



(e) Alltoall: Each process sends a distinct data to all receiver in the communicator. i-th block in the process j is sent to process i and placed in the j block of the receive buffer.



(f) Reduce-Scatter: Performs a reduction of data across all processes, then distributes segments of the reduced result to each process.



(g) Allreduce: Combines data from all processes using a reduction operation (e.g., sum, max) and then distributes the final result to all processes. Each process receives the same complete reduced result.

Figure 2.7: Communication pattern for common collective operations. Subfigures show examples for six processes.

2.2.3 Collective Communication Algorithms

Collective communication algorithms define how all processes in a communicator coordinate to execute collective operations using the resources of the network topology. The goal of these algorithms is to minimize latency [53, 54, 5, 55], maximize global bandwidth [53, 56, 57, 58, 59], and avoid contention. In essence, all these efforts aim to minimize the total communication time.

Early works on collective communication centered on optimizing algorithms for specific architectures such as hypercube [60], mesh [60], or fat-tree [4] to reduce the network contentions [61, 62, 63]. State-of-the-art communication libraries, e.g., MPI MPICH [64], and MVAPICH [65] select the best-suited algorithms for each collective operation depending on the message size and the number of involved computing nodes [66]. For example, the recursive doubling [67], binomial tree [64], and 2Tree [68] algorithms are optimized for small messages with a power-of-two number of nodes. The Bruck algorithm [69] is optimized for a non-power-of-two number of nodes n. It has been reported that these algorithms reach the lower bound of the latency factor, i.e., $O(\log(n))$ communication steps [67]. In contrast, the vector halving and distance doubling algorithm [64] and the ring-based algorithm [70] are optimized for large messages and reach the lower bound of the bandwidth factor, i.e., $O(\frac{n-1}{n}M)$ [67] units of data. Most of these works assume that the node's network interface is single-ported, i.e., one node sends and receives only one message simultaneously.

In recent years, there have been extensive efforts to improve the performance of collective communication by leveraging the underlying emerging hierarchical heterogeneous architectures that could be modeled as multi-port systems. When each computing node has $d > 1$ ports (degree-d), it can send and receive at most d messages in every communication step. Several collective communication techniques have been reported to achieve better resource utilization [71, 72, 73] by taking into account the hierarchy of the network architecture. On the other hand, there is an alternative that utilizes the custom network architecture to minimize the network contention [74, 5, 75]. The MULTITREE algorithm [75], for example, can deliver contention-free communication. By scheduling communication patterns based on the available network resources, the algorithm conducts communication only over links that are not yet occupied. The MULTITREE algorithm exploits the topology information and makes the schedule for collective communication.

In the later part of this section, we will present the operation of five collective communication algorithms that work on HPC systems. They can be used in the production libraries as in the case of Pairwise, Bruck and Ring algorithm. Another one is a state-of-the-art algorithm to design bandwidth-optimal communication as in the case of MULTITREE algorithm.

Pairwise-Exchange Algorithm

The Pairwise-Exchange (or pairwise for short) algorithm is one of the early techniques used to implement efficient collective communication operations. It is particularly well-suited for alltoall, allgather, and allreduce operations. Because of its simple logic, symmetric structure, and ability to fully utilize point-to-point communication, the pairwise algorithm is especially attractive for systems with uniform topologies (e.g., mesh, torus, fat-tree). Early studies of the algorithm can be found in the works of Gropp et al. [64, 49].

Pairwise remains a key algorithm in production MPI implementations including MPICH [65], OpenMPI [76] and Intel MPI [77]. It is typically selected when:

- The message size is large.
- The number of processes is a power of two, enabling XOR partner logic.

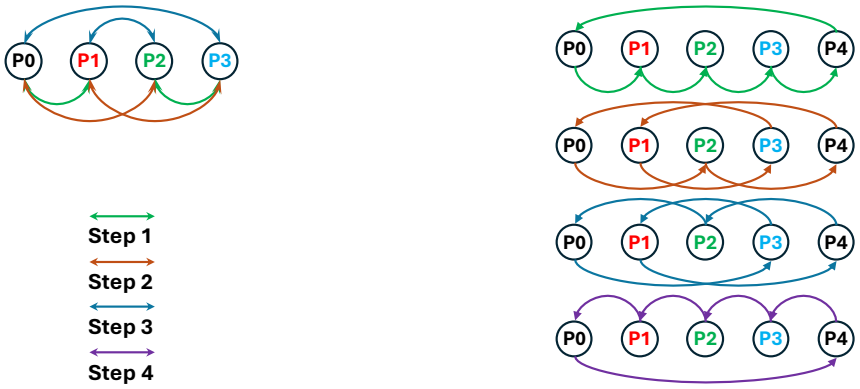
- Hardware supports multi-ported communication for overlapping send/receive.

In a communicator of P processes, the algorithm finishes in $P - 1$ steps. Each process communicates directly with a unique partner in every step. For power-of-two case ($P = 2^k$) partners are computed using the bitwise XOR. In step k:

$$partner = rank \oplus k$$

For non-power-of-two case ($P \neq 2^k$)m, a modular scheme is used. In step k:

$$\begin{aligned} sendto &= (rank + k) \mod P \\ recvfrom &= (rank - k + P) \mod P \end{aligned}$$



(a) Power-of-Two Case (XOR Pattern) **(b)** Non-Power-of-Two Case (Modular Shift)

Figure 2.8: Pairwise algorithm.

Figure 2.8 shows the communication pattern of pairwise exchange in two cases: four processes (power-of-two) 2.8a and five processes (non-power-of-two) 2.8b.

Despite the simplicity of pairwise algorithm, it is an index-based algorithm. This means that the algorithm ignores the topology information that can cause a collision for a specific topology. Since it only uses rank indices, it does not optimize message routing based on how nodes are physically or logically connected. If many processes try to send messages over the same network links at the same time, network congestion or link contention can occur.

Bruck Algorithm

Bruck's algorithm was originally proposed for alltoall (index) and allgather (concatenation) [78, 69]. It was designed to minimize communication steps. It has since become a standard method in many MPI implementations. Libraries that include Bruck or Bruck-like implementations: MPICH [49], OpenMPI [76], Intel MPI [77]. Bruck algorithm implements an alltoall or allgather in $\log_2 P$ communication steps.

Bruck algorithm proceeds in two local rotations and $\log_2 P$ communication in a P-process communicator. Steps of the algorithm:

1. **Local Rotation**: Each process logically rotates its local data buffer to align destination offsets.

2. **Logarithmic Communication**: In step $k = 0$ to $\log_2 P - 1$, each process i sends data to process $(i + 2^k) \mod P$ and receives data from process $(i - 2^k + P) \mod P$

3. **Final Rotation**: rotate back to restore the correct order.

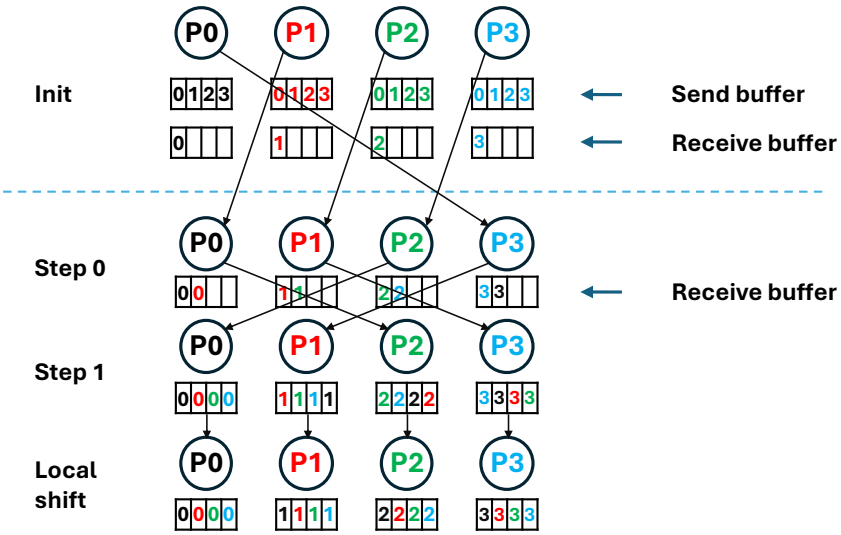


Figure 2.9: Bruck algorithm.

Figure 2.9 shows the communication pattern of the Bruck algorithm[2] with four processes. Each process starts with its own data, and the goal is for every process

[2]This communication pattern can be applicable for alltoall or allgather.

to gather data from all other processes. The algorithm completes in $\log_2 4 = 2$ communication steps.

It begins with a local rotation (omitted in in the figure) during which each process copies its local data to the result buffer. Next, two communication rounds are performed, where the sender and receiver are determined based on the current step, following the principle of **logarithmic communication**. Finally, the algorithm finishes with a local shift.

The Bruck algorithm does not take network topology into account. Thus It faces the same issues as the pairwise algorithm. When many processes attempt to send messages over shared network links concurrently, it can lead to congestion or link contention. However, the algorithm completes in $\log P$ communication steps, which is particularly efficient for small message exchanges. Because, the small message size introduces little contention, and the overhead from the local rotation is also small.

Ring Allreduce Algorithm

Ring Allreduce algorithm was proposed to optimize the bandwidth utilization in HPC systems by minimizing redundant data movement and avoiding network congestion. The ring algorithm is bandwidth-optimal for large messages, making it ideal for applications where data size per process is large and uniform, such as deep learning. It became especially well-known through its implementation in NVIDIA's NCCL [79] (used in GPU collectives), and has also been adopted by OpenMPI [76], Intel MPI [77], and Horovod [80] for distributed training.

Let:

- P be the number of processes in the communicator.
- Every process has a local buffer size N. The process partition the buffer into P smaller chunks.Each chunk has size N/P
- The goal of the algorithm is combining the data of all processes into a global result. Then distribute the result to all processes.

The Ring Allreduce operates in two phases reduce-scatter and allgather as Rabenseifner algorithm [52]. **Phase 1: Reduce-Scatter**: each process send one chunk to the

next process, and receives one chunk from the previous process. Reduction operation (e.g., sum, max, etc.) is performed between the received chunk and the local one for that segment. The reduction is repeated $P-1$ times. After this phase, each process holds one chunk of the final result. **Phase 2: Allgather**: each process sends its reduced chunk to the next process and receives a different chunk from the previous process. This continues for $P-1$ steps. After this phase, all processes have gathered all the reduced chunks — completing the allreduce.

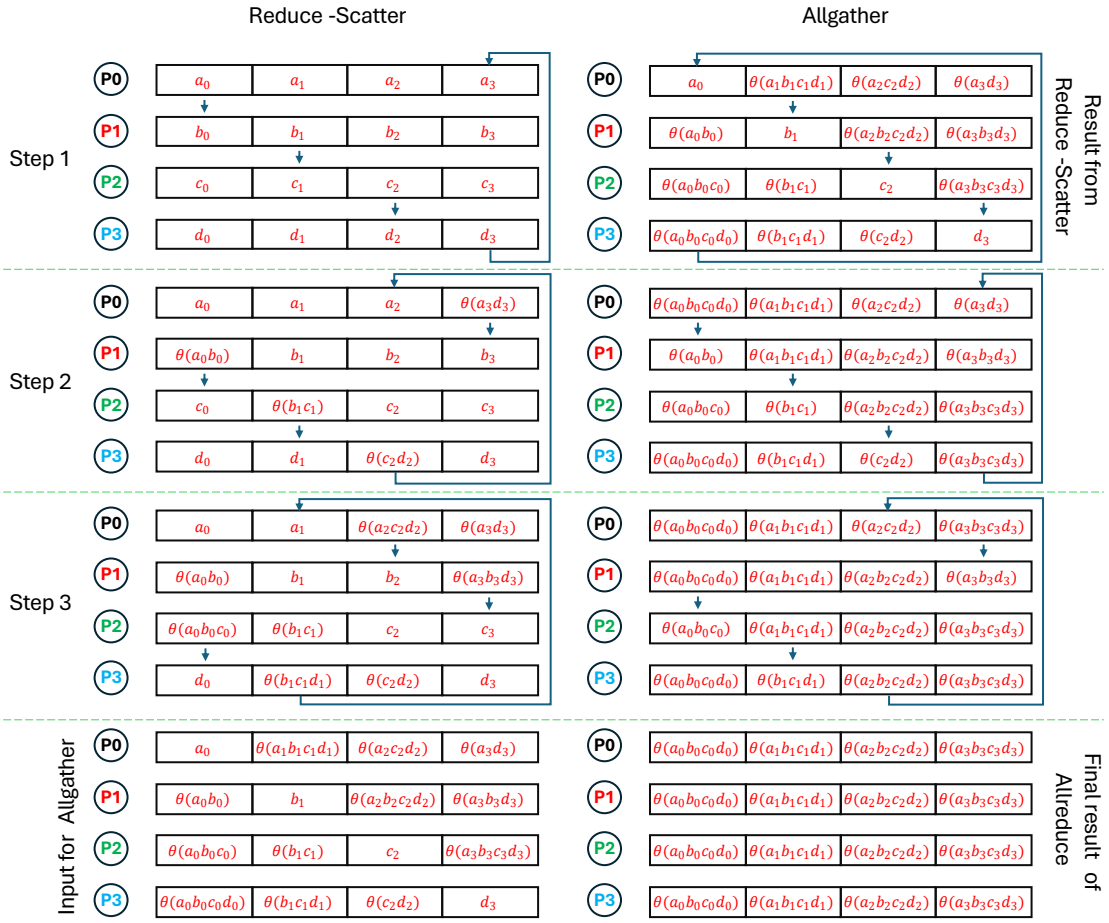


Figure 2.10: Ring Allreduce algorithm.

Figure 2.10 shows an example or ring allreduce in a four-node setup. Because there are four processes in the communicator, each phase of the algorithm has $4-1=3$ steps.

In summary, the Ring Allreduce algorithm finishes in $2\times(P-1)$ steps. This high number of steps is not good for latency-sensitive applications. Ring allreduce is proved

bandwidth optimal [70, 81, 82](see Section 2.4). Bandwidth optimal algorithms are especially for applications exchanging large message size. In addition to the limitation of high number of step, the performance of the Ring Allreduce algorithm depends on finding a contention-free ring [70]. The method for finding a contention-free ring is presented in [81].

MULTITREE Algorithm

MULTITREE algorithm [75] was originally proposed for enhancing the efficiency of allreduce operations for deep learning training. The research proposes co-designing the allreduce communication algorithm with the interconnection architecture. The co-design has three core ideas:

- **Spanning Trees instead of rings:** aims to reduce algorithmic steps to $2\log_k n$ for k-ary trees, improving latency compared to $n-1$ steps in ring.
- **Topology awareness:** message scheduling is embedded to tree construction by using topology information.
- **Balanced communication:** iteratively choosing the links to build the schedule trees make the algorithm face no contention.

Briefly, the algorithm operates in two phases . Phase one, done once, build the schedule trees. The trees store the topology information. In addition, they are also the instructor for doing the operation. The tree will provide the sender/receiver and the data to send in each communication step. Phase two, do the collective operation based on the schedule trees.

Main idea of the tree construction (phase one): given a network graph $G(V, E)$ MULTITREE creates $|V|$ spanning trees. The trees are built from their roots in a top-down fashion, making the higher level denser and balancing communication across all tree level [3]. **Link Allocation and Scheduling:**

- For each time step (one tree level), a topology graph is used to allocate links to connect remaining nodes to the spanning trees.

[3]Using tree level to indicate communication step is a great way to save the schedule information, but this limits the flexibility of the construction of the trees. We will give example of this limitation in Chapter 3 and Chapter 4

- Once allocated, links are removed from the graph.
- When no more links are available for the current time step's connections or all the links in the topology graph are used, a new topology graph is used, and a new time step begins.

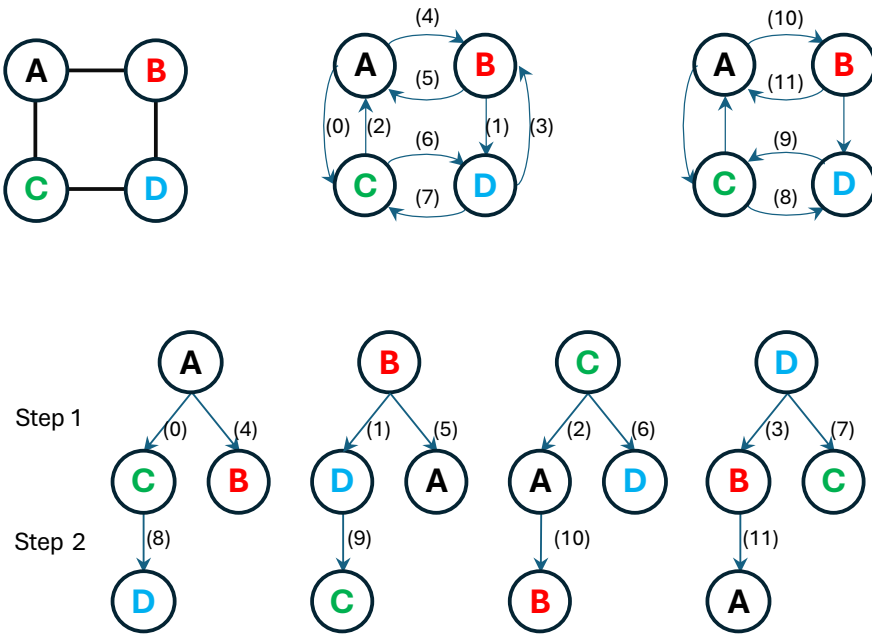


Figure 2.11: Tree building of MULTITREE algorithm for 2×2 Mesh.

Figure 2.11 shows the construction of schedule trees for 2×2 Mesh network. The algorithm starts with the initiation for four roots. For each time step a full copy of the topology graph $G'(V', E')$ is used. Trees take turn to add a new node c by connecting it to an already added predecessor node p. Then the $p \rightarrow c$ edge is removed from G' and scheduled for communication at the current time step. For example, in step one, links are removed from the graph and added to trees in round robin order. The number in the figure shows the order of links that added to the trees. After all links of G' are used, a new time step begins. For example, after all eight links of 2×2 Mesh are removed from the graph, a new graph are used. In Figure 2.11, links (8) to (11) belongs to the second time step. The process of adding links continues until nodes are connected to their respective trees. In static systems, phase one only runs once. In dynamic systems, it runs every when a new node allocated for the workload. After building the

schedule trees, now we can do the collective operations based on the network-topology information on the trees.

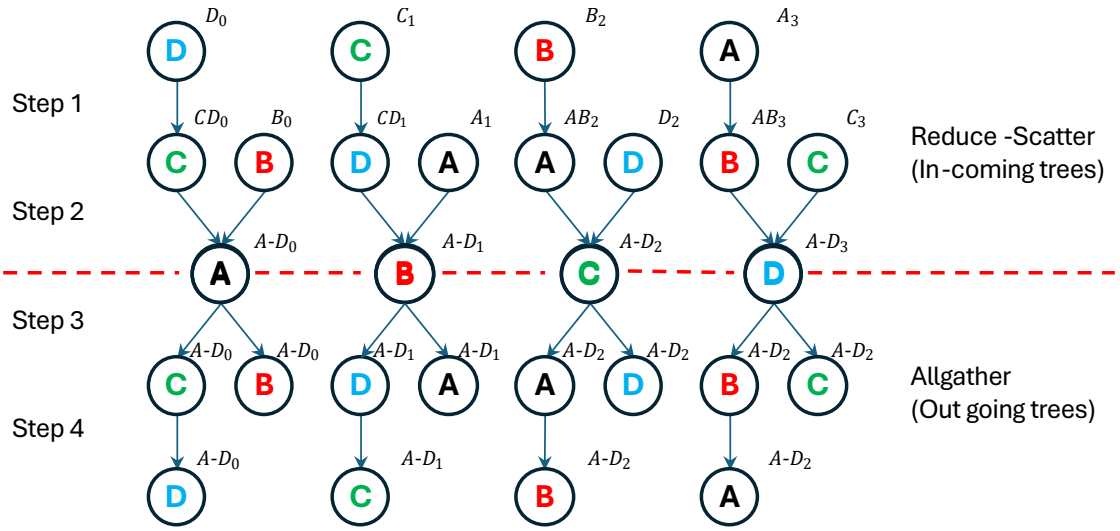


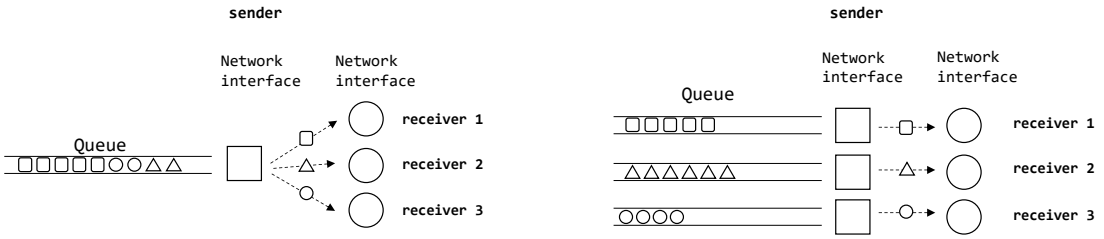
Figure 2.12: Allreduce using the schedule from MULTITREE algorithm.

Figure 2.12 shows the allreduce operation based on the tree information. Allreduce operation in the figure consist of two phases: reduce-scatter and allgather. Reduce-scatter on 2×2 Mesh has 2 communication steps. It uses the topology information from the in-coming trees. After two communication steps, reduce-scatter results are available in each process. After two steps allgather using the schedule information of outgoing trees, the allreduce completes.

For more information, formal definition and procedure of the algorithm can be found in the original paper [75].

As HPC systems become larger and more complex, collective communication has to evolve. Modern algorithms like MULTITREE help achieve fast and efficient data exchange by considering both hardware port capabilities and the network structure.

2.3 Network Interface Porting Capabilities on Collective Communication



(a) Single ported system **(b)** Multi ported system

Figure 2.13: Single-ported system vs Multi-ported system.

In addition to algorithmic design and network topology, the characteristics of the underlying hardware play a crucial role in collective communication performance. In particular, the porting capability of network interfaces is one of the most important factors. *Single-ported* network interfaces can transmit or receive only one message at a time, as shown in Figure 2.13a. The limitation in concurrency may lead to the communication serialization when multiple messages are sent to the network. *Multi-ported* interfaces, on the other hand, allow a node to send and receive multiple messages concurrently, as shown in Figure 2.13b, improving aggregate bandwidth. Multi-porting is especially important in collective operations when each process communicates with every other process. It enables overlapping of communications, which helps reduce the total communication time. Research on multi-ported technology and related tools has been ongoing for several decades[83, 84]. To the best of my knowledge, many HPC systems listed in the Top500 [1] currently use multi-ported switches as their network infrastructure. But research seen forgot this capability of the switch. This dissertation will analyze the port utilization of a conventional collective communication algorithm to propose an improvement of a conventional collective communication algorithm in 2dfc. Detail will be presented in Chapter 4.

2.4 Network Cost Model and Bandwidth Optimality in Collective Operation

Understanding the cost of communication between processes is important for designing efficient parallel algorithms. To analyze and predict communication performance, an effective model is necessary.

The $\alpha - \beta$ cost model, also known as the Hockney model [47], is one of the most widely used abstraction for qualifying the time to send m bytes between two processes. It models two fundamental components of communication cost: latency and bandwidth. The model give use the insights into how message size and network characteristics influence the communication performance.

The definition is as follows: the communication time $T(m)$ for sending m bytes message is given by:

$$T(m) = \alpha + m\beta \tag{2.14}$$

Where:

- α is the start-up latency. It is independent of message size. It is typically depended on hard/software overhead and propagation delay.
- m denotes the message size in bytes
- β is the per-byte transmission time. β is also understand as the inverse of bandwidth. It captures the time to transmit each additional byte of data.

The α-β model is simple and well-suited for analytical purposes; however, it does not account for network contention in its formula. In this dissertation, we examine only contention-free algorithms, so the α-β model remains sufficiently accurate.

Two most important factors contribute to the communication time is latency and bandwidth. This is shown in the widely accepted $\alpha - \beta$ model in Section 2.4, only the start up latency and the invert of the bandwidth shown in the formula. For large data sizes, the $m\beta$ term (related to bandwidth) dominates the α term (related to latency). Bandwidth optimal algorithms aim to minimize the influence of $m\beta$ to the total communication time. A key aspect of such algorithms is that each node transmits only the necessary data needed to complete the operation [70]. An algorithm achieving

bandwidth optimality communicates an amount close to the theoretical minimum which leads to the communication time close to the data size. Under the $\alpha - \beta$ model:

$$T(m) = m\beta$$

ignore the contribution of start-up latency. Another attention to bandwidth optimal algorithms is that network contention should be avoided. Contention can significantly reduce the available bandwidth and increase communication time [70]. Algorithm like Ring-based Allreduce [85, 86] for ring embedded topologies such as Torus and Mesh achieves contention-free communication, allowing them to use the network bandwidth efficiently.

Two conditions of bandwidth optimality that state in the work of Patarasuk et.al. [70] (which is widely used in HPC community)

- Each node sends the minimum amount of data required to complete the operation,
- All communications are contention free.

This dissertation will use these two conditions to design my algorithms.

Bandwidth optimal algorithms are typically preferred and shown their performance advantage when the message size are sufficiently large. This is because, for large messages, transfer time ($m\beta$) is the dominant factor in total communication time.

Note that the two conditions of bandwidth optimal algorithms do not specify how the links are used. As a result, two different bandwidth optimal collective communication algorithms on the same topology may still have different communication time. This is because they utilize the network links differently.

3

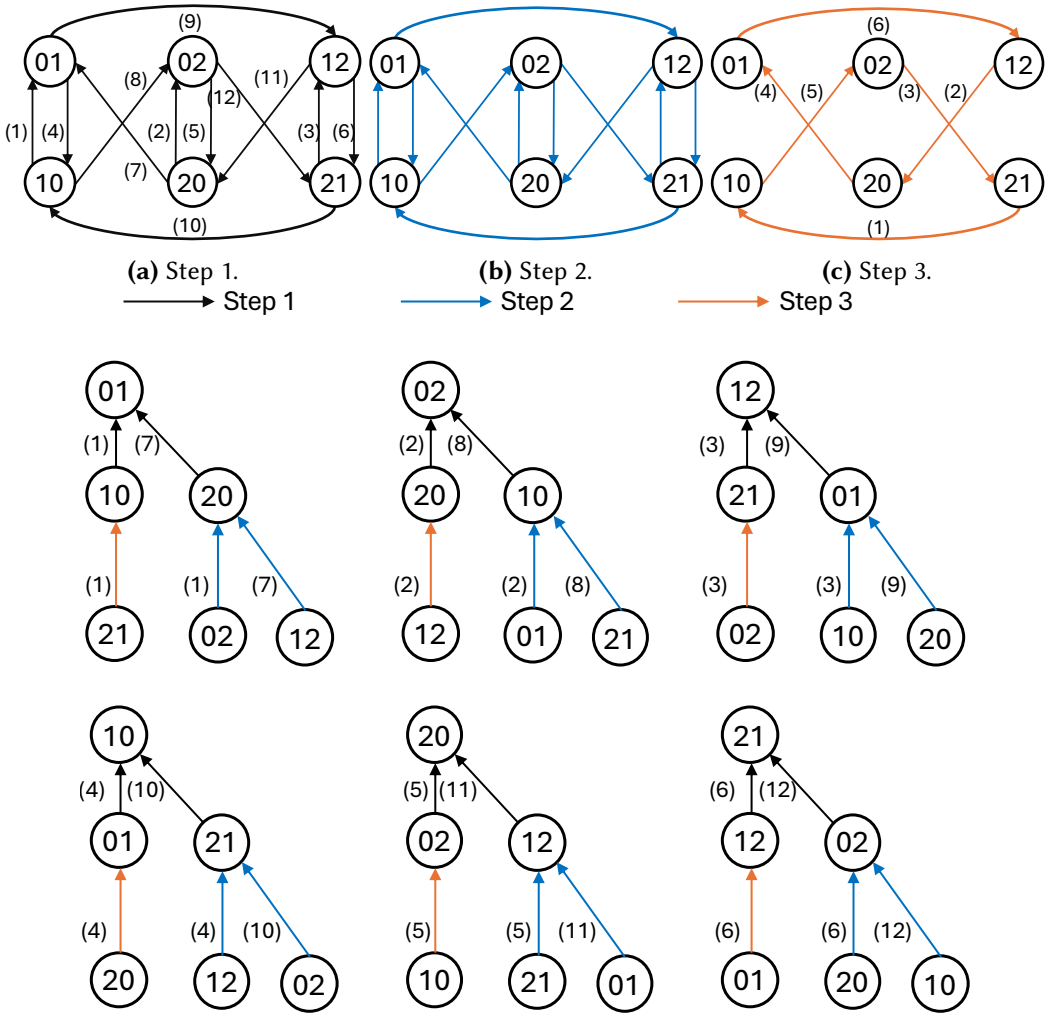
Collective Communication for Kautz Network Topology

In this chapter, we explore efficient collective communication in Kautz network topology. The selection of Kautz network topology is motivated by their favorable properties for large network, including low diameter, high connectivity, and deterministic construction. Kautz network topology is also known as the best known solution for degree-diameter graph problem. While Kautz network topology is promising for HPC interconnection network, their widespread adoption has been hindered by the lack of efficient communication algorithms. To address this problem, we develop a collective communication algorithm for the topology. We focus on two key ideas: multi-port operations and message-combine strategy. We design communication strategies that utilized all available links of a source node for collective communications. This approach aim to maximize network resource utilization. In addition to using multiple network port, we employ message-combine technique to aggregate multiple messages into a single larger message. This is important for reducing the communication latency

by minimizing startup overhead, especially for small message size.

3.1 Multi-port Collective Communication

Collective communication involves all the process in a communicator. The global behavior can lead to a bottleneck in some place in the network. Our objective here is to design a contention-free collective communication algorithm. In addition we also avoid indirect communication at each step of the collective communication process. Specifically, collective communication is built-up from multiple point-to-point communication between adjacent nodes. For example, the broadcast communication from one node to all other nodes can be performed along the Breadth First Search tree of the source node. Network contention can occur at any point throughout the collective communication process, e.g., when θ flows pass through node $\overline{A}$ and one of its nearby links at the same time. One of the strategies to avoid the contention is inspired by the MULTITREE algorithm [75]. Node $\overline{A}$ only sends one message over the link and saves the rest in its memory buffer, and as a result, contention communication is divided into θ sub-steps. Although this strategy ensures contention freedom, it raises the delay factor due to the increased number of communication steps. The collectives will be depended on the schedule trees of MULTITREE algorithm. We will take the procedure of building schedule trees for $Kautz(2, 2)$ as an example.



(a) Step 1. **(b)** Step 2. **(c)** Step 3.

(d) Schedule trees of MULTITREE algorithm.

Figure 3.1: Procedure of building incoming schedule trees of MULTITREE algorithm for $Kautz(2, 2)$.

The procedure follows the instructions of the MULTITREE algorithm, which can be summarized as follows.

Given a topology $G(V, E)$ with $|V|$ nodes and $|E|$ edges, the goal is to build $|V|$ trees, each containing all $|V|$ nodes.

- Initialize $|V|$ root nodes. Initialize a topology G.
- For each tree, if it is not complete (i.e., the number of nodes $< |V|$), add an edge to the current tree and remove the corresponding edge from the topology G.

Table 3.1: Type of trees used in different collective operations.

Kind of Tree	Collective
Incoming Tree	reduce, gather, reduce-scatter
Outgoing Tree	broadcast, scatter, allgather, alltoall

- When the topology G has no remaining edges, reset G and proceed to the next communication step until all trees are complete.

Figure 3.1 shows the procedure for building the incoming trees of the MULTITREE algorithm for $Kautz(2, 2)$. Six root nodes are initiated corresponding to six nodes of the topology. In the first step, black links in Figure 3.1a are added to the trees. The numbers beside the links indicate the order in which they are added. When all links in the topology are removed, a second topology is used. The blue links are added to the trees in the second step. In the last step, the orange links are added to the trees. Note that in the last step, only six per twelve links of the topology are used. Compared to the original MULTITREE algorithm, we simplify the building procedure. Instead of using the tree level to indicate the communication step, we use a separate table to store the order of links used in each communication step. By removing the constraint "each tree level is a communication step", we can build the schedule trees simpler. The additional table to store the link can be simplified by finding the rule of communication in each step. Therefore, we successfully remove the overhead from the table.

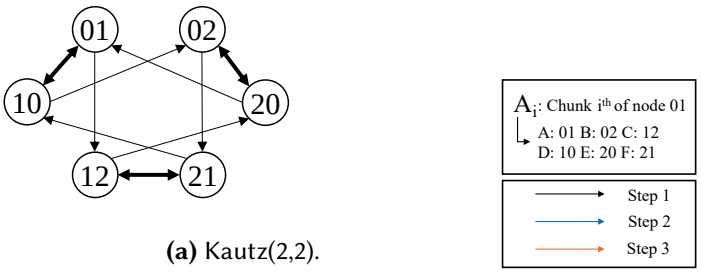
The procedure for building the the out-going trees is the same as the procedure for building the incoming trees. However, the order of used links is different.

Depending on the type of collective operation, either incoming or outgoing trees are used. Table 3.1 presents which type of tree is used in each collective.

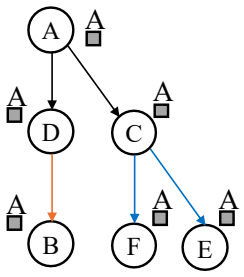
We, therefore, propose a concatenate strategy to reduce the number of communication steps. Node $\overline{A}$ aggregates θ messages and sends them as a single message of size $M \times \theta$ (assume that all messages have the same size M). This operation, message-combine, can significantly reduce the communication latency because the startup communication latency is not trivial, especially when the message size is small.

We discuss the details of each collective communication with the analytical model. It is worth noting that the algorithms used in reduce, gather, and reduce-scatter are comparable to those used in bcast, scatter, and allgather, respectively, but utilize a

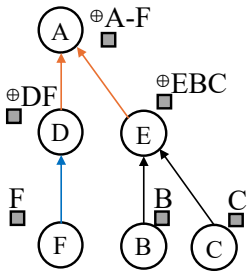
different subset of links. For example, the bcast from one node uses the outgoing links while reduce uses the incoming links, as the links are unidirectional and not symmetrical. Furthermore, it is possible to implement the allreduce operation as a combination of reduce-scatter followed by allgather (or reduce followed by bcast). We thus present only bcast, scatter, allgather, and alltoall in this study.



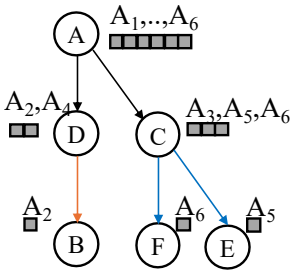
(a) Kautz(2,2).



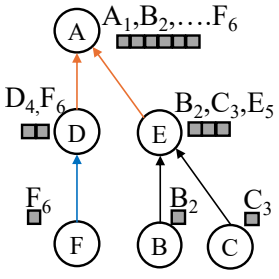
(b) Broadcast.



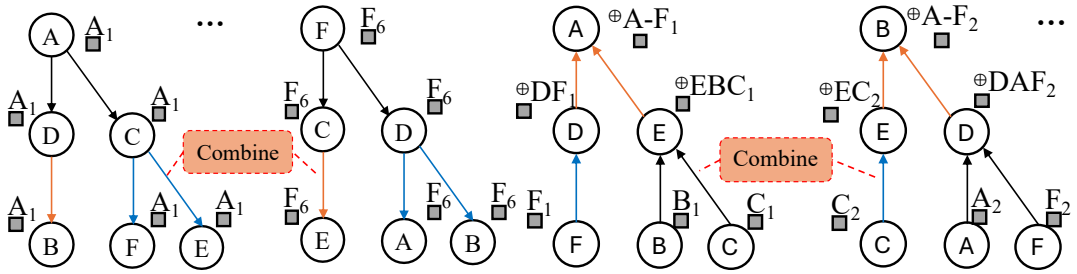
(c) Reduce.



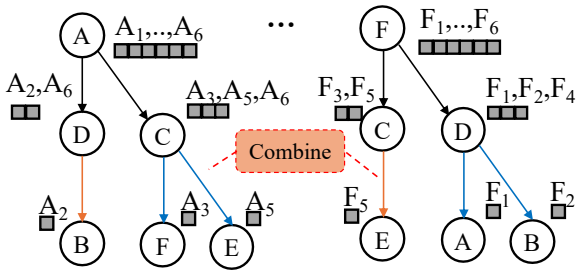
(d) Scatter.



(e) Gather.



(f) Allgather. **(g)** Reduce-Scatter.



(h) All-to-All.

Figure 3.2: Collective communication on degree-2, diameter-2 Kautz(2,2) network.

Assumptions and cost model We use the $\alpha-\beta$ model to estimate the communication cost (see Section 2.4). Ideally, in the absence of network contention, the unicast communication (or point-to-point communication) time to transfer a message of size M between two adjacent nodes is modeled as $T_{uc} = \alpha + M\beta$, where α is the startup time or the time it takes to send a small amount of data from source to destination and β is the time to inject one byte of data into the network (the inverse of the link bandwidth). We further assume that the startup time α is independent of message size, the number of nodes and the pair of two adjacent sender-receiver. As mentioned, the indirect point-to-point communication between two nodes of distance t is separated into t steps. Its cost is calculated as:

$$T_{uc}(t) = t \times T_{uc} = t(\alpha + M\beta) \tag{3.1}$$

When using the message-combine operation at a given step i, an node $\overline{A}$ aggregates θ_i messages and sends them as a single message of size $M_{A,i} \times \theta_i$. In this case, the communication time of this step can be modeled by $(T_{uc}^{\theta_i})_i = \alpha + (M_i\beta) \times \theta_i$, in which

$M_i = \max(M_{A,i})$ is the maximum message size transferred from one node at step i. When the message-combine is introduced, the total communication time becomes:

$$T_{mc}(t) = \sum_{i=1}^{t} (T_{uc}^{\theta_i})_i = \sum_{i=1}^{t} (\alpha + (M_i\beta) \times \theta_i) \tag{3.2}$$

One-to-all broadcast (bcast) One root node has a piece of data (of size M) that it needs to convey to every other nodes in the Bcast operation. We utilize a tree-based algorithm in which the outgoing broadcast tree is construct using the Breadth First Search (BFS) algorithm to prevent network contention during multiple one-hop data transfers[1]. Because a node can send and receive d messages simultaneously the Bcast is carried out in k steps for a specified diameter-k network. In one step, a node sends out a message of size M to all its child nodes simultaneously. The cost of Bcast is:

$$T_{bcast} = T_{reduce} = k \times (\alpha + M\beta) \tag{3.3}$$

One-to-all personalized (scatter) In the Scatter operation, a single node has a piece of data M that it needs to send to $n-1$ other nodes. Unlike the Bcast operation, which transmits the same piece of data to all the compute nodes, Scatter delivers distinct chunks of data (of size $\frac{M}{n}$) to different nodes. Despite this fundamental difference, the algorithmic structure is similar, it follows the BFS broadcast tree, except for the differences in communication message size at each step. Specifically, the message size for communication between a given node with depth i to its j^{th} child node in the BFS tree is proportional to the total number of compute nodes $n_{i,j}$ of this branch, i.e., $n_{i,j} \times \frac{M}{n}$ where $i = 0 \rightarrow (k-1)$ and $j = 1 \rightarrow d$. For example, the root node A needs to transmit messages of size $\frac{2M}{n}$ and $\frac{3M}{n}$ to its child nodes $\bar{D}$ and $\bar{C}$, respectively, in the first step, (as illustrated in Figure 3.2d). The communication cost of Scatter is:

$$T_{scatter} = T_{gather} = \sum_{i=1}^{k} (\alpha + \max_{j=1}^{d} (n_{i,j}) \frac{M}{n} \beta) \tag{3.4}$$

All-to-all broadcast (allgather) A node broadcasts the same piece of data to all the other nodes in an allgather operation, but distinct nodes broadcast different pieces of

[1]For the all-to-one reduction (Reduce) operation, we also construct a BFS tree from the root node using the incoming links (Figure 3.2b) and then performs in reverse order (Figure 3.2c)

data. We assume that the message size at each node is M without loss of generalization, i.e., the total buffer size of $n \times M$. We first construct n outgoing BFS trees separately and then free the contention at one step by using the message-combine operation or MULTITREE algorithms based on the data size (or message size). For example, at the second step, the link $\overline{CE}$ is used by the BFS tree of both $\overline{A}$ and $\overline{F}$ as shown in Figure 3.2f. If the message size is small enough, the node $\overline{C}$ will send one message that is a combination of A_1 and F_6 data chunks. The allgather operation, for example, is done in two steps, as indicated in Figure 3.2f. In the degree-d network, since each node has d incoming links and d outgoing links, a given node u appears at the depth $i(i = 0 \rightarrow k-1)$ of all the BFS trees at most d^i times. Therefore, an outgoing link between u and its neighbor v should be shared by at most d^i different flows. Thus, the communication time of allgather is:

$$T_{allgather} = T_{rs} = \sum_{i=0}^{k-1} (\alpha + d^i M\beta) \tag{3.5}$$

It is important to note that the cost of using the MULTITREE technique to avoid contention is $\sum_{i=0}^{k-1} d^i(\alpha + M\beta)$.

All-to-all personalized (alltoall) All-to-All is also known as a total exchange operation, in which all the nodes have their own piece of data M and need to send distinct chunks of data (of size $\frac{M}{n}$) to different nodes. Similar to the alltoall broadcast operation, a message-combine operation is applied with a consideration of the differences in communication message size at each step (Figure 3.2h). The alltoall operation requires k steps in which all steps $\sum_{i=0}^{k-1} d^i$ with a message of size $\max_{j=1}^{d} (n_{i,j})\frac{M}{n}$ are combined. In total the communication cost of alltoall operation is:

$$T_{a2a} = \sum_{i=0}^{k-1} (\alpha + d^i \times (\max_{j=1}^{d} (n_{i,j})\frac{M}{n})\beta) \tag{3.6}$$

Communication cost on diameter-2 networks

The communication costs discussed in the this section can be applied to any specific network topologies. As an example, the costs of collective communications on Fully Connected (FC), Dragonfly [7] and Kautz networks are summarized in Table 3.2. In the

Table 3.2: Comparison of collective communication cost on degree-d network. FC refers to a Fully Connected network. $d_i = \lceil d/2 \rceil$.

	FC	**Dragonfly**	**Kautz-Mtree**	**Kautz-Combi.**
Bcast/Reduce	$\alpha + M\beta$	$2(\alpha + M\beta)$	$2(\alpha + M\beta)$	$2(\alpha + M\beta)$
Scatter/Gather	$\alpha + \frac{M}{d+1}\beta$	$2\alpha + \frac{M}{d_i}\beta$	$2\alpha + \frac{(d+2)M}{d(d+1)}\beta$	$2\alpha + \frac{(d+2)M}{d(d+1)}\beta$
Allgather/ Reduce-Scatter	$\alpha + M\beta$	$2\alpha + (d_i + 1)M\beta$	$(d+1)(\alpha + M\beta)$	$2\alpha + (d+1)M\beta$
Allreduce	$\alpha + \frac{M}{d+1}\beta$	$4\alpha + 2\frac{M}{d_i}\beta$	$2(d+1)\alpha + 2\frac{M}{d}\beta$	$4\alpha + 2\frac{M}{d}\beta$
Alltoall	$\alpha + \frac{M}{d+1}\beta$	$2\alpha + 2\frac{M}{d_i+1}\beta$	$(d+1)\alpha + \frac{(2d+1)}{d(d+1)}M\beta$	$2\alpha + \frac{(2d+1)M}{d(d+1)}\beta$

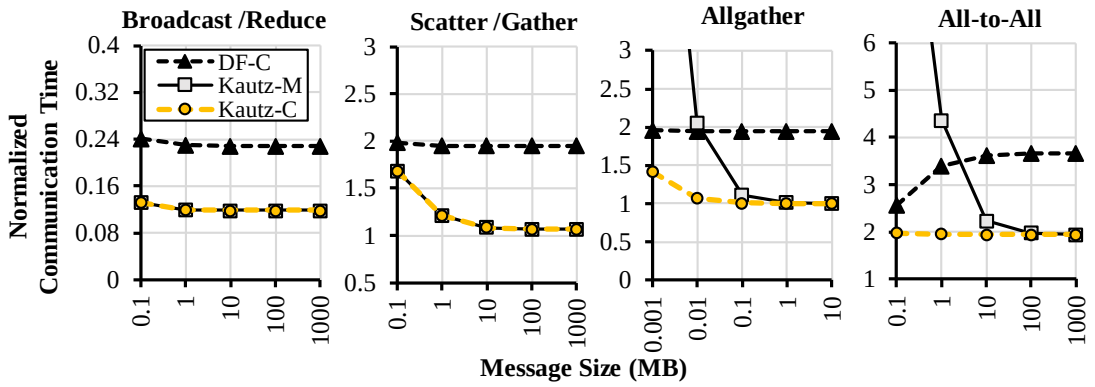
Kautz(d,2) network, because $\max_{j=1}^{d}(n_{1,j}) = d + 1$; $\max_{j=1}^{d}(n_{2,j}) = 1$ and $n = d(d+1)$, the cost of our proposed allgather algorithm becomes $2\alpha + (d+1)M\beta$ while that of the MULTITREE algorithm is $(d+1)(\alpha + M\beta)$. Our algorithm reaches the lower bound of latency, i.e., $O(\log_k(n))$, and of bandwidth, i.e., $O(\frac{M(n-1)}{\min(d,n-1)})$ as reported in [69].

3.2 Performance Evaluation

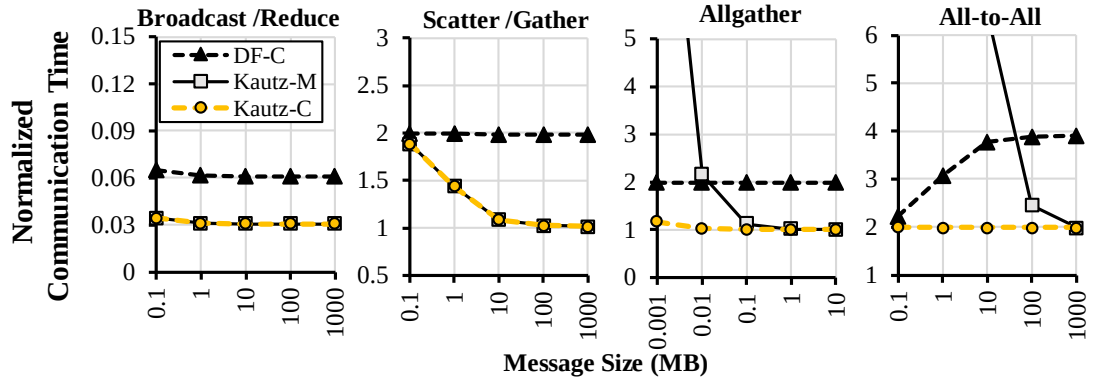
In this section, we measure the performance of collective communications using our proposed concatenate algorithm in the fully connected (FC), Dragonfly (DF-C), and Kautz (Kautz-C - our proposed method) networks at a large scale. We also compare the performance with the algorithm using the MULTITREE technique to avoid network contention (named as Kautz-M). We first estimate the ideal communication time and then simulate the targeted collective communication using a discrete-event simulator, i.e., SimGrid version 3.28 [87].

In this comparison, we assume an node behaves like a network interface of the CPU server. We fix the number of nodes n, i.e., network degrees are different for a distinct targeted network. Specifically, we compare a degree-d Kautz network with a degree-$(2d-1)$ Dragonfly network. We also fix the total amount of bandwidth B per node, that is, a link from the degree-d network has a bandwidth of $\frac{B}{d}$. We select the start-up latency from the real average experiment result, i.e., $\alpha \approx 1$ microsecond.

3.2.1 Numerical Analysis



(a) Kautz(16,2) - 272 nodes



(b) Kautz(64,2) - 4, 160 nodes

Figure 3.3: Normalized communication time on Kautz and Dragonfly (DF) to those on the fully connected network.

In this section, we measure the ideal communication time using the theoretical cost stated in Table 3.2. Note that the cost of Kautz-M and Kautz-C is different in the number of communication steps only. For example, Kautz-M requires $d + 1$ steps while Kautz-C requires only 2 steps for the allgather operations. Figure 3.3 shows the normalized communication time of the targeted collective operations on the Dragonfly and Kautz network to those on the fully connected network in the case of 272 and

4, 160 nodes[2]. The lower values are considered better.

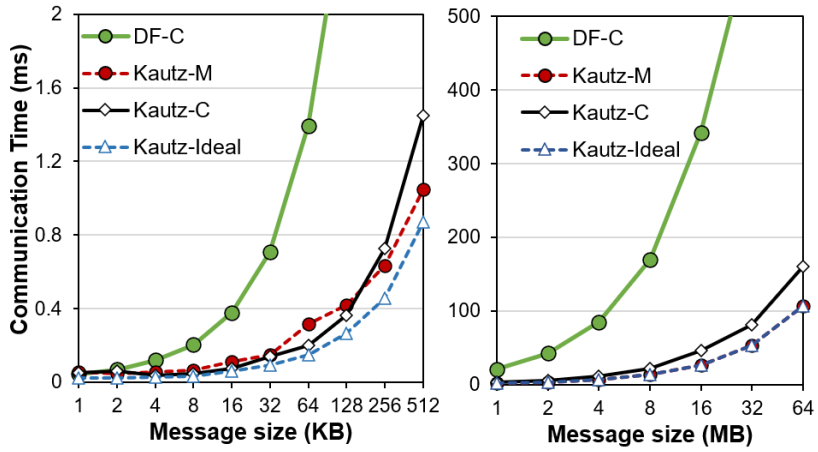
Interestingly, the broadcasting time of both DF-C and Kautz-C is significantly less than that of FC when the message M is varied, e.g., only 6% and 3% in the case of 4,160 nodes, respectively. This is because the link bandwidth of FC is much small than that of FC and Kautz, i.e., a higher number of links per node. For scatter operation, both Kautz-C and Kautz-M gradually reach the performance of FC while the communication time of DF-C is about 2× that of FC. Note that, it would be possible to implement the bcast operation by a combination of the scatter operation followed by an allgather operation. In this scenario, the relative performance of bcast would likely have the same behavior as scatter. Similar to scatter, our proposed Kautz-C algorithms have a performance comparable to FC in allgather (≈ 1×) and alltoall (≈ 2×). It is also up to 2× faster than DF-C when the message size is large enough. Our proposal significantly improves the performance of Kautz-M, especially when the message size is small, e.g.,by more than 20× in the case of 100-KB message size.

These findings demonstrate that our proposed collective communication algorithms can achieve the best performance among the targeted diameter-2 networks and algorithms.

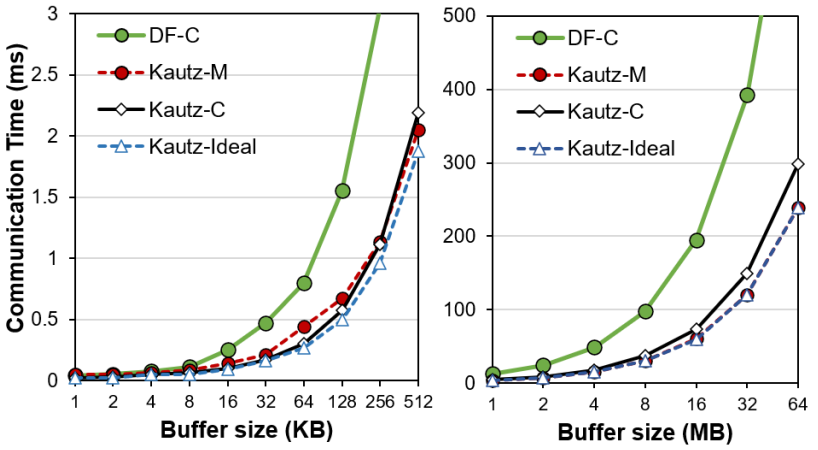
3.2.2 Micro-Benchmark Simulated Results

We next evaluate the performance of our proposed collective communication algorithms in real systems. Our targets are the collective operations that may have message contention such as allgather, alltoall and allreduce. We first implement the operations in C++ using MPI libraries. We then show that our proposed algorithm is well optimized for the Kautz network by simulating the MPI algorithms on SimGrid simulator [87]. To simulate the multi-port collective communication algorithm, we use the non-blocking MPI, i.e., isend/ irecv, which is supported by SimGrid.

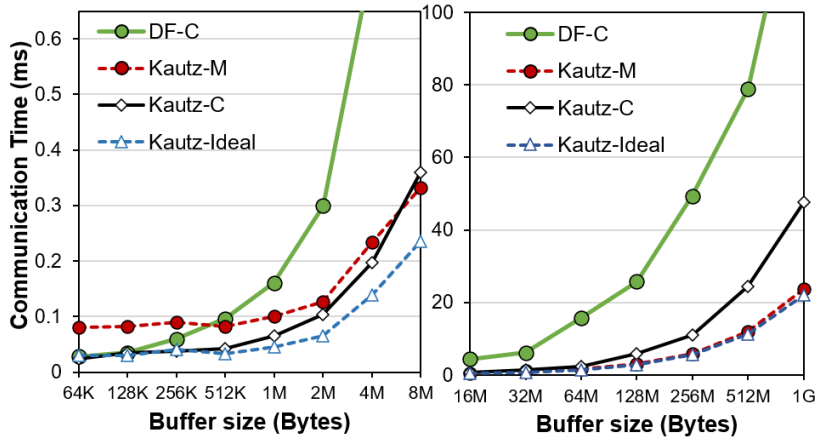
[2]Kautz network of degree 16, and 64, respectively. There was a similar result for networks of 1, 056 nodes, e.g., Kautz network of degree 32.



(a) Allgather



(b) AlltoAll.



(c) Allreduce.

Figure 3.4: Execution time of Allgather, AlltoAll, and Allreduce benchmarks on SimGrid (272 nodes).

The resultant communication times of the targeted operations are shown in Figure 3.4 when the message size M ranges from 1KB to 64MB, i.e., when the total data buffer at each node ranges from 272KB to 17GB, respectively. The environment for the micro-benchmark showed in Table 3.3.

Table 3.3: Environment setup for collective experimental result.

Host System		
	Operating System	SMP Debian 3.16.64-2
	Processor	Intel(R) Xeon(R) CPU E5-2667 v4 @ 3.20GHz
Simulation environment		
	SimGrid	v3.28
	Topology	Kautz(16,2) - 272 nodes, bandwidth 12.5GBps/link
		Dragonfly 16x17 - 272 nodes, bandwidth 6.45GBps/link
	Startup latency	1µs
	Host speed	100Gflops

As expected, the results in Figure 3.4 support our findings on communication cost discussed in the previous subsection. Both the Kautz-C and Kautz-M algorithms significantly reduce the communication time of the DF-C algorithm on the Dragonfly network. Interestingly, when the message is small, e.g., less than 128 KB in the case of the allgather benchmark, Kautz-C outperforms the DF-C and Kautz-M algorithms. However, when the message size becomes large, e.g., $2 \longrightarrow 64$ MB, the performance of Kautz-C is around 1.5× and 1.1× slower than Kautz-M in the cases of allgather and alltoall, respectively. In particular, Kautz-C and Kautz-M are 7× and 11× faster than DF-C in the case of messages sized 32MB and 64MB, respectively. This is because our implementation of message-combine operations requires a *memcpy* at the end of the communication to re-organize the order of data segments in the combined messages. The extra overhead of this *memcpy* is proportional to the message size and could not be avoided. In this case, we suggest falling back to using our Kautz-M (Tree) algorithm. Its performance is equivalent to that in the ideal case (no contention, no *memcpy* overhead), i.e., the blue dashed line in Figure 3.4.

3.2.3 Real Workload Simulated Results

In this section, we examine the overall performance of our proposal with a typical application that requires memory-to-memory communication. We utilize the sharing averaged parameters application mentioned in [88] which is frequently used in the training process of recent Deep Learning (DL) workloads in parallel [89]. At the end of each iteration during the training process, all the computing elements, e.g., GPUs or FPGAs, have their local trained parameters, or weight gradients of the DL model. The global parameters are then calculated by averaging all the local parameters using an allreduce communication.

We utilize a method similar to the one in [5], where the computation time and communication time are estimated separately. We empirically measure the (average) computation time for training one batch of samples on the real system with NVIDIA Tesla V100 and use the result for estimating the total computation time per epoch (the time for covering all the samples of the dataset). We then simulate the communication time on SimGrid, i.e., we perform allreduce with a buffer size equal to the size of the DL parameters (4 bytes/parameter). In this work, we estimate the training time of ResNet50 (25M parameters) [90] with a batch size of 64 and VGG16 (138M parameters) [91] with a batch size of 32 on the ImageNet-1K dataset [92] (1.28M samples).

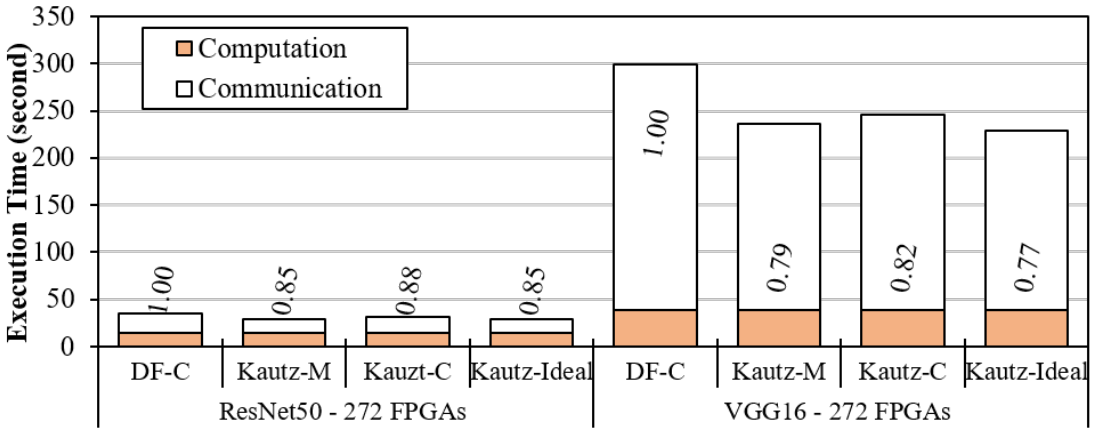


Figure 3.5: Simulated training time in one epoch of ResNet50 and VGG16. The label at each column shows its relative execution time to that of the baseline FC_C.

Figure 3.5 shows the breakdown of training time in one epoch. Overall, the relative

relation between our targeted system and the collective communication algorithms does not change. Specifically, the training process with the Kautz networks outperforms those of the Dragonfly networks because of a significant reduction in communication time. To this end, the impact of our proposal is stronger when training a bigger model. For example, the training time of ResNet50 and VGG16 with our proposed Kautz-M method is only 85% and 79% that of DF-C, respectively.

3.3 Discussion: Point-to-Point Communication

3.3.1 Strategy

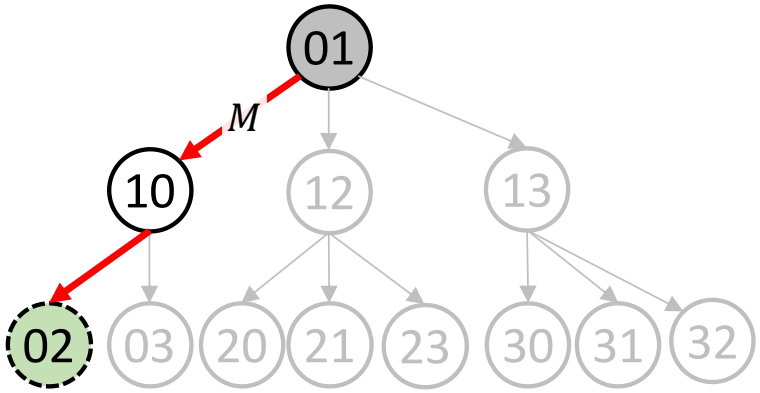


Figure 3.6: Single-path point-to-point communication on degree-3, diameter-2 Kautz(3,2) network from the source node (gray) to the destination node (dashed border, green).

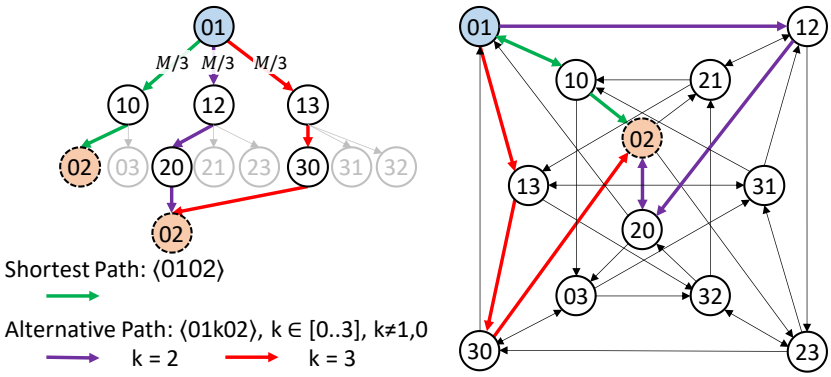
Conventional approach with a single shortest routing path uses only one single port of the high radix source node (Figure 3.6). Many links are unused which may lead to inefficiency in communication. Our objective here is to reduce the communication time by using multiple routing paths. Specifically, a message can be divided equally and sent simultaneously via both the shortest routing path and non-shortest alternative routing paths, i.e., via at most d routing paths through its d different ports. In this dissertation,

we only focus on paths whose lengths are less than or equal to $k + 1$ hops, where k is the network diameter, because increasing the routing path length may lead to the increase of latency, and affect the performance. For example, three partitions of the message M are routed via the shortest path and two other alternative length-3 routing paths as shown in Figure 3.7a. The number of alternative paths is selected based on the relative position between the source and destination. Three kinds of relative position is presented as follows.

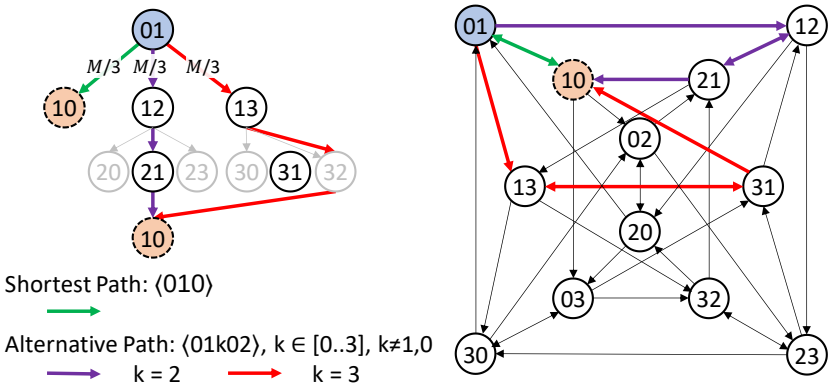
Type 1 ($\overline{xy} \rightarrow \overline{zt},\ z \neq y$): there is no direct link from a source to a destination. In this case, there are one length-2 shortest path, i.e, $P(\overline{xy}, \overline{zt}) = \langle \overline{xyzt} \rangle$, and $d - 1$ length-3 alternative paths $P(\overline{xy}, \overline{zt}) = \langle xykzt \rangle, k \in [0..d], k \neq y, z$.

Type 2 ($\overline{xy} \rightarrow \overline{yx}$): there is a bidirectional link between a source and a destination. In this case, there are one length-1 shortest path $P(\overline{xy}, \overline{yx}) = \langle \overline{xyx} \rangle$ and $d - 1$ length-3 alternative paths i.e, $P(\overline{xy}, \overline{yx}) = \langle xykyx \rangle, k \in [0..d], k \neq x, y$

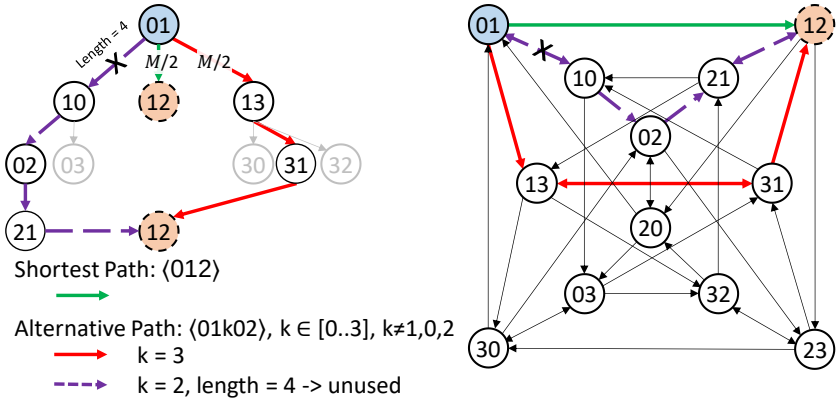
Type 3 ($\overline{xy} \rightarrow \overline{yz},\ z \neq x$): there is a unidirectional link from a source to a destination. In this case, there are one length-1 shortest path i.e, $P(\overline{xy}, \overline{yz}) = \langle \overline{xyz} \rangle$, and $d - 2$ length-3 alternative paths i.e, $P(\overline{xy}, \overline{yz}) = \langle xykyz \rangle, k \in [0..d], k \neq x, y, z$. Noted that, we do not use the length-4 path. Because it increases the startup latency which can affect the performance of the transmission.



(a) Type 1



(b) Type 2



(c) Type 3

Figure 3.7: Multi-path point-to-point communication on Kautz(3,2). Source nodes are drawn in blue. Destination nodes are drawn in orange and dashed border.

Consider $Kautz(3, 2)$ as an example (Figure 3.7). Figure 3.7a shows an instance of Type 1. The communication is between source node $\overline{01}$ and destination node $\overline{02}$. In this type, all the outgoing links of the source node is used: one link for shortest path and two links for alternative paths. The length for the shortest path is equal to two while the length of two alternative path is three. In Type 2 (Figure 3.7b), all the outgoing links are also used. The difference is that the shortest path of Type 2 has length one. In the Type 3 (Figure 3.7c), messages can only be sent in two over three links. The reason is that the length-four link is not considered in this dissertation.

Notice that in the case of Type 3 in Kautz(2,2) (Figure 3.8), due to the length of the alternative is equal to four (the purple, dashed link), we do not consider this link. Therefore the performance of point-to-point communication in this case is equal to single shortest path.

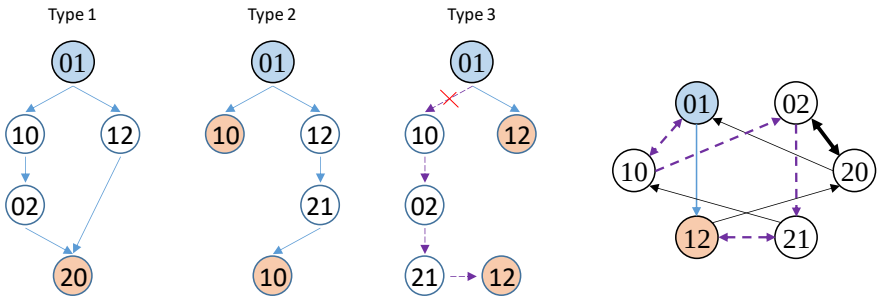


Figure 3.8: Multi-path point-to-point communication on Kautz(2,2). Source nodes are drawn in blue. Destination nodes are drawn in orange.

The summary of two point-to-point communication is as follows.

- Single-path: only the shortest path is used. The shortest path has length no greater than 2 in the degree d diameter 2 Kautz graphs.

- Multi-path: a message is fragmented into a number of partitions that depended on the relative position of the source and the destination. The partitions will be sent up to d paths simultaneously. This will significantly reduce the communication time of point-to-point communication, especially for large degree network.

3.3.2 Performance Evaluation

Numerical Analysis Evaluation

Message inside Kautz network degree d diameter 2 - $Kautz(d, 2)$ - can be transferred by shortest path length $\leqslant 2$. Between any pair of arbitrary nodes $a = \overline{xy}$ and $b = \overline{zt}$, if there is a direct link between a and b, a message is sent directly through the path $P_{direct} = \langle xyt \rangle (y = z)$. On the other hand, the message is sent over the path $P_{indirect} = \langle xyzt \rangle$ through an intermediate node $\overline{yz}$. From an arbitrary source node a, there is d nodes that connect to a by direct links which cost $\alpha + M\beta$ time units to send a message size M. $d^2 - 1$ remaining nodes transfer messages to source node a through an indirected path which costs $2\alpha + 2M\beta$ time units per transfer. Therefore the average time to send a message when using a single shortest path is equal to:

$$\begin{aligned} T_{uc}^{avg} &= \frac{1}{d(d+1))}(\sum time(direct) + \sum time(indirect)) \\ &= \frac{1}{d(d+1)}(d \times (\alpha + M\beta) + (d^2 - 1) \times (2\alpha + 2M\beta)) \\ &= \frac{2d^2 + d - 2}{d(d+1)}\alpha + \frac{(2d^2 + d - 2)M}{d(d+1)}\beta \end{aligned} \tag{3.7}$$

Considering the cost of multi-path routing P2P communication on a Kautz(d,2) network, we have:

Type 1 since d disjoint partitions of the message (with the size $\frac{M}{d}$) are sent through these d routing paths simultaneously, the communication cost, in this case, is $T_{mp,1} = \max_{1 \leq i \leq d}(2\alpha + 2\frac{M}{d}\beta, 3\alpha + 3\frac{M}{d}\beta) = 3(\alpha + \frac{M}{d}\beta)$.

Type 2 ($\overline{xy} \rightarrow \overline{yx}$): similar to the type 1, the communication cost in this case is $T_{mp,2} = 3(\alpha + \frac{M}{d}\beta)$.

Type 3 ($\overline{xy} \rightarrow \overline{yz}; z \neq x$): the communication cost in this case is $T_{mp,3} = 3(\alpha + \frac{M}{d-1}\beta)$.

For a given source node in a Kautz(d,2) network, there are one type-2 destination, $d - 1$ type-3 destination, and $d^2 - 1$ type-1 destination. Therefore, the average cost of

point-to-point communication with the multi-path routing is

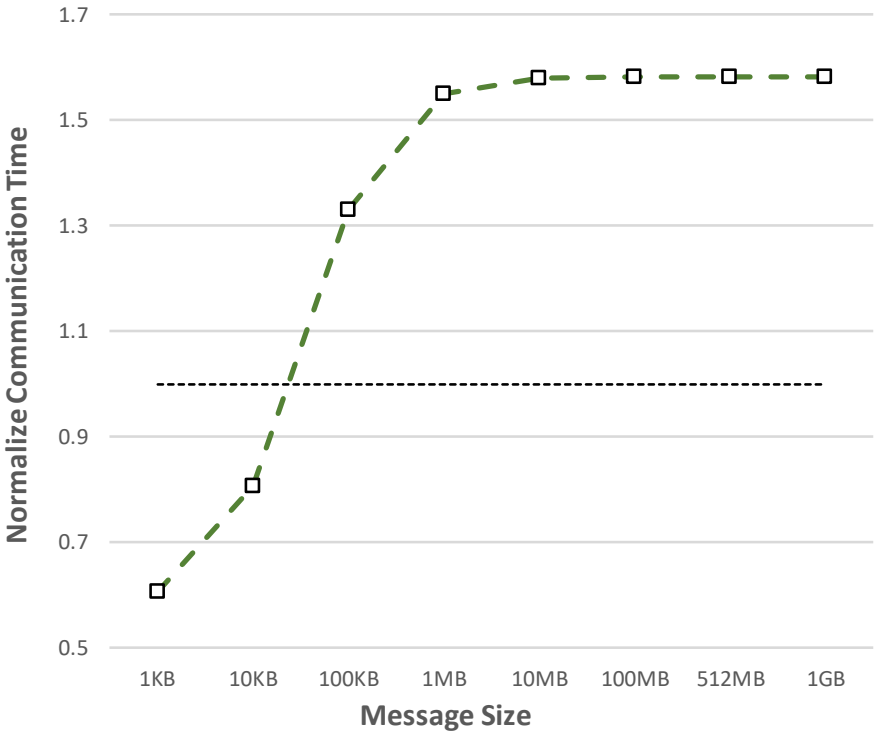
$$\begin{aligned} T_{uc,mp}^{avg} &= \frac{1}{d(d+1)}\Big(3(\alpha + \frac{\beta M}{d}) + (d-1)\times 3(\alpha + \frac{\beta M}{d-1}) + (d^2-1)\times 3(\alpha + \frac{\beta M}{d})\Big) \\ &= 3(1 - \frac{1}{d(d+1)})\alpha + 3\frac{M}{d}\beta < T_{uc}^{avg} \text{ for M large enough} \end{aligned} \tag{3.8}$$

Figure 3.9 shows the normalized ideal communication time of multi-path to single-path point-to-point algorithm. Assume that, common network parameters are used i.e, $\alpha = 1\mu s,\ \beta = 8 \times 10^{-11} s$ (equivalent to 100Gbps). The figure shows the consistent results between topology at small scale 3.9a and large scale 3.9b. When message is small, the startup time is dominant. This leads to the performance of multi-path algorithm becoming worse (normalized time less than one). When the message size increases to some certain point, the benefit of multi-path algorithm takes over the startup latency. Beyond that point, the speed up between single-path algorithm and multi-path algorithm increases and converges to:

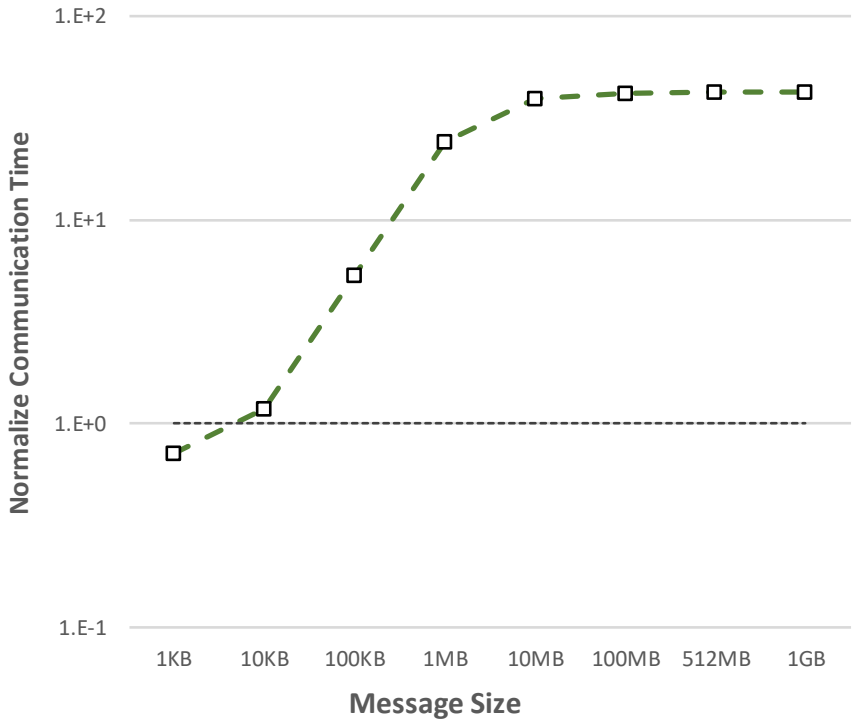
$$\begin{aligned} \lim_{M\to\infty} \frac{T_{uc}^{avg}}{T_{uc,mp}^{avg}} &= \lim_{M\to\infty} \left(\frac{2d^2+d-2}{d(d+1)}\alpha + \frac{(2d^2+d-2)M}{d(d+1)}\beta\right) \div \left(3\left(1 - \frac{1}{d(d+1)}\right)\alpha + 3\frac{M}{d}\beta\right) \\ &= \lim_{M\to\infty} \left(\frac{(2d^2+d-2)M}{d(d+1)}\beta\right) \div \left(3\frac{M}{d}\beta\right) \\ &= \left(\frac{(2d^2+d-2)}{d(d+1)}\right) \div \left(\frac{3}{d}\right) = \frac{2d^2+d-2}{3(d+1)} \\ &\approx \frac{2}{3}d \end{aligned} \tag{3.9}$$

Micro Benchmark

In this section, we measure the performance of point-to-point communication strategies which are multi-path and single-path. We implement two strategies in C/C++ using MPI libraries. Then, the topology and the network information are simulated inside SimGrid simulator [87]. The average communication time showed in Figure 3.10 when the message size M ranges from 1KB to 1GB. The detail of the experimental environment is described in the Table 3.4.



(a) Kautz(3,2) - 12 nodes.



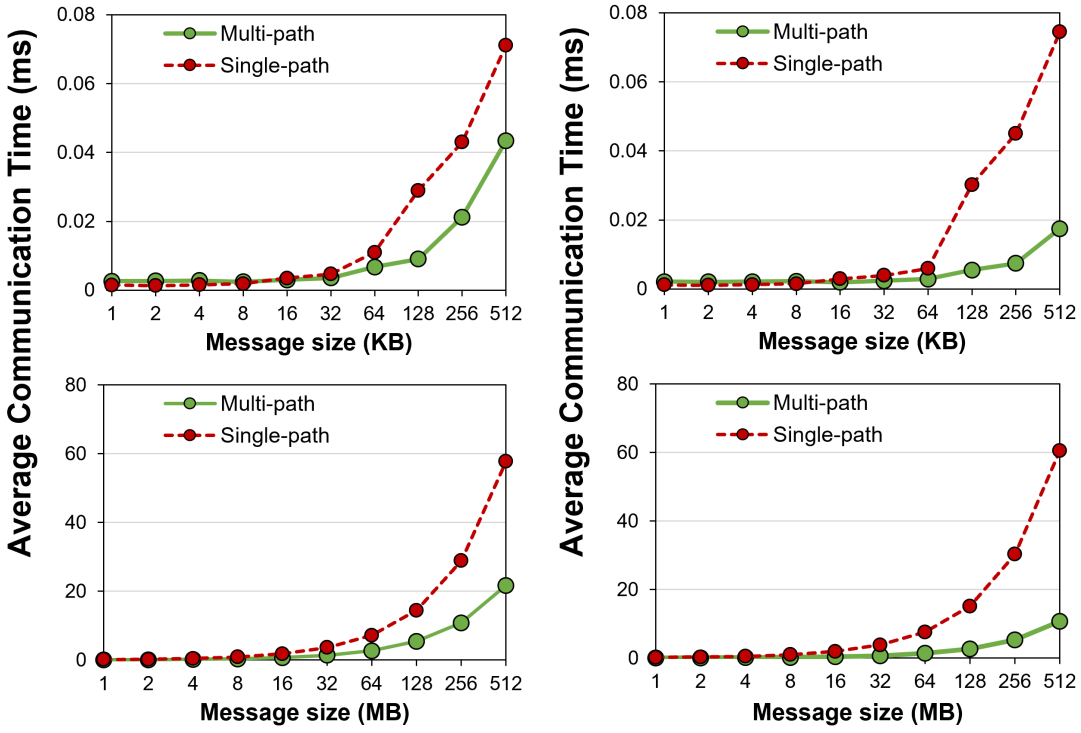
(b) Kautz(64,2) - 4160 nodes.

Figure 3.9: Normalized communication time of single-path point-to-point communication $T_{uc,mp}^{avg}$ to that of multi-path communication T_{uc}^{avg}.

Table 3.4: Environment setup for point-to-point experimental result.

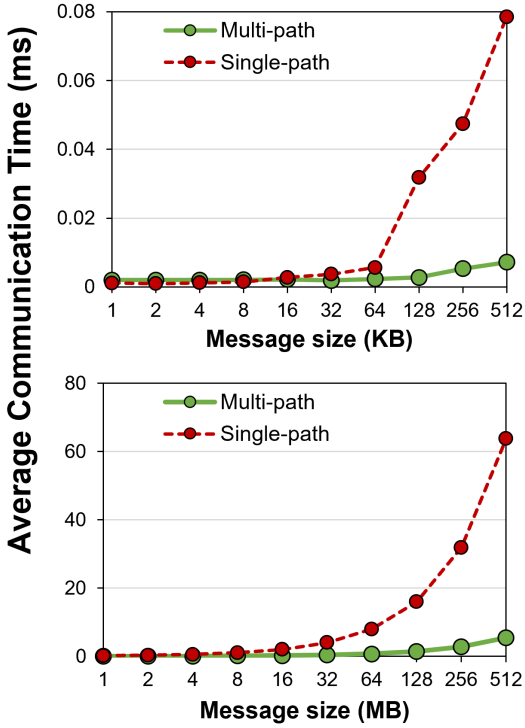
Host System		
	Operating System	SMP Debian 3.16.64-2
	Processor	Intel(R) Xeon(R) CPU E5-2667 v4 @ 3.20GHz
Simulation environment		
	SimGrid	v3.28
	Topology	K(4,2) - 20 nodes, K(8,2) - 72 nodes, K(16,2) - 272 nodes
	Link Bandwidth	100Gbps
	Startup latency	1µs
	Host speed	100Gflops

The results, showed in Figure 3.10, support our finding on the communication cost discussed in the previous section. When the message size is small, the performance of single-path is a bit better than that of multi-path. However, the time consuming of single-path increases in a higher order of magnitude than the time of multi-path. To this end, the performance of multi-path is around 6.1× to 10.3× faster than single-path when the message size ranges from $2MB \rightarrow 500MB$ in Kautz(16,2). This confirms the theoretical analysis on section 3.3.2.



(a) Kautz(4,2) - 20 nodes.

(b) Kautz(8,2) - 72 nodes.



(c) Kautz(16,2) - 272 nodes.

Figure 3.10: Average execution time of point-to-point communication on SimGrid.

We introduce a multi-path point-to-point communication, which is compared to shortest single-path, on $Kautz(d, 2)$ graph. The experimental results show that the multi-path communication can speed up 10× for large message size in Kautz(16,2) with 272 nodes. This result is also close to the theoretical analysis.

3.4 Summary

In this chapter, we focused on developing multi-port message-combine collective communication algorithms specifically for Kautz network topology. This network topology has great properties for building collective communications for large networks such as low diameter and deterministic construction. The core idea was to design multi-port operations that exploit all available links of a source node during collective communication. In addition, we employed message-combine strategy to aggregate messages efficiently. With these improvements, we enhanced the network resource utilization, avoided contention, and reduced communication latency.

Our main findings through the simulation are listed as follows.

- For small message size (for example less than 128 KB in allgather), the proposed Kautz-C (message-combine) algorithm significantly outperformed both the Dragonfly (DF-C) and MULTITREE (Kautz-M) algorithms, achieving more than a 20× improvement over Kautz-M for a 100 KB message size. This advantage results from Kautz-C's ability to reduce the number of communication steps. As a result, it minimizes the impact of startup latency.

- For large message sizes (2 MB to 64 MB), both proposed algorithms, Kautz-C and Kautz-M, achieved substantial performance gains over DF-C, being up to 7× and 11× faster for 32 MB and 64 MB messages in allgather and alltoall benchmarks, respectively.

- Simulations of deep learning training workloads (ResNet50 and VGG16) showed that the multi-port message-combine collective communications on Kautz network topology reduced training time to 85% and 79% respectively, compared to the Dragonfly network topology. The improvement shows the algorithm's efficiency for parallel applications.

In essence, we borrowed the Kautz network topology from graph theory to develop a low-latency collective communication framework. The key techniques underpinning this design are multi-port and message-combine strategies, which lead to significant performance improvements particularly for latency-sensitive tasks and large-scale deep learning workloads in HPC environments. We also discussed the possibility of multi-path point-to-point communication, with results showing substantial performance gains for large messages.

4

Collective Communication for Two-Dimensional Fully Connected Network Topology

Uni-directional Kautz network topology has attractive statistics of low diameter and low average shortest path length. In the previous chapter, we explore efficient collective communications for the Kautz network topologies. However, existing HPC systems have not been used uni-directional interconnection networks because of their relevant practical concerns, such as the lacks of efficient switch-by-switch flow control. In this context, more practically, we focus on the collective communication in 2dfc network topologies.

4.1 Limitations of Conventional and Contemporary Collective Algorithms for 2dfc

The 2dfc is widely used in the world of HPC networks. It can serve as the primary topology to connect an entire system as in [8, 9, 6]. Alternatively, it can be used to connect a local group as in the case of Cray Aries interconnect [43] which is employed in the Piz Daint [44] and Trinity supercomputers [45]. Furthermore, 2dfc is also used as the global topology to connect groups of nodes as demonstrated in Hamming Mesh [42]. Thus, the demand for collective operation in 2dfc is high.

Collective operation on 2dfc can be approximately classified into two schools of thought: 1. Topology specific algorithm, such as Dimensional Order (DO) 2. Topology agnostic algorithm, such as MULTITREE algorithm

This chapter mainly focuses on the alltoall operation, as it captures the essential communication pattern underlying several other collective operations. Since the differences among these operations lie only in the data exchanged, the insights obtained here directly extend to other operations. In this context, we will detail the specific limitations of two key algorithms: DO and MULTITREE.

4.1.1 The Problem of Dimension Order Algorithm

DO is a naive version of the topology-aware alltoall algorithm for 2dfc. The algorithm exchanges the data of the alltoall operation in one dimension. Then, in the next communication step, the algorithm exchanges the remaining data in the remaining dimension. A similar algorithm is described in 3D torus in [93].

Input for the algorithm is the shape of 2dfc ($m_2 m_1$) and the process name $a_2 a_1$ (illustrated in Figure 4.1). Data in each process is stored in an array D which has $N = m_2 \times m_1$ (the number of processes) data blocks as the notation in Section 2.1.4. Note that in this section we change the indexing of a process from an integer i to a 2-tuples ($a_2 a_1$). We can easily convert two kinds of notation by the expression $i = a_2 \times m_1 + a_1$.

Algorithm 1: DO: All-to-All Collective Communication on 2lvfc

Data: Data Array D
Result: Each process receives data from all other processes
Input: Topology 2lvfc(m_2, m_1), process (a_2a_1)

// All-to-all exchange in the first dimension
// at the same time send data for second dimension exchanging
1 **for** $k \leftarrow 1$ **to** $m_1 - 1$ **do**
2 $a_k \leftarrow (a_1 + k) \mod m_1$; // (a_2a_k) is the receiver
3 SB appends $D[a_2a_1, a_2a_k]$; // init sendbuf SB
4 **for** $z \leftarrow 1$ **to** $m_2 - 1$ **do**
5 $a_z \leftarrow (a_1 + z) \mod m_1$
6 SB $\leftarrow$ SB appends $D[a_2a_1, a_2a_z]$
7 Send SB to node (a_2a_k)
8 **for** $z \leftarrow 1$ **to** $m_1 - 1$ **do** // add data for second step
9 $a_z \leftarrow (a_1 + z) \mod m_1$; // (a_2a_z) is the sender
10 Receive from node (a_2a_z)

// All-to-all exchange in the second dimension and finish the algorithm
11 **for** $k \leftarrow 1$ **to** $m_2 - 1$ **do**
12 $a_k \leftarrow (a_2 + k) \mod m_2$; // (a_ka_1) is the receiver
13 SB appends $D[a_2a_1, a_ka_1]$; // init sendbuf SB
14 **for** $z \leftarrow 0$ **to** $m_1 - 1$ **do**
15 $a_z \leftarrow (a_1 + z) \mod m_1$
16 SB $\leftarrow$ SB appends $D[a_2a_z, a_ka_1]$; // $D[a_2a_z, a_ka_1]$ received from the first step
17 Send SB to node a_ka_1
18 **for** $z \leftarrow 1$ **to** $m_2 - 1$ **do**
19 $a_z \leftarrow (a_2 + z) \mod m_2$; // (a_za_1) is the sender
20 Receive from node (a_za_1)

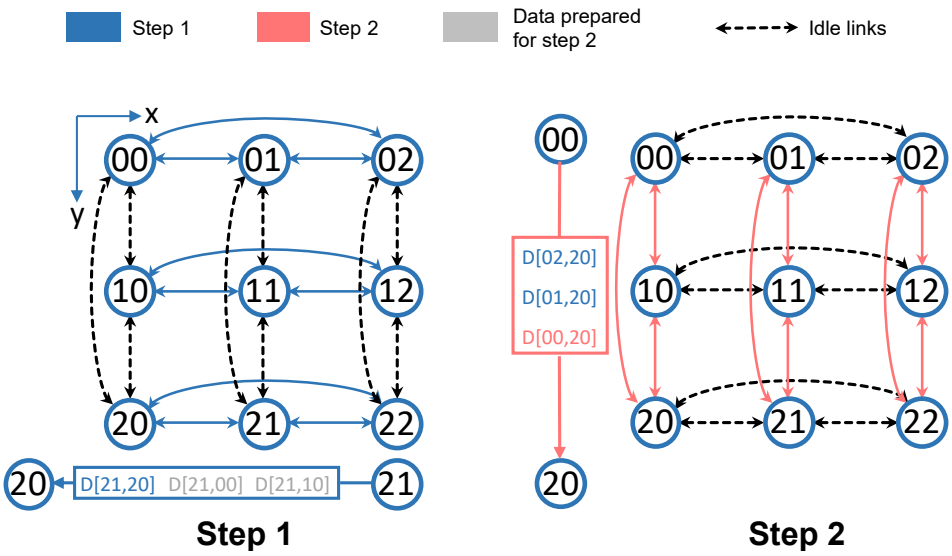


Figure 4.1: Links used by nodes in the topology in two communication steps of DO algorithm. Example of data sent from node (21) to node (20) in the first step and data sent from node (00) to node (20) in the second step.

Figure 4.1 and Algorithm 1 show the algorithm and an example of data exchange of DO algorithm on 3×3 2dfc. The algorithm starts with exchanging the data inside an arbitrary dimension (in this dissertation, we do the exchange in the x-axis first, blue links in Figure 4.1). In line 3, data appended to sending buffer SB is responsible for the data exchanged inside the fully connected group. For loop in lines 4-6 in Algorithm 1 is the preparation of data for the second step. Each node (a_2a_k) (line 7) will act as an intermediate node. It will receive m_2 chunks of data from a neighbor a_2a_z (where $a_z \neq a_k$) in the group, after the first step: one chunk $D[a_2a_z, a_2a_k]$ to store to the result, $m_2 - 1$ chunks to re-transmit in the second step.

In the second step (lines 11-20), data is exchanged in the remaining dimension. In this step, a node a_ka_1 will receive m_1 chunks of data from a sender a_za_1. One chunk $D[a_ka_1, a_za_1]$ is the data from (a_ka_1). $m_1 - 1$ chunks $D[a_kx, a_za_1]$[1] is the re-transmitted data.

DO algorithm is divided in two almost identical steps. In the first step, the algorithm exchanges the data in the first dimension. The data for the second step is also prepared and sent in this step. In the step 2, the algorithm sends the data of second dimension,

[1]where $x \in [0..m_1 - 1], x \neq a_1$. We use this notation to present a group of nodes from this point to the rest of this section.

forwards the data of the first step, and finishes the algorithm.

Figure 4.1 shows the data transfer on node (20) on 3×3 2dfc. In the first step, the blue solid links are used. Let consider an example of data transmission in node (20). In this step, nodes inside the group $(2x)$ exchange data with node (20). Node (21) and (22) sends the data $D[21, 20]$ and $D[22, 20]$ to node (20) respectively[2]. In the second step, node (00) and (10) send 6 chunks of data in the group $(0x)$ and $(1x)$ to node (20). Note that, only 2 data chunks $D[00, 20]$ and $D[10, 20]$ originate from nodes (00) and (10), while the remaining data, such as $D[0x, 20]$ and $D[1x, 20]$, originate from other nodes in the groups $(0x)$ and $(1x)$. After two communication steps, node (20) receives 6 data chunks from the other nodes in the topology and finishes the DO algorithm.

We can prove that DO is bandwidth optimal. To achieve bandwidth optimal, an algorithm must ensure two following conditions:

1. Communicate with the minimum amount of data required.

2. The network contention must be avoided.

Let take alltoall operation as an entity for proving bandwidth optimality of collective operation on 2dfc. For other algorithm, the prove takes the same idea. In a 2dfc network, there are $N = m_2 \times m_1$ nodes, where m_1 and m_2 are the sizes along the X-axis and Y-axis, respectively. Each node is addressed by a 2-tuple (a_2a_1). For an All-to-All operation, each process has N distinct data elements (one for each process, including itself), each of size b. Let's examine how DO fulfills the two conditions.

Minimum Data Transmission

Each packet of DO algorithm is sent once from the origin node. If the packet is transferred through an intermediate node (because there is no direct link), the packet only is resent once. DO algorithm proceeds in two sequential steps. First, data is exchanged within one dimension (e.g., X-axis), and then in the second step, the remaining data is exchanged in the other dimension (Y-axis). There are some data cannot reach final destination (because 2dfc is a diameter 2 network topology) will be

[2]Data for the second step are also sent in this step. For example, (21) also sends $D[21, 00]$ and $D[21, 10]$ to node (20)

forwarded to and intermediate node within the same dimension. Then, in the next step the data will be sent to the destination.

In the first step, a node (a_2a_1) sends data to $(m_1 - 1)$ neighbors in the same row ((a_2a_k) where $k \neq a_1$). The amount of data sent to each neighbor is $m_2 \times b$ bytes. The amount of $m_2 \times b$ includes:

- The original data ($1 \times b$ bytes) from node (a_2a_1) which is sent to other nodes within the same row. Example, $D[21, 20]$ in Figure 4.1.

- The data ($1 \times b$ bytes) from node (a_2a_1) which is sent to nodes in other $m_2 - 1$ rows. Example, $D[21, 00]$ and $D[21, 10]$ in Figure 4.1. This amount of data will be sent to an intermediate node. Then it will be forwarded to the final destination in the next step. In total, this amount of data equals to $(m_2 - 1) \times b$

Therefore, in the first step the total amount of data sent from each node $(m_1 - 1) \times m_2 \times b$ bytes.

In the second step, a node (a_2a_1) sends data to $(m_2 - 1)$ neighbors in the same column ((a_ka_1) where $k \neq a_2$). The amount of data sent to each neighbor is $m_1 \times b$ bytes. This data includes:

- The original data ($1 \times b$ bytes) from node (a_2a_1) directly sends to nodes in the same column. Example, $D[00, 20]$ in Figure 4.1.

- The data originated from the first step to be forwarded in this step from $m_1 - 1$ nodes. Example, $D[01, 20]$ and $D[02, 20]$ in Figure 4.1.

Therefore, in the second step the total amount of data sent from each node $(m_2 - 1) \times m_1 \times b$ bytes.

Total data sent by a node over the entire operation: The sum of data sent in both steps is: $(m_1 - 1)m_2b + (m_2 - 1)m_1b = (m_1m_2b - m_2b) + (m_1m_2b - m_1b) = 2m_1m_2b - (m_1 + m_2)b$

$$= (2m_1m_2 - (m_1 + m_2))b \tag{4.1}$$

Compare to theoretical minimum, in a 2dfc, a node (a_2a_1) has $(m_2 - 1) + (m_1 - 1)$ $= m_2 + m_1 - 2$ distance-1 neighbors, and $m_2 \times m_1 - (m_2 + m_1) + 1$ distance-2 neighbors.

Therefore, in total the amount of data to be sent is: $(m_2 + m_1 - 2) \times b + (m_2 \times m_1 - (m_2 + m_1) + 1) \times 2 \times b$

$$= (2m_1m_2 - (m_1 + m_2))b \tag{4.2}$$

Compare Equation 4.1 and Equation 4.2, DO algorithm transfer the minimum amount of data.

Contention-Free Communication

DO algorithm uses two different set of links sequentially. In the first step, the algorithm exchanges within $x - axis$ (using blue links in Figure 4.1). In the second step, the algorithm exchanges within $y - axis$ (using blue links in Figure 4.1). The nodes in each dimension are fully connected. Therefore, there is no contention.

In summary, DO algorithm satisfies two conditions of a bandwidth optimal algorithm. Hence DO is bandwidth optimal.

However we can see that DO algorithm uses two set of links sequentially. For example, Figure 4.1 shows the link usage of DO algorithm in the two steps on 3×3 2dfc. DO uses only blue links in the first step and red links in the second step. This causes resource underutilization because some links are left unused. We will solve this problem of DO algorithm in the proposed solution.

4.1.2 The Problem of MULTITREE Algorithm

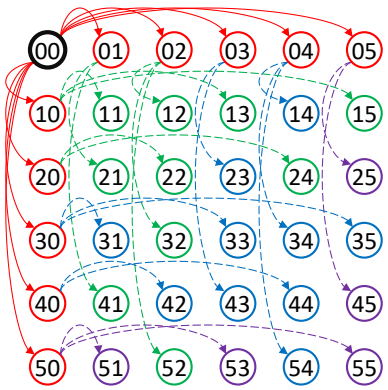
MULTITREE algorithm [75] is a bandwidth optimal, low-latency algorithm, originally proposed for allreduce operation.

The original algorithm creates N spanning trees $T_i, i \in [0..N-1]$ for a N-node directed $G(V, E)$ network topology. Balance trees with global bandwidth coordination are ensured by building the trees in parallel. The information of the structure of the tree T_i is used to schedule the communication in node i. The co-design of communication and design made the algorithm collision-free and bandwidth optimal.

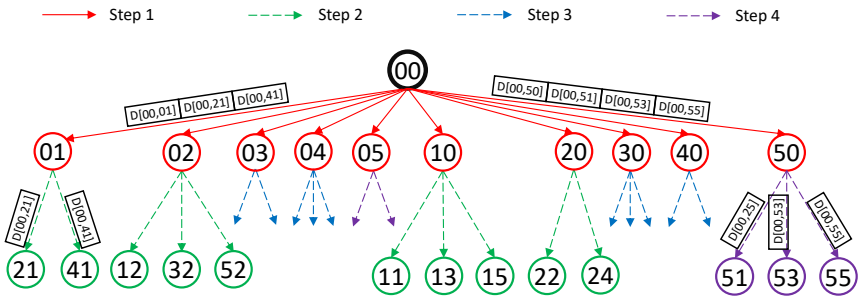
Tree building algorithm also proposes a way to support All-to-All algorithm as in [94, 95]. The adding method makes the schedule trees denser to the leaves leading to reduction of fan-out at higher levels, enabling more efficient resource utilization and lower latency.

The main takeaway of MULTITREE algorithm is that in each communication step, the links of the topology must be used as much as possible. Based on this insight, we can flexibly change the shape of the trees. We do not have to follow the instruction from the original paper: Each tree level is a communication step. In stead of using tree level, we use marks that describe the link to be used in a specific communication step. In this section, we would like to provide a specific way to apply MULTITREE algorithm to 2dfc (in addition to the general method of MULTITREE algorithm in Section 2.2.3).

In 2dfc, each node connects to $m_2 + m_1 - 2$ nodes in two dimensions. Therefore, the optimal way to built the schedule trees is adding $m_2 + m_1 - 2$ each communication step. This can be easily explained, as each node uses all of its connections. Because building building the schedule trees for MULTITREE algorithm is not the main contribution of this dissertation so we will give the construction of schedule trees of MULTITREE algorithm by an example (instead of formal algorithm).



(a) Connection of schedule tree from node $(a_2a_1) = (00)$.



(b) Shape of the schedule tree from node $(a_2a_1) = (10)$.

Figure 4.2: Schedule tree from node $(a_2a_1) = (00)$ on a 6×6 2dfc. Figure 4.2a shows the connection from root node $(0, 0)$. The physical connection of the topology is omitted for the sake of clarity. The colors of links and nodes show the order of communication steps. Figure 4.2b shows the shape of the schedule tree of node (00) and the data flow from the root to branches (01) and (50).

Figure 4.2 shows an example of building a schedule tree from root node (00) on 6×6 2dfc. Building schedule of each root node starts with adding all the neighbor nodes in $x-$ axis and $y-$ axis. From step two to numstep (see Equation 4.4), children of two nodes from $x - axis$ and two nodes from $y - axis$ are added to the schedule trees. Because two nodes (in the same axis) are chosen, there is one node in the odd position and one node in the even position. The children are added using the following method:

- For nodes along the y-axis: If the parent is in an odd position, its children are all nodes in the same row at odd positions. If the parent is in an even position, its children are all nodes in the same row at even positions.

- For nodes along the x-axis: If the parent is in an odd position, its children are nodes in the same column at even positions. If the parent is in an even position, its children are in the same column at odd positions.

For example, in the step one, all the the nodes shown in red are added to the tree. The nodes from the $x - axix$ includes: (01) (02) (03) (04) (05). Nodes from the $y - axis$ includes: (10) (20) (30) (40) (50). From step two to step four, in every step, children of two nodes from the $x - axis$ and two nodes from the $y - axis$ is added until there are no remaining nodes. For example, in step two, nodes (01) (02) and nodes (10) (20) are

chosen. Then the children are added using the adding method. In detail, node (10) is in the odd position from root node (00) so node in the same row with the odd positions are added: (11) (13) (15). Node (20) is in the even position from the root node (00) so nodes in the even positions in the same row will be added: (22) (24)

Building the schedule with this method holds the bandwidth optimal property of MULTITREE algorithm, but simplify the building of schedule trees. In the experiment, we have tried to build the tree follow the method using in the original paper, but we find the trees with higher number of step than by using my method.

Analysis of MULTITREE algorithm on 2dfc

Consider a $G(V, E)$ is a 2dfc network. V and E[3] are the sets of vertices and edges, respectively. $|V| = N = m_1 \times m_2$ corresponds to the size of the vertex set where m_1 and m_2 are the sizes of two dimensions. $|E|$ corresponds to the size of the edge set.

We have:

- $|V| = N = m_1 \times m_2$. m_1 and m_2 are the sizes of two dimensions.
- $|E| = m_1 \times m_2 \times (m_1 + m_2 - 2)$ [4]
- A node $A \in V(G)$ is indexed by a 2-tuples $A = (a_2 a_1), 0 \leq a_2 < m_2, 0 \leq a_1 < m_1$

The target of the MULTITREE algorithm is to build a set $T \ni T_i$ of N spanning trees. A tree $T_i \in T$ is differed by a root node $v \in V(G)$. $|T|$ is the total number of uni-directional links used to build N trees. $|T| = \sum |T_i|$ in which $|T_i|$ is the number of links in the tree T_i. It is easy to see that $|T_i| = N - 1$ because it needs $N - 1$ uni-directional links to connect N nodes. MULTITREE algorithm requires N trees; therefore, the number of links needed for N trees is:

$$|T| = N \times (N - 1) = m_1 m_2 \times (m_1 m_2 - 1) \tag{4.3}$$

Therefore, the least number of times topology G needed to be reset to fill in T is

[3] Edge $e \in E$ is bi-directional; $|E| = m_1 \times m_2 \times (m_1 + m_2 - 2)$.

[4] 2-fold the number of links described in Equation 2.9 because MULTITREE algorithm considers bi-directional links.

$|T|/(|E|)$

$$\begin{aligned} \text{numstep}_{min} &= \left\lceil \frac{|T|}{|E|} \right\rceil = \left\lceil \frac{m_1 m_2 \times (m_1 m_2 - 1)}{m_1 m_2 \times (m_1 + m_2 - 2)} \right\rceil \\ &= \left\lceil \frac{m_1 m_2 - 1}{m_1 + m_2 - 2} \right\rceil \end{aligned} \tag{4.4}$$

Equation 4.4 represents the lower bound for the number of steps in the MULTITREE algorithm. In order to achieve this bound, a tree T_i has to add $\frac{|E|}{N} = m_1 + m_2 - 2$ nodes in each time step, where $|E|$ is the number of links that need to be added in a step for all trees and N is the number of trees. Note that $m_1 + m_2 - 2$ is the switch radix. This means that, in each communication step, the number of added nodes to a tree T_i must equal to the switch radix[5] to achieve the lower bound of the number of communication steps.

MULTITREE algorithm fully utilizes the resource of 2dfc but the number of communication steps becomes larger than the DO algorithm. This poses a performance problem for the communication with small-size messages or in a network with a large startup time $\boldsymbol{\alpha}$.

4.2 Multi-Dimensional Algorithm

In this section, we proposed an algorithm called: Multi-Dimension (MD) algorithm. The algorithm is based on the idea that we need to fully utilize the resources of the topology while maintaining a small number of communication steps. DO uses two different sets of links in two different communication steps serially. The order of which of the first dimension does not affect the result because 2dfc is symmetric. Therefore, we will utilize the two sets of links in parallel.

[5]In a regular network, where each node has the same number of neighbors, the number of added nodes in each communication step is equal.

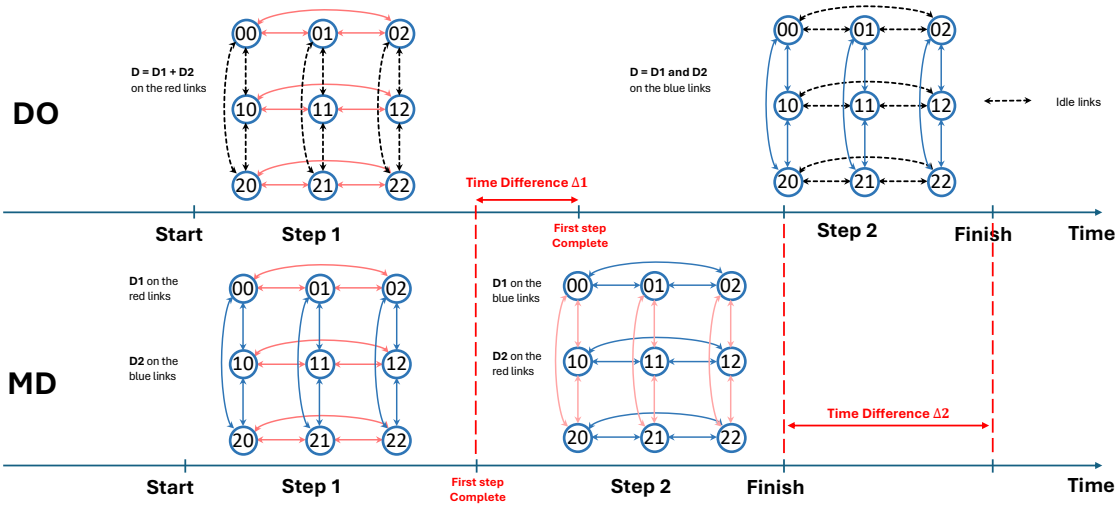


Figure 4.3: Status of links in each step of DO algorithm for a collective operation. For a specific operation the data exchange will be different.

The idea can be visualized as in the timeline Figure 4.3. DO use two sets of links (blue and red) consequently. On the other hand, MD divides the data into two parts and uses all the links of the topology at the same time. Due to the use of more network resources and ensuring no contention, MD can save $\Delta 1$ time unit for the first step and $\Delta 2$ time unit for the overall process (see Figure 4.3). MD can be regarded as an extension of the proposed collective communication in the previous chapter.

Let use the linear cost model (Section 2.4) to examine the communication time. In the first step of DO algorithm, a node sends $m_2 \times b$ bytes of data (see Section 4.1.1) to a node in the same dimension (x-axis). Hence, time for the first step $t_1 = \alpha + m_2 \times b \times \beta$. In the second step, the amount of data sent to a neighbor node is m_2. Therefore the communication time of the second step $t_2 = \alpha + m_1 \times b \times \beta$. Total communication time for a DO algorithm

$$t_{DO} = t_1 + t_2 = 2\alpha + (m_1 + m_2) \times b \times \beta. \tag{4.5}$$

In the MD algorithm, data array D is split into two parts D_1 and D_2 to send to two dimensions simultaneously. Data in D_1 is sent to nodes in the x-axis first, then exchanged with nodes in the y-axis. Data in D_2 is exchanged in the reverse direction y-axis first then x-axis. Let x be the ratio of data in D that is used in D_1. Hence, the rate of data in D_2 is equal to $1 - x$.

Table 4.1: Communication time of MD algorithm.

	D_1	D_2
Step 1	$t_{D_1,1} = \alpha + xbm_2\beta$	$t_{D_1,2} = \alpha + (1-x)bm_1\beta$
Step 2	$t_{D_2,1} = \alpha + xbm_1\beta$	$t_{D_2,2} = \alpha + (1-x)bm_2\beta$

Table 4.1 shows the communication time in each step of MD algorithm. The time for sending D_1 and D_2 in two paths must be balanced to ensure load balancing and reduce waiting time. In this work, we simply align the communication time of the first step of D_1 and D_2. By solving the equation $t_{D_1,1} = t_{D_2,1}$, we get $x = \frac{m_2}{m_2+m_1}$. Therefore, the total communication time for MD algorithm t_{MD}:

$$t_{MD} = \begin{cases} t_{D_1,1} + t_{D_1,2} = 2\alpha + m_2 b\beta & ; m_2 \geqslant m_1 \\ t_{D_2,1} + t_{D_2,2} = 2\alpha + m_1 b\beta & ; m_1 > m_2 \end{cases} \tag{4.6}$$

The alignment in the first step causes a timing discrepancy t_{dics} in the second step. The timing discrepancy equals to $\left|t_{D_1,2} - t_{D_2,2}\right|$:

$$\begin{aligned} t_{dics} &= \left|t_{D_1,2} - t_{D_2,2}\right| \overset{x=\frac{m_2}{m_2+m_1}}{=} \left|\frac{m_1^2}{m_1+m_2} - \frac{m_2^2}{m_1+m_2}\right| b\beta \\ &= |m_2 - m_1|\, b\beta \end{aligned} \tag{4.7}$$

t_{dics} is the time that the data between two paths wait for each others to complete because of the difference in resource arrangement in each path. However, t_{dics} is not big compared to total communication time. t_{dics} becomes bigger when the difference in size of two dimensions is bigger. In case size of two dimension are equal, $m_2 = m_1$, t_{dics} equals zero.

Algorithm 2: Proposed Multi-Dimension **(MD)** All-to-All Collective Communication on 2lvfc

Data: Data Array D has N blocks of data, each block is of size b
Result: Each process receives data from all other processes
Input: Topology 2lvfc(m_2, m_1), process (a_2a_1)

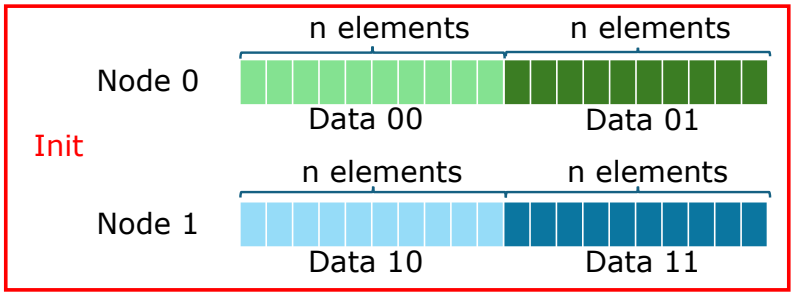
$x \leftarrow \frac{m_2}{m_2+m_1}$; // Calculate percentage of data in each path
$(D_1, D_2) = \text{split}(D, x)$; // Split data for 2 paths
// First step:
for $k \leftarrow 1$ **to** $m_1 - 1$ **do**
  ; // X-axis first
  $a_k \leftarrow (a_1 + k) \mod m_1$; // (a_2a_k) is the receiver
  SB1 appends $D_1[a_2a_1, a_2a_k]$; // init sendbuf SB1
  for $z \leftarrow 1$ **to** $m_2 - 1$ **do**
    $a_z \leftarrow (a_1 + z) \mod m_1$
    SB1 $\leftarrow$ SB1 appends $D_1[a_2a_1, a_2a_z]$
  Non-blocking send SB1 to node (a_2a_k)
for $k \leftarrow 1$ **to** $m_2 - 1$ **do**
  ; // Y-axis first
  $a_k \leftarrow (a_2 + k) \mod m_2$; // (a_ka_1) is the receiver
  SB2 appends $D_2[a_2a_1, a_ka_1]$; // init sendbuf SB2
  for $z \leftarrow 1$ **to** $m_1 - 1$ **do**
    $a_z \leftarrow (a_2 + z) \mod m_2$
    SB2 $\leftarrow$ SB2 appends $D_2[a_2a_1, a_za_1]$
  Non-blocking send SB2 to node (a_2a_k)
for $z \leftarrow 1$ **to** $m_1 - 1$ **do**
  $a_z \leftarrow (a_1 + z) \mod m_1$; // (a_2a_z) is the sender
  Non-blocking receive from node (a_2a_z)
for $z \leftarrow 1$ **to** $m_2 - 1$ **do**
  $a_z \leftarrow (a_2 + z) \mod m_2$; // (a_za_1) is the sender
  Non-blocking receive from node (a_za_1)
Wait for all sends and receives to complete.
// Second step:
for $k \leftarrow 1$ **to** $m_2 - 1$ **do**
  ; // X-axis first
  $a_k \leftarrow (a_2 + k) \mod m_2$; // (a_ka_1) is the receiver
  SB1 appends $D1[a_2a_1, a_ka_1]$; // init sendbuf SB1
  for $z \leftarrow 0$ **to** $m_1 - 1$ **do**
    $a_z \leftarrow (a_1 + z) \mod m_1$
    SB1 $\leftarrow$ SB1 appends $D1[a_2a_z, a_ka_1]$; // $D1[a_2a_z, a_ka_1]$ received from the first step
  Non-blocking send SB1 to node a_ka_1
for $k \leftarrow 1$ **to** $m_1 - 1$ **do**
  ; // Y-axis first
  $a_k \leftarrow (a_1 + k) \mod m_1$; // (a_2a_k) is the receiver
  SB2 appends $D2[a_2a_1, a_2a_k]$; // init sendbuf SB2
  for $z \leftarrow 0$ **to** $m_2 - 1$ **do**
    $a_z \leftarrow (a_2 + z) \mod m_2$
    SB2 $\leftarrow$ SB2 appends $D2[a_za_1, a_2a_k]$; // $D2[a_za_1, a_2a_k]$ received from the first step
  Non-blocking send SB2 to node a_2a_k
for $z \leftarrow 1$ **to** $m_2 - 1$ **do**
  $a_z \leftarrow (a_2 + z) \mod m_2$; // (a_za_1) is the sender
  Non-blocking receive from node (a_za_1)
for $z \leftarrow 1$ **to** $m_1 - 1$ **do**
  $a_z \leftarrow (a_1 + z) \mod m_1$; // (a_2a_z) is the sender
  Non-blocking receive from node (a_2a_z)
Wait for all sends and receives to complete.

Algorithm 2 shows the MD algorithm. MD algorithm can be summarized as follows:

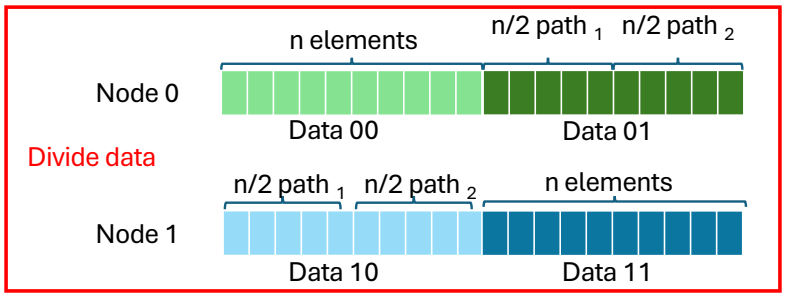
1. Calculate the percentage of data in each path $x = \frac{m_2}{m_2+m_1}$.
2. Split data array D into two arrays D_1 and D_2 by using a split function.
3. First step: Non-blocking transfer D_1 and D_2 through two paths simultaneously. D_1 uses the links in the x-axis first (red links). D_2 uses links in the y-axis first (blue links). In this step, we can imagine that 2 DO algorithms work simultaneously.
4. Wait for non-blocking the transfers in the first step to complete.
5. Second step: non-blocking transfer data D_1 and D_2 in the links in the remaining axis.
6. Wait for the transfer in the second step to complete.

In the implementation, we can use an offset k to calculate the index of data elements inside D without copying. We use D_1 and D_2 for the presentation purpose.

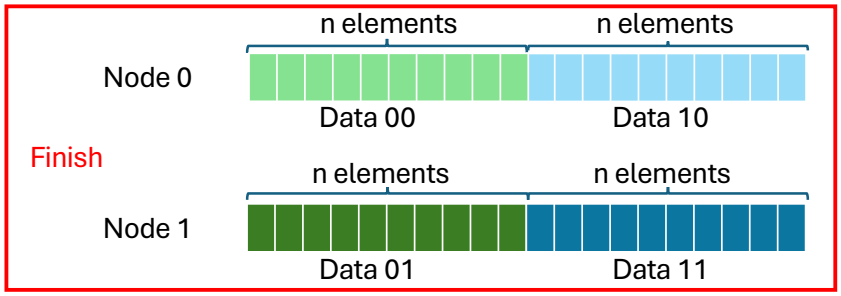
(a) 1×2 fully connected.



(a) Init.



(a) Divide data.



(b) Finish.

Figure 4.6: The alltoall operation using MD algorithm on a 1×2 2dfc.

Note that in the step of splitting data (2), each chunk of data $D[i, j]$ in D is split into two parts $D_1[i, j]$ and $D_2[i, j]$. The work of splitting data of two processes is illustrated in Figure 4.6. The operation begins with the preparation of the different data for the transmission. In Figure 4.5a, $Data01$ on node 0 will be sent to node 1. $Data10$ on node 1 will be sent to node 0. In the next step, the data to be sent ($Data01$ and $Data10$) are divided into two parts. If the number of elements is not divided by two, the later part will take the remainder. Then, in the next step, two parts of data will be sent in two directions simultaneously. Note that for a larger number of nodes, the data always be divided into two parts.

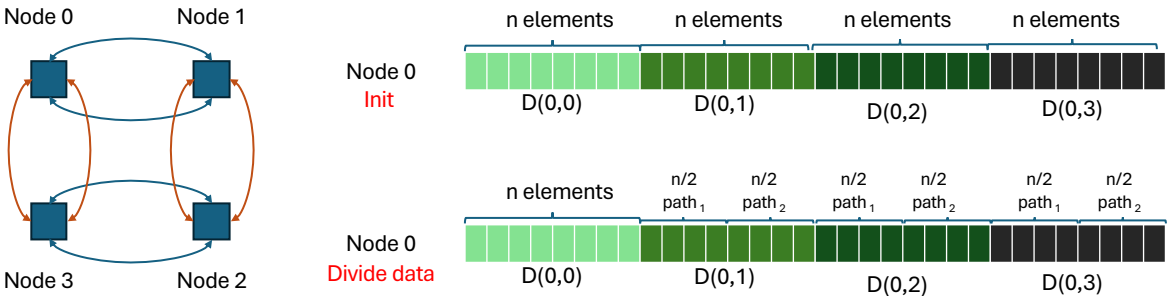


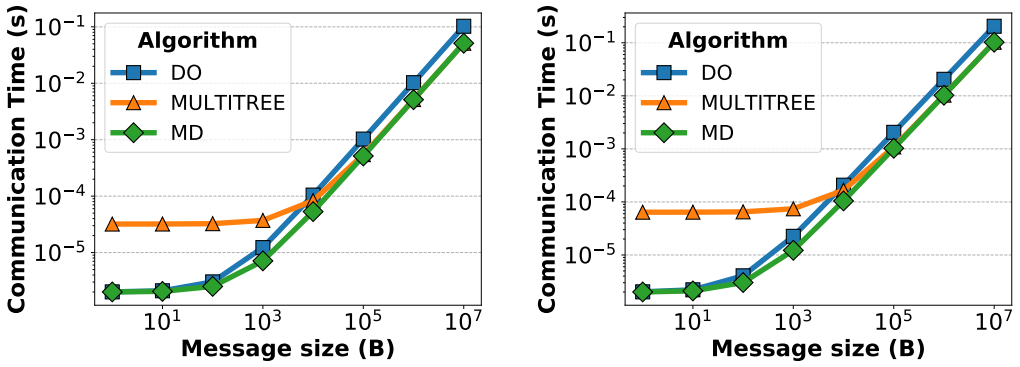
Figure 4.7: Data partition of 2×2 2dfc for MD algorithm

Figure 4.7 shows the data partition of MD algorithm for 4-node 2×2 2dfc. Because $Data01$, $Data02$ and $Data03$ are sent to node 1, node 2 and node 3, respectively, they are divided into two parts.

Multi-Dimension algorithm sends the data through two dimensions at the same time. In each dimension, half the amount of data is sent separately inside a fully connected group. Because data is sent by two paths simultaneously, the communication time is expected to be reduced by the percentage of the number of ports increased. In addition, by sending the data through two sets of links independently, the bandwidth optimality is reserved in MD.

4.3 Performance Evaluation

In this section, we evaluate the performance of our proposed Multi-Dimension collective communication algorithms (named **MD**) for various 2dfc network topologies and settings. The result shows that our algorithm is able to achieve shorter communication time compared with other topology-aware algorithms optimized for 2dfc networks such as Dimension Order (**DO**)), and tree-based algorithms (MULTITREE [75]). We also consider the other conventional algorithms in production libraries such as Bruck [78] and Pairwise algorithm as the baseline collective communication algorithms (more detail in Section 2.2.3). We use **Bruck** and **Pair** when referring to those algorithms, respectively. In the following, we first theoretically estimate the communication cost of the MD algorithm using the cost model mentioned in Section 2.4. We also compare it with the cost **DO** and **MULTITREE** algorithms in Section 4.3.1. We then report the compare the simulation performance of our algorithms with the others in Section 4.3.2.



(a) 64×64 2dfc **(b)** 128×128 2dfc

Figure 4.8: Estimated communication time of alltoall algorithms.

4.3.1 Numerical Analysis

Without loss of generality, in a $m_2 \times m_1$ 2dfc, assume that $m_2 \geqslant m_1$, and m_2 is divisible by two. The cost for DO can be obtained from Equation 4.5. Equation 4.6 shows the communication time of the MD algorithm in the general case. When $m_2 \geqslant m_1$ the communication time t_{MD} is dominant by $t_{D_1,1} + t_{D_1,2}$. Therefore $t_{MD} = 2\alpha + m_2 b\beta$.

In the case of the MULTITREE algorithm, the first step takes the longest time because the root node has to send the data for the subsequent steps, i.e., $t_{MT,1^{st}} = \alpha + \left(\left\lceil \frac{m_2}{2} \right\rceil + 1\right) b\beta$ where $\lceil \frac{m_2}{2} \rceil$ is the largest number of chunks the root node has to send to its children. From the second step to the step $\frac{m_2}{2}$th (see Equation 4.4), only one chunk of data is transferred to leave nodes. Hence, the time cost is equal to:

$$t_{MT,2^{nd} \to (\frac{m_2}{2})^{th}} = \sum_{2}^{\frac{m_2}{2}} (\alpha + b\beta) = \left(\frac{m_2}{2} - 1\right)(\alpha + b\beta) \tag{4.8}$$

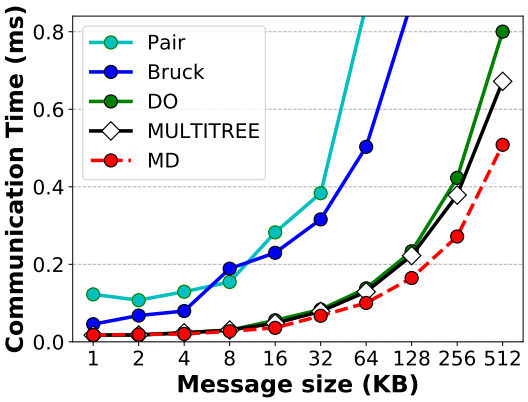
Table 4.2: Communication cost of topology-aware All-to-All communication algorithms on a $m_1 \times m_2$ 2dfc network.

Algorithm	Cost
DO	$t_{DO} = 2\alpha + (m_1 + m_2)b\beta$
MULTITREE	$t_{MT} = \frac{m_2}{2}\alpha + m_2 b\beta$
MD (Ours)	$t_{MD} = 2\alpha + m_2 b\beta$

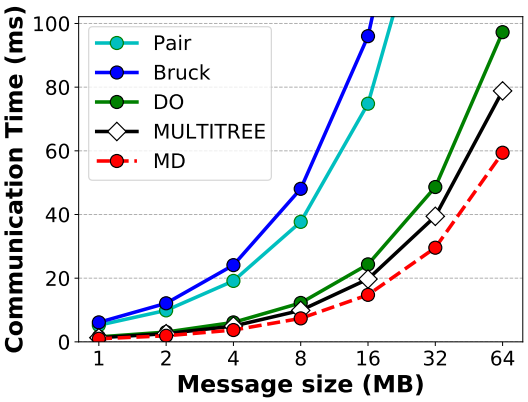
Table 4.2 summarizes the communication cost of the three algorithms. It is worth noting that the coefficient related to α reflects the start-up time used for the communication. The bigger the coefficient of α the bigger the start-up time due to the bigger communication step. Besides, the coefficient related to β represents the time for transferring data. Figure 4.8(a) and Figure 4.8(b) show the estimated communication time of All-to-All communication versus the message size b. We set α equal to $1\mu s$ (microsecond). The value of β is set to 8×10^{-11}, which is equivalent to a bandwidth of 100 Gbps. Compared to the DO algorithm, MULTITREE has a higher startup time (2α compared to $\frac{m_2}{2}\alpha$). However, the time for sending data of the DO algorithm is bigger ($(m_1 + m_2)b\beta$ compared to $m_2 b\beta$) due to resource underutilization of DO. This means that DO is good for All-to-All communication with small data sizes while the MULTITREE algorithm is better for bigger data sizes. In the case of the MD algorithm, it achieves the best communication time because it fully utilizes the resource of 2dfc with a small number of communication steps.

4.3.2 Simulation Results

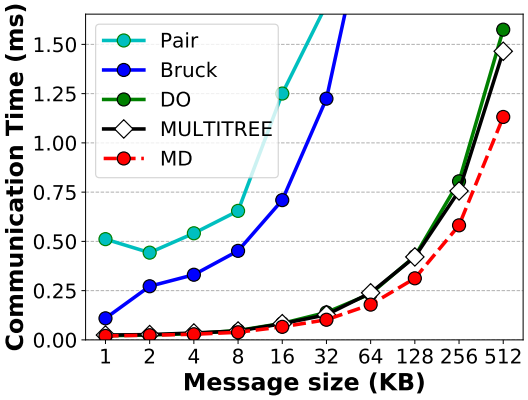
In this section, we use Simgrid framework version 3.28 [87] to evaluate the performance of MD and other collective communication algorithms. In the evaluation, we consider three different 2dfc network sizes N including 16 (4×4 2dfc), 64 (8×8 2dfc), and 256 (16×16 2dfc). We aim to estimate the communication time of the algorithms with different message sizes b ranging from 1KB to 64MB, as common appears in Deep Learning and big data workloads [5]. The total buffer size of a process in this case is $b \times N$ where N is the number of processes. As a result, the total buffer size varies from $4 \times 4 \times 1 = 16$ KB to $16 \times 16 \times 64 = 16384$ MB. The startup latency for each communication is set to 1μs. Link bandwidth is set to 100 Gbps. We run the simulation on a host system with SMP Debian 3.16.64-2 operating system on Intel(R) Xeon(R) CPU E5-2667 v4 and 794 GB of RAM.



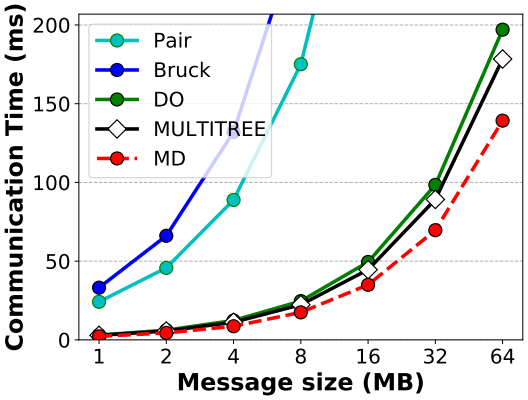
(a) 4×4 2dfc small data size



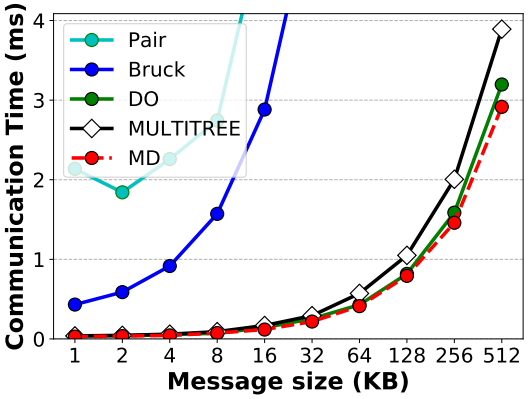
(b) 4×4 2dfc big data size



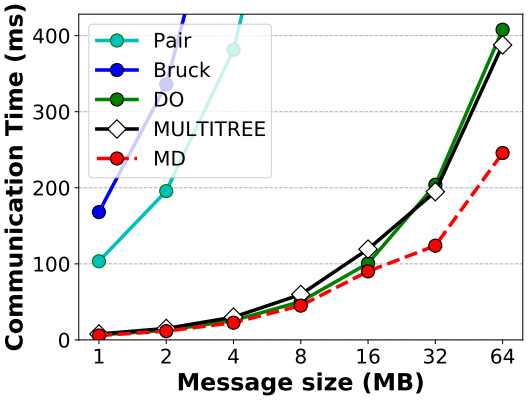
(c) 8×8 2dfc small data size



(d) 8×8 2dfc big data size



(e) 16×16 2dfc small data size



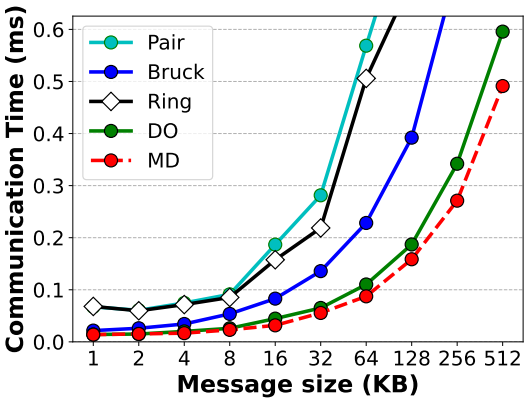
(f) 16×16 2dfc big data size

Figure 4.9: Communication time of micro alltoall benchmark versus message size on 4×4, 8×8 and 16×16 2dfc.

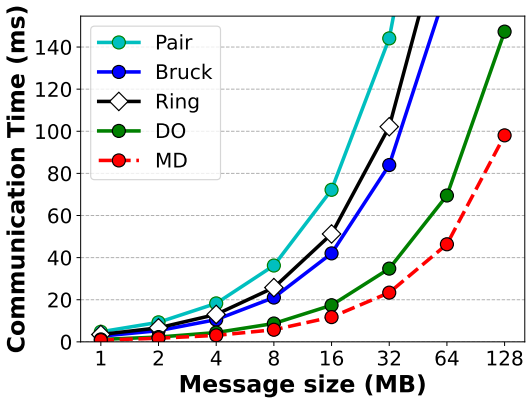
Figure 4.9 shows the simulated communication time in milliseconds when changing

the message size b. Firstly, the result shows that Bruck is better than the Pair algorithm for small-size messages (Figure 4.9a, 4.9c, and 4.9e). On the contrary, the Pair algorithm is faster than the Bruck algorithm for large-size messages (Figure 4.9b, 4.9d, and 4.9f). Reconfirm the related work in Section 2.2.3 and validates the result obtained from our simulation. Secondly, the topology-aware algorithms, i.e., DO, MULTITREE, and MD, gain a huge advantage over conventional algorithms, i.e., Bruck and Pair. For example, DO, MULTITREE, and MD are 8.78×, 9.60×, 11.98× faster than Bruck in the case of 8×8 2dfc network and 32KB message size. This is quite obvious because topology-aware uses the topology information better which reduces the amount of transferring data and avoids collision.

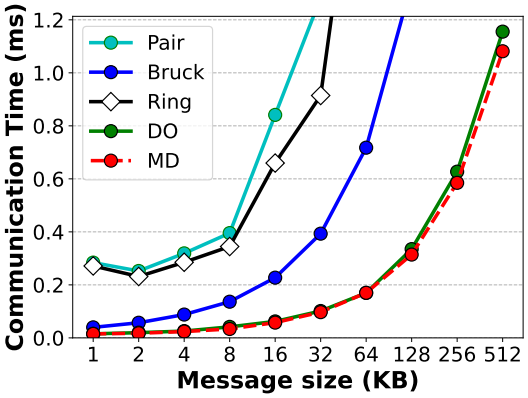
Considering the three topology-aware algorithms, our MD algorithm is the best in all of the cases. Specifically, the total communication time of MD is faster than the MULTITREE and DO algorithms up to 1.58×, and 1.66× on 16×16 2dfc, respectively. This result again confirms our finding in Section 4.3.1. That is, MD requires both the smallest latency coefficient (related to α) and the smallest bandwidth coefficient (related to β) as shown in Table 4.2. The MULTITREE algorithm is faster than DO in most cases, especially when the message size is bigger. It is because in these cases, the latency coefficient dominates mainly the total communication time. By contrast, when the message size is small, the latency coefficient becomes the main component of the total communication time. In this case, the MULTITREE algorithm outperforms the DO algorithm. In addition, in the case of larger network size, i.e., larger m_2, m_1, the impact of the latency coefficient, i.e. $\frac{m_2}{2}$, to the total communication of the MULTITREE becomes more significant and mitigates the impact of bandwidth coefficient.



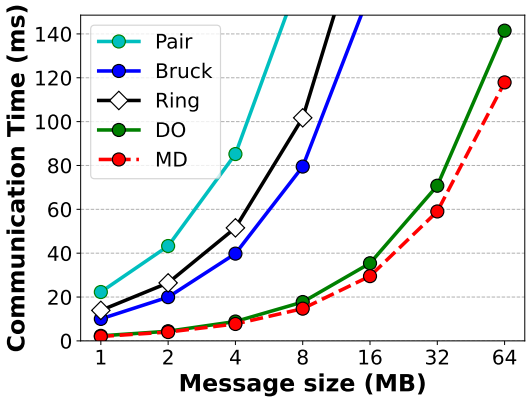
(a) 4×4 2dfc small data size



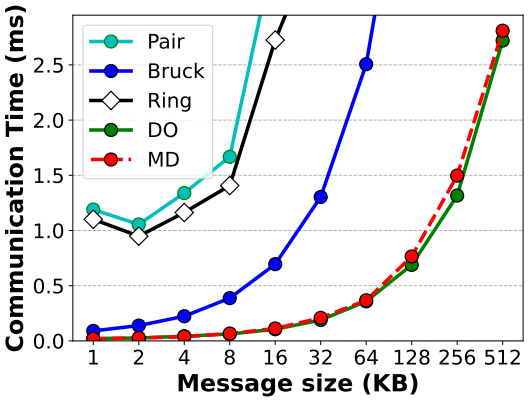
(b) 4×4 2dfc big data size



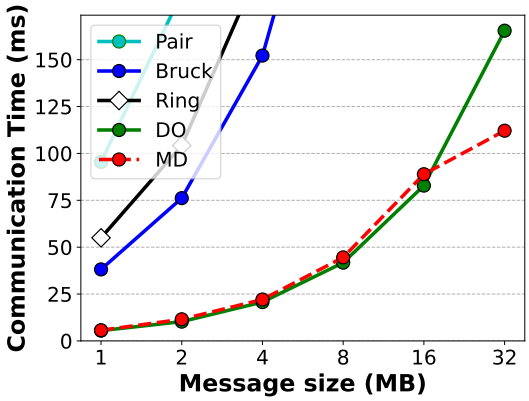
(c) 8×8 2dfc small data size



(d) 8×8 2dfc big data size



(e) 16×16 2dfc small data size



(f) 16×16 2dfc big data size

Figure 4.10: Communication time of micro allgather benchmark versus message size on 4×4, 8×8 and 16×16 2dfc.

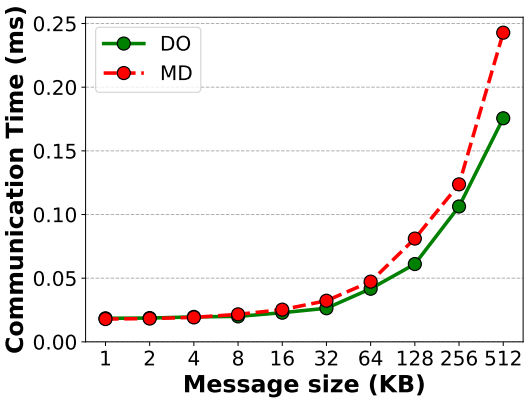
Figure 4.10 shows the simulated communication time for the allgather operation

across various message sizes and topology scales. Five algorithms to compare include: Pairwise (Pair), Bruck, Ring, Dimension Order (DO), and our proposed Multi-Dimension (MD). For small message sizes, the Bruck algorithm consistently outperforms the Pair and Ring algorithms, especially on the 4×4 and 8×8 topologies. For example, with data size 1 MB, Bruck achieves 2.7 ms, which is faster than both Pair (4.8 ms) and Ring (3.5 ms). Compared to Figure 4.9, Bruck algorithm shows an advantage over other conventional algorithms. Unlike alltoall Bruck, allgather Bruck sends the same data for other processes in the communicator, therefore, it saves the bandwidth cost. In addition, data in Bruck allgather arrive in the right place on the result buffer. It does not require data reshuffling, hence reduces the communication time even if data size increases.

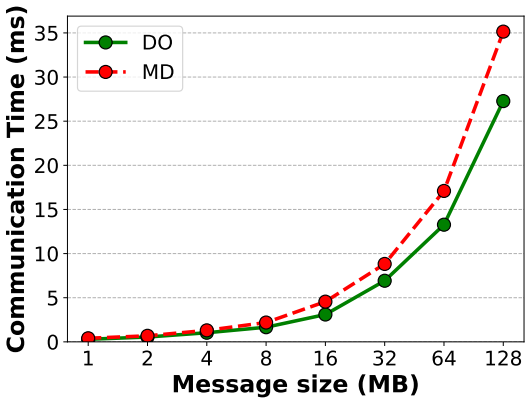
For topology-aware algorithms, DO and MD show substantial performance gains over conventional algorithms (Bruck, Pair and Ring) across all size of message. For example, on a 16×16 topology with data size 4 MB, MD algorithm finishes in 44.5 ms, which is 6.83× faster than Bruck and 16.3× faster than Pair. These improvements result from exploiting topology information, reducing data movement, and eliminating contention.

Within the group of topology-aware algorithms, MD outperforms DO for small (4×3) and medium (8×8) network topology, regardless of message size. The advantage comes from more efficient utilization of topology information. MD achieves up to 1.5× and 1.2× faster performance compared to DO on 4×4 and 8×8 respectively. For large topology (16×16), the performance of MD and DO is nearly the same (DO a bit better), because the overhead of data partitioning and connection initialization becomes comparable to the algorithmic advantages. However, when the message size increases to 32MB, MD outperforms DO again in 16×16 network topology. Overall, while DO and MD reduce communication steps along one dimension, MD more effectively leverages topology information to exploit available bandwidth.

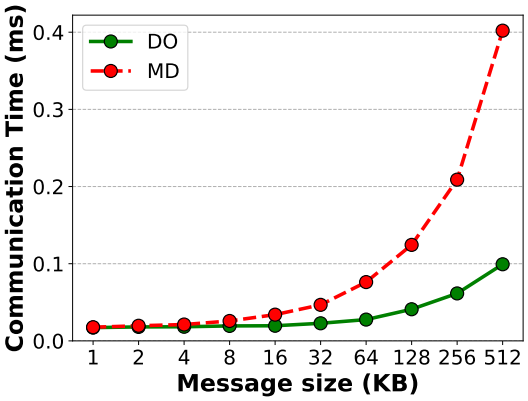
In conclusion, conventional algorithms neglect network-topology information, resulting in poor performance. In contrast, MD efficiently utilizes the resources of the 2dfc network topology with minimal latency overhead, providing superior performance across nearly all tests.



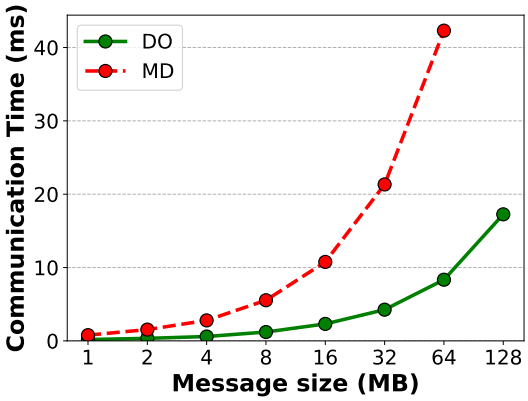
(a) 4×4 2dfc small data size



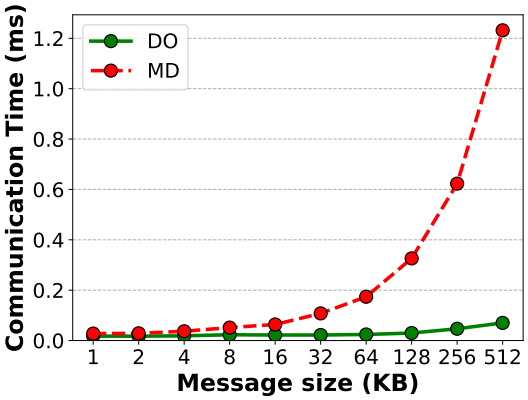
(b) 4×4 2dfc big data size



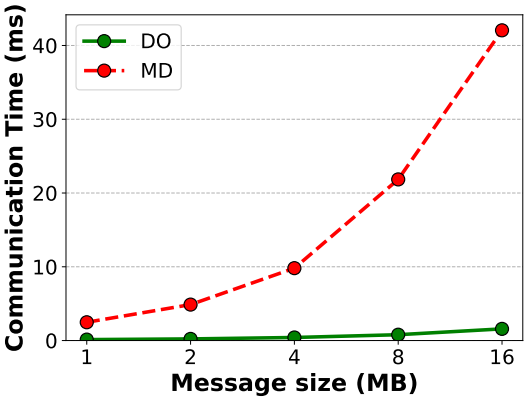
(c) 8×8 2dfc small data size



(d) 8×8 2dfc big data size



(e) 16×16 2dfc small data size



(f) 16×16 2dfc big data size

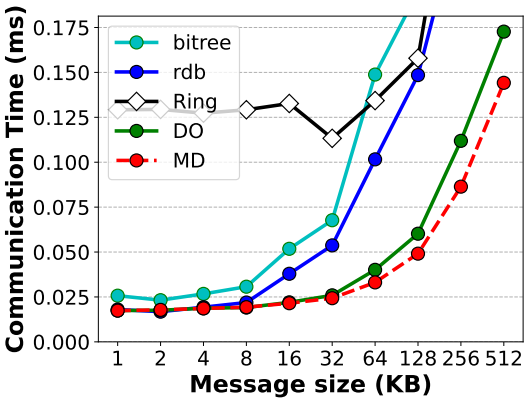
Figure 4.11: Communication time of micro allreduce benchmark versus message size on 4×4, 8×8 and 16×16 2dfc.

In the next part of this section, we will examine the performance of the MD algorithm for the allreduce operation. Figure 4.11 shows the comparison between the DO and MD algorithms. The results show that the performance of DO is better than MD in almost all the experiments, except for the data size $<$ 8KB on the 4×4 2dfc.

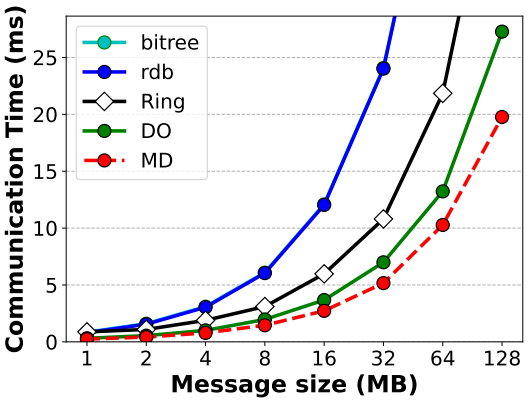
The allreduce operation is comprised of a reduce-scatter followed by an allgather. The performance of allgather using MD is usually better than DO (see Figure 4.10). Therefore, the performance disadvantage of MD compared to DO is due to the reduce-scatter operation.

Reduce-scatter is an aggregate operation (see Figure 2.7f). In other words, the data size decreases after the operation. This is the reason why the algorithm' s advantage is not present in this operation: the idea of MD is partitioning, while the target of reduce-scatter is merging. Applying a different target to the operation leads to a decrease in performance.

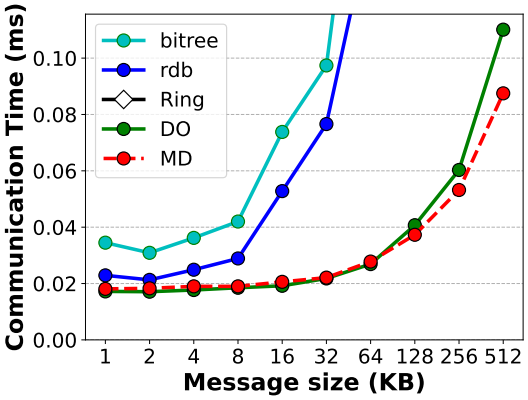
So, in this work, we suggest using DO for reduce-scatter and MD for allgather in the allreduce operation.



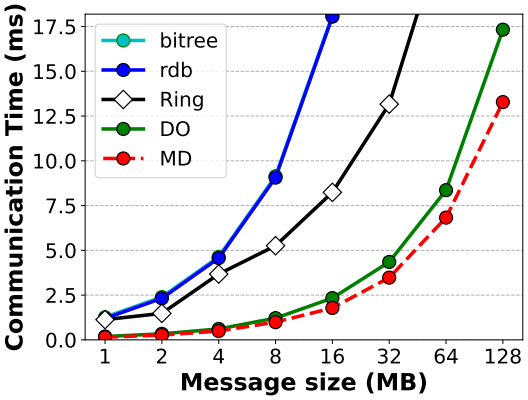
(a) 4×4 2dfc small data size



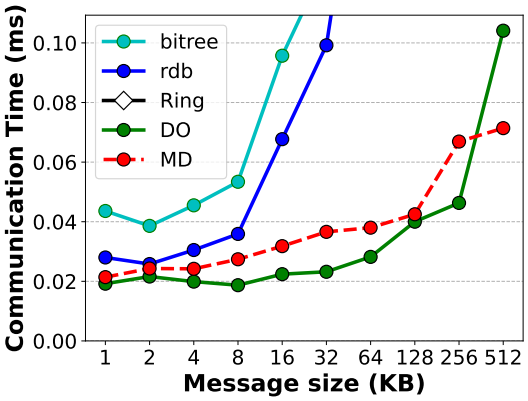
(b) 4×4 2dfc big data size



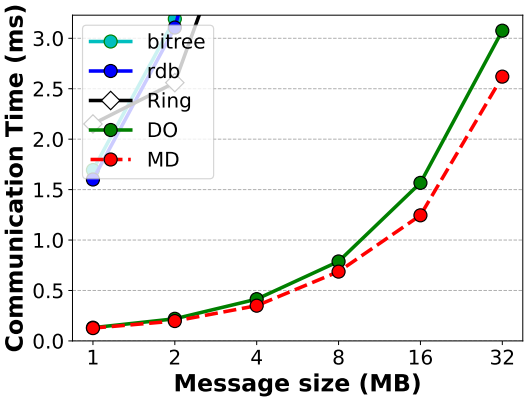
(c) 8×8 2dfc small data size



(d) 8×8 2dfc big data size



(e) 16×16 2dfc small data size



(f) 16×16 2dfc big data size

Figure 4.12: Communication time of micro allreduce benchmark versus message size on 4×4, 8×8 and 16×16 2dfc after modifying the algorithm.

Figure 4.12 shows the simulated communication time of the allreduce operation

after modifying its reduce-scatter phase to use DO. The experiment is conducted across different message sizes and topology scales. The results demonstrate that this modification improves the performance of the allreduce operation. A detailed analysis is provided below. The five algorithms compared are Double Binary Tree (bitree) [68], Recursive Doubling (rdb) [64], Ring Allreduce (ring), DO and MD. Among these, bitree, rdb and ring are conventional algorithms commonly used in production MPI library and nd frequently evaluated in research studies. The bitree algorithm schedules the steps of allreduce by constructing two trees based on even and odd steps, ensuring that a node never sends or receives the data simultaneously in both trees. However, bitree suffers from a mismatch between the logical and physical network topology, which leads to contention during execution. The rdb reduces the latency through recursive distance doubling in the reduce-scatter phase, followed by distance halving in allgather phase. Nevertheless, rdb faces the same limitation as bitree, it ignores the underlying network-topology information. As a result, its performance degrades significantly for larger message sizes.

Among conventional algorithms, for small message size, rdb continuously outperforms ring and bitree due to its logarithmic step complexity. For example, with data size 8 KB, rdb achieves 35.9 μs which is faster than both ring (2.06 ms) and bitree (53.4μs) on 16×16 2dfc network topology. However, as the data size increases, ring demonstrates its advantage because of bandwidth optimality. From 2 MB onward, ring outperforms both rdb and bitree across all topologies. At 8 MB, ring achieves 13.1 ms, which is 2.74× faster than both bitree and rdb.

Among the group of topology-aware algorithms, MD outperforms DO for small to medium message sizes. However, on 8×8 2dfc network topology, the performance of MD is slightly worse than DO because of connection initialization overhead. This affect is more clearly on 16×16 2dfc network topology, where the message size exceeds than 512 KB. The algorithmic advantages outweigh the initialization cost. For large message size, MD outperforms DO across all network scales. With a data size of 32 MB, MD speeds up 1.38×, 1.30× and 1.17× over DO on 4×4, 8×8 and 16×16 2dfc, respectively.

In conclusion, topology-aware algorithms outperform topology-unaware algorithms by efficiently exploiting topology information while avoiding contention. Among them, MD utilizes network resources most effectively and therefore outperforms other algorithms for large message sizes. Overall, we consider that MD algorithm represents

the most efficient collective communication method optimized for the 2dfc network topology.

Impact of link bandwidth

Table 4.3: Collective operations and algorithms evaluated in this dissertation.

Operation	Algorithm	Description
alltoall	Pairwise	Exchanges data directly between each pair of processes in $p-1$ steps.
	Bruck	Uses a sequence of cyclic shifts to complete alltoall communication in $\lceil \log_2 p \rceil$ steps with index remapping.
	DO	Conventional algorithm for doing collective in 2dfc.
	MULTITREE	Organizes processes into multiple spanning trees to perform simultaneous data exchanges.
	MD	Our proposed for enhancing performance for collectives on 2dfc.
allgather	Pairwise	Reduces data between process pairs in a direct exchange pattern until completion.
	Ring	Passes partial results around a logical ring, performing reduction and distribution in $p-1$ steps.
	Bruck	Similar to alltoall Bruck, but incorporates a reduction operation in each cyclic shift step.
	DO	Conventional algorithm for doing collective in 2dfc.
	MD	Our proposed for enhancing performance for collectives on 2dfc.
allreduce	Binomial	Uses a binomial tree to double the amount of data held by each process in $\log_2 p$ steps.
	Local Ring (lr)	Performs communication within local subgroups arranged in a ring before global aggregation.
	Recursive-Doubling (rdb)	Pairs processes in each step to exchange and merge data, doubling the gathered dataset per step.
	DO	Conventional algorithm for doing collective in 2dfc.
	MD	Our proposed for enhancing performance for collectives on 2dfc.

There is a clear trend in designing HPC systems toward increasing link bandwidth to support heavy communication workloads. The 22^{nd} (and 69^{th}) supercomputers in the Top500 [1] list, for example, are equipped with link bandwidths of 200 Gbps using SlingShot-11 and InfiniBand HDR interconnects, respectively. In this part of this dissertation, we evaluate how link bandwidth impacts collective algorithms and their

performance. Specifically, we vary the 2dfc link bandwidth from 100Gbps to 400Gbps while keeping the per-message latency fixed at 1 µs, reusing the algorithms described in Section 4.3.2. Table 4.3 summarizes the algorithms under evaluation.

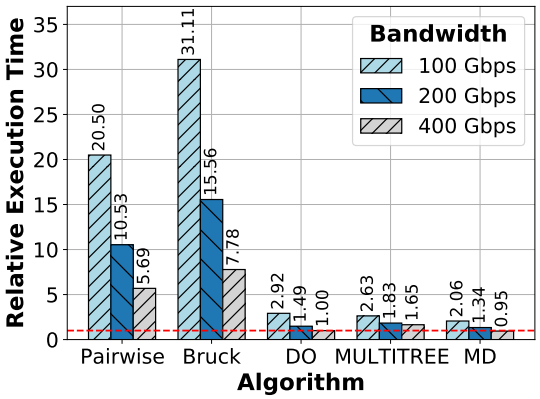


Figure 4.13: Impact of the link bandwidth to performance of alltoall algorithms (8×8 2dfc, 16MB message size). Values are normalized to the DO algorithm with 400Gbps.

Figure 4.13 shows the normalized communication time of the evaluated alltoall algorithms, relative to the communication time of the DO algorithm at a link bandwidth of 400 Gbps. Increasing link bandwidth improves the performance of all algorithms, and MD algorithm consistently achieves the best performance across all cases. However, the performance advantage of MD over DO diminishes as bandwidth increases. For example, the communication time of MD is only 0.70×, 0.89×, and 0.95× that of DO at the link bandwidth of 100, 200, and 400 Gbps, respectively. This decreases is expected because the ratio between data transfer time and startup latency per communication is reduced as bandwidth increases. Specifically, data transfer time reduces with higher bandwidth, whereas the startup latency (α) is assumed to remain constant.

Figure 4.14 shows the normalized communication time of the evaluated allgather algorithms, relative to the communication time of the DO algorithm at a link bandwidth of 400 Gbps. As expected, increasing link bandwidth improves the performance of all algorithms, and MD algorithm consistently achieves the best performance in all cases. However, the performance advantage of MD over the others decreases as bandwidth increases. The communication time of MD is only as 0.16×, 0.25×, and 0.40× as those of the Ring (the while used algorithm in production libraries) with link bandwidth of 100, 200, and 400 Gbps, respectively. This reduction is expected because the ratio

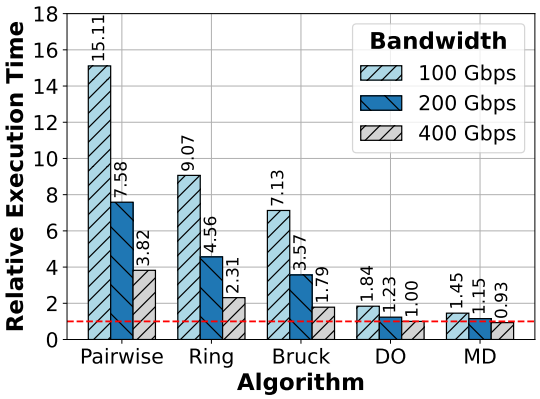


Figure 4.14: Impact of the link bandwidth on the performance of allgather algorithms (8×8 2dfc, 16MB message size). Values are normalized to the DO algorithm with 400Gbps.

between data transfer time and startup latency per communication step decreases with higher bandwidth. Specifically, data transfer time decreases as bandwidth increases, while the startup latency (α) is assumed to remain constant.

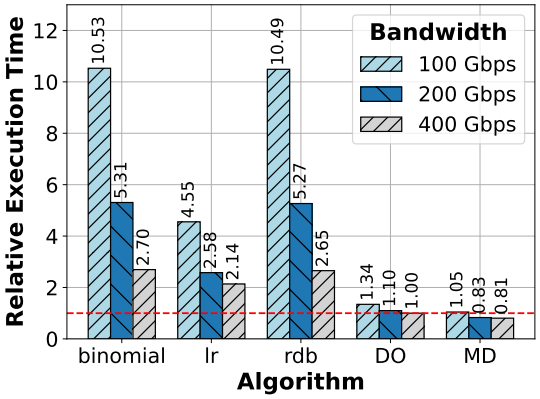


Figure 4.15: Impact of the link bandwidth on the performance of allreduce algorithms (8×8 2dfc, 16MB message size). Values are normalized to the DO algorithm with 400Gbps.

Figure 4.15 shows the normalized communication time of the evaluated allreduce algorithms, relative to the communication time of the DO algorithm at a link bandwidth of 400 Gbps. The figure exhibits a trend similar to that observed for the alltoall and allgather operations:

- The MD algorithm consistently achieves the best performance across all cases.

- As bandwidth increases, the performance advantage of MD over the other algorithms decreases.

4.4 Comparison with Other Topologies

4.4.1 Scalability

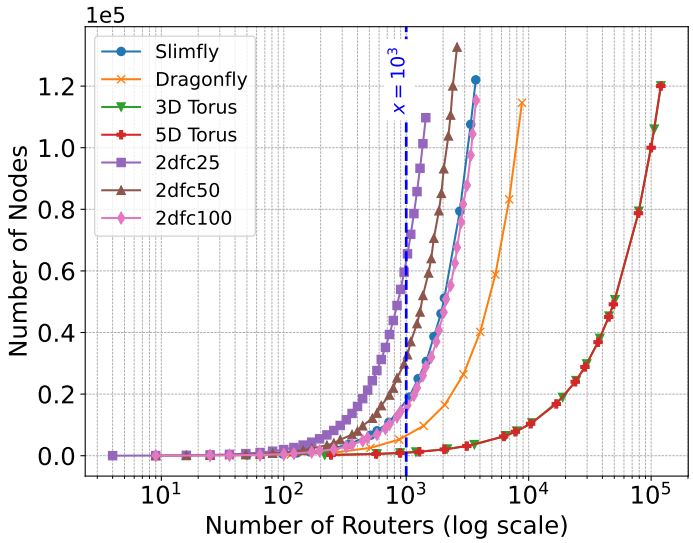


Figure 4.16: Comparison of node scalability between different network topologies.

Figure 4.16 shows the relationship between the number of routers (x-axis, log scale) and the corresponding number of nodes (y-axis) for different topology, including Slimfly, Dragonfly, 3D/5D Torus, and 2-level fully connected (2dfc) variants with 25% (2dfc25), 50% (2dfc50), and 100% (2dfc100) bisection bandwidth. For a fixed number of switches, the more nodes are connected, the better. For example, the vertical dashed blue line at $x = 10^3$ shows the configuration point with 1000 switches. With this configuration, Torus 3D can connect the least number of nodes, about 1000 nodes. In contrast, 2dfc25 can connect the largest number of nodes, over 65000 nodes. Dragonfly, 2dfc100, Slimfly, and 2dfc50 show an intermediate performance with approximately

7400, 16000, 190000 and 37000 nodes respectively [6].Note that the data for Slimfly, Dragonfly, 3D and 5D Torus is reproduced by the script of Slimfly paper [10].

Overall, 2dfc is comparable with state-of-the-art topologies in terms of network size. In the next section, we will research the cost of each topology.

4.4.2 Cost Comparison

The cost of the network is one of the key criteria for evaluating the practicality of a topology. This cost analysis focuses on the hardware expenses required for the 2dfc topology compared to other topologies. We adopt the cost model described in [20] and [10], which includes the costs of routers and interconnection cables. Following the same assumption as in [10], we consider a ground installation where each rack of size $1 \times 1 \times 2$ meters contains routers together with endpoints. Under this assumption, the maximum Manhattan distance [96] between routers or between routers and endpoints is approximately two meters, and the minimum is 5–10 cm. Therefore, the average intra-rack cable length is assumed to be one meter. Additionally, we include a two-meter cable overhead for each global link, as in [10]. In this section, we use the hardware listed in Listing 4.1 to estimate the cost of different systems. Most prices are obtained from the webpage [97].

Listing 4.1: Mellanox InfiniBand Equipment List

```
Mellanox Passive Copper Cable, IB EDR, 100Gb/s, QSFP28, LSZH, 1 meter, Part ID: MCP1600-E001E30 $134.17
Mellanox Passive Copper Cable, IB EDR, 100Gb/s, QSFP28, LSZH, 1.5 meter, Part ID: MCP1600-E01AE30 $143.75
Mellanox Passive Copper Cable, IB EDR, 100Gb/s, QSFP28, LSZH, 2 meters, Part ID: MCP1600-E002E30 $159.32
Mellanox Passive Copper Cable, IB EDR, 100Gb/s, QSFP28, LSZH, 3 meters, Part ID: MCP1600-E003E26 $179.69
Mellanox Passive Copper Cable, IB EDR, 100Gb/s, QSFP28, LSZH, 5 meters, Part ID: MCP1600-E005E26 $339.01
Mellanox Passive Copper Cable, IB HDR, 200Gb/s, QSFP56, LSZH, 1 meter, Part ID: MCP1650-H001E30 $205.00
Mellanox Passive Copper Cable, IB HDR, 200Gb/s, QSFP56, LSZH, 1.5 meter, Part ID: MCP1650-H01AE30 $225.00
Mellanox Passive Copper Cable, IB HDR, 200Gb/s, QSFP56, LSZH, 2 meters, Part ID: MCP1650-H002E26 $248.00

Mellanox VCSEL-Based Active Fiber Cable, IB EDR, 100Gb/s, QSFP, LSZH, 1 meter, Part ID: MFA1A00-E001 $668.44
Mellanox VCSEL-Based Active Fiber Cable, IB EDR, 100Gb/s, QSFP, LSZH, 10 meters, Part ID: MFA1A00-E010 $673.23
Mellanox VCSEL-based Active Fiber Cable, IB EDR, 100Gb/s, QSFP, LSZH, 15 meters, Part ID: MFA1A00-E015 $710.36
Mellanox VCSEL-Based Active Fiber Cable, IB EDR, 100Gb/s, QSFP, LSZH, 20 meters, Part ID: MFA1A00-E020 $751.09
Mellanox VCSEL-Based Active Fiber Cable, IB EDR, 100Gb/s, QSFP, LSZH, 3 meters, Part ID: MFA1A00-E003 $639.69
Mellanox VCSEL-Based Active Fiber Cable, IB EDR, 100Gb/s, QSFP, LSZH, 30 meters, Part ID: MFA1A00-E030 $848.12
Mellanox VCSEL-Based Active Fiber Cable, IB EDR, 100Gb/s, QSFP, LSZH, 5 meters, Part ID: MFA1A00-E005 $649.27
```

[6] Bisection bandwidth of 3D Torus and 5D Torus varies and depend on the number of nodes in the topology. The bisection bandwidth equal to $\frac{2}{D\sqrt[D]{N}}$%, for 3D Torus and 5D Torus respectively, where D is the number of dimensions, N is the total number of nodes. The bisection bandwidth varies from 20% to 4% and 50% to 20% for 3D Torus and 5D Torus respectively. Bisection bandwidth of balanced Dragonfly is $\frac{N+2p^2-1}{2N} \approx 50\%$. According to the report of Slimfly paper [10] Slimfly has better bisection bandwidth than Dragonfly and 5D Torus but less than 100%.

```
Mellanox VCSEL-Based Active Fiber Cable, IB EDR, 100Gb/s, QSFP, LSZH, 50 meters, Part ID: MFA1A00-E050 $1,211.09

Mellanox Switch-IB 2 MSB7570-E 36 x EDR InfiniBand 100 Gigabit QSFP28 $19,139
Quantum HDR 40 QSFP56 40-port HDR InfiniBand $30,720
Mellanox Quantum-2 QM9700 64-port $34,555
```

Cable Cost Estimation Function

To estimate the cable cost, we apply linear regression to the cost in Listing 4.1. Finally, we obtain $f_{el}(x) = 22.76x + 111.41$ $[\$/m]$ function for the cost of electrical cables and $f_{op}(x) = 6.76x + 624.68$ $[\$/m]$ for optical cable. We plot the estimate function and the actual prices on Figure 4.17.

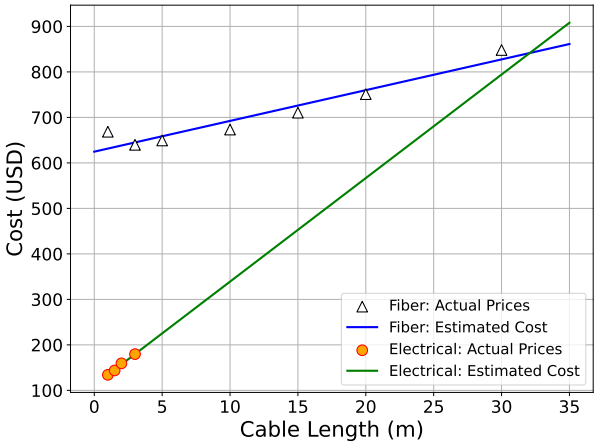


Figure 4.17: Estimated cost for electrical and optical cables.

Switch Cost Estimation Function

Using the same method for the switch prices on in Listing 4.1, we obtain the cost function for switches $f(x) = 414.25x + 8806.20$ $[\$]$. We plot the estimation function and the actual price on Figure 4.18.

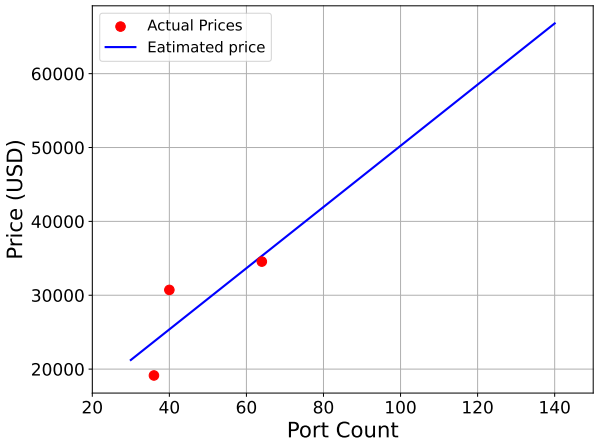


Figure 4.18: Estimated cost for network switches.

Total Cost Comparison

Tori: we take 3D-Torus and 5D-Torus as the competitors for 2dfc. We also assume that tori have folded design that do not require the optical links as in [10].
Dragonfly: we use balanced Dragonfly [7] ($a = 2p = 2h$). Where a is the number of routers in a group, h is the number of optical cable to connect each router, p is the number of endpoints in each router. $g = a \times h + 1$ is the number of fully connected groups in the topology.
Slimfly: we use the Slimfly-MMS graph as in the original paper. Different configuration of Slimfly will be use to match the number of endpoints in the experiment.
2dfc: physical layout of 2dfc will be arranged into $m_2 \times m_1$ racks. Each rack has one top-of-rack switch and endpoints (terminals). There are $m_2 + m_1$ fully connected groups of routers. In a group of m fully connected routers, there are:

- $m - 1$ distance one optical cables, total length of cables is $(m - 1) \times 1$
- $m - 2$ distance two optical cables, total length of cables is $(m - 2) \times 2$
- $m - x, x < m$ distance x optical cable, total length of cables is $(m - x) \times x$

Based on this the total length of the cable in a m-router fully connect group will be

$$\sum_{x=1}^{m-1} (m - x)x \tag{4.9}$$

We add one meter to each global link for overhead. Therefore the total length of optical cable for one fully connect group is:

$$\sum_{x=1}^{m-1}(m-x)(x+1) \tag{4.10}$$

Apply the cost function for optical links f_{op} to Equation 4.10, we obtain the optical cable cost for one fully connected group:

$$\sum_{x=1}^{m-1}(m-x)f_{op}(x+1) \tag{4.11}$$

There are m_2 groups of m_1 switches. There are m_1 groups of m_2 switches. Therefore the total cost for optical cable to connect top-of-rack switch is

$$m_2\sum_{x=1}^{m_1-1}(m_1-x)f_{op}(x+1)+m_1\sum_{x=1}^{m_2-1}(m_2-x)f_{op}(x+1) \tag{4.12}$$

There are T terminals in each rack, therefore, the total cost for electrical cable is

$$m_2\times m_1\times T\times f_{el}(1) \tag{4.13}$$

The cost for a 2dfc system is the total of electrical links, optical links and switches. Different configurations of 2dfc will be used. In this experiment, we take 2dfc with 25%, 50% and 100% bisection bandwidth to compare with Slimfly, Dragonfly, 3D Torus and 5D Torus.

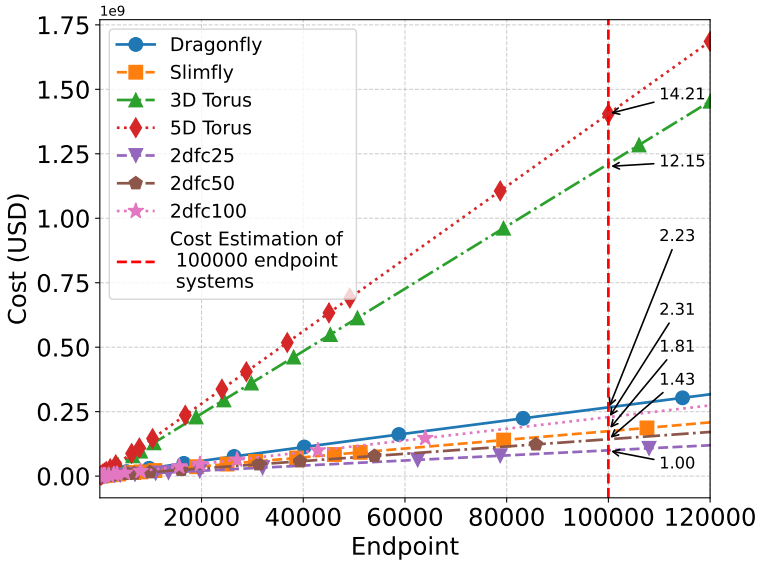


Figure 4.19: The cost comparison of 2dfc to other topologies.

Figure 4.19 presents the cost comparison in USD for different network topologies. Using the updated device prices as of 2025 (see Listing 4.1), the 3D Torus and 5D Torus remain the most expensive topologies, consistent with the findings of the Slimfly paper [10]. Among the group of high-radix topologies, Dragonfly shows the highest cost, followed by 2dfc100, while Slimfly exhibits a moderate price level. Since it satisfies the bisection bandwidth, 2dfc25 achieves the lowest cost, followed by 2dfc50.

In particular, the red line in Figure 4.19 indicates the cost estimation for systems with 100,000 endpoints. This provides a clear reference for comparing the relative scaling of different topologies. The costs are normalized relative to 2dfc25, yielding values of 1.43, 1.81, 2.23, 2.31, 12.15, and 14.21 for 2dfc50, Slimfly, 2dfc100, Dragonfly, 3D Torus, and 5D Torus, respectively. Overall, with its flexible configuration, 2dfc can be comparable to state-of-the-art topologies such as Dragonfly in terms of system construction cost.

Considering system cost and scalability, 2dfc is an attractive network topology. In contrast, 3D torus and 5D torus require a larger number of switches to connect the same number of nodes. In our network cost model, the number of switches has a significant impact on the total network cost. From this perspective, high-radix network topologies are preferable. Since 2dfc can be optimized to meet different cost constraints

(e.g., 2dfc25, 2dfc50, and 2dfc100), it offers flexible design points. Therefore, 2dfc is a good candidate among high-radix network topologies in terms of cost-performance trade-offs.

4.5 Extensibility to Network Topologies in Existing Supercomputers

The 2dfc topology can be regarded as part of the Cray Aries interconnect [41]. As of June 2025, systems such as Piz Daint (ranked 77th) [44] and Theta (ranked 194th) [98] employ a 2dfc network internally. The proposed MD algorithm exploits the multi-port capability of the network to improve the performance of the DO algorithm.

The core idea of MD is to maximize link utilization. This idea is inspired by prior research. Peng et al. [99] leverage multi-core processors and multi-network-interface systems to improve the performance of all-to-all communication. Their work uses multiple workers (CPU cores) for gathering and scattering, thereby achieving higher throughput. In addition, multiple network interfaces at a node allow the simultaneous processing of multiple communication requests, further improving overall performance. Wilkins et al. [58] apply a similar idea to optimize a set of ten collective algorithms. Their work exploits multi-port capabilities to generalize three conventional algorithms: binomial [64], recursive doubling [100], and ring [85]. By using multiple interfaces to send and receive multiple messages concurrently, they achieve improved communication performance. The proposed algorithms are named k-nominal, recursive multiplying, and k-ring. Zong et al. [101] observed that existing algorithms do not sufficiently exploit cluster bandwidth, resulting in significant bandwidth underutilization. They proposed TIAD (Topology-Aware Interleaved All-Reduce), which partitions data into multiple segments and performs intra- and inter-group communication concurrently.

The applicability of the MD algorithm extends beyond the two-dimensional case. In fact, the underlying idea of maximizing link utilization via multi-port communication naturally generalizes to higher-dimensional fully connected network topologies. In the following, we discuss how the MD algorithm can be extended to three-dimensional and, more generally, to n-dimensional fully connected networks.

An ***n*-Dimensional Fully Connected (ndfc) Network Topology** is defined as a bidirectional graph, consisting of N nodes, represented as a Cartesian product:

$$N = \prod_{i=1}^{n} m_i \tag{4.14}$$

nodes, where m_i denotes the size of the network along dimension i.

Each node is identified by an n-tuple:

$$A = (a_n, a_{n-1}, \ldots, a_1), \quad 0 \leq a_i < m_i, \ \forall i \in \{1, \ldots, n\}. \tag{4.15}$$

Two nodes $A = (a_n, \ldots, a_1)$ and $A' = (a'_n, \ldots, a'_1)$ are connected if and only if there exists exactly one dimension k such that

$$a'_k \neq a_k \quad \text{and} \quad a'_j = a_j, \ \forall j \neq k. \tag{4.16}$$

This means a node is connected to all other nodes that differ in exactly one dimension.

The MD algorithm for an ndfc can be conceptualized as follows. The data at each node is divided into n partitions D_i. The size of each partition D_i is carefully calculated to balance the communication time among all dimensions. This balancing minimizes idle time between communication steps.

The algorithm proceeds in n communication steps. In each step, all available ports of a node are fully used to enable parallel communication. In a given step, the data partition D_i at each node is transmitted along the i-th dimension of the network.

Notably, at step j, the transmitted data includes not only the partition destined for the current dimension, but also the partitions that must be forwarded for the remaining $n - j$ dimensions in subsequent steps.

For example, in a three-dimensional fully connected network (3dfc), during the first step, communication is performed along the X-axis. The transmitted data consists of the partitions destined for nodes on the same X-axis, as well as the partitions that will be forwarded along the Y- and Z-axes in later steps. In the second step, the transmitted data includes the partitions for the Y-axis and the partitions to be forwarded along the Z-axis. In the third step, the transmitted data contains only the partitions for the Z-axis, completing the algorithm.

This generalized algorithm preserves the bandwidth-optimal property[7], as it transmits only the minimum amount of data required and remains contention-free by independently scheduling different data flows across different dimensions. However, as the number of dimensions in the topology increases, the overhead of partitioning and combining messages at each node introduces non-negligible latency. Consequently, the n-dimensional MD algorithm is most effective for large message sizes, where transmission time dominates startup latency.

The idea of examination on link utilization can be applied to the network topologies employed in existing supercomputers such as Slingshot [102] or Dragonfly [7] network topology. Currently, collective communication algorithms used on top supercomputers listed in the Top500 [1] are mainly traditional algorithms such as Ring algorithm [86], Recursive-Doubling [103], Pairwise, Bruck algorithm [78, 69]. These algorithms tend to underutilize network links and often suffer from high contention. As a result, their performance is generally worse in simulations when compared with topology-aware algorithms. For example, the ring algorithm exhibits lower performance than TIAD [101], which is a topology-aware algorithm. By applying link utilization analysis to topology-aware algorithms on Dragonfly networks, we can similarly identify opportunities for further optimization. For instance, Group-First and Router-First [104] are two complementary algorithms designed for broadcast operations. Based on their operational characteristics, Group-First can be interpreted as a local-links-first strategy, whereas Router-First corresponds to a global-links-first strategy. An analysis of link utilization in these algorithms reveals that network links are not fully exploited, indicating potential for further improvement.

Similar observations can be made when analyzing the communication timeline of the PAARD algorithm [105]. In PAARD, local and global links are utilized sequentially rather than concurrently, which leads to link underutilization. In fact, Zong *et al.* [101] adopt a similar analytical perspective to propose an improved collective communication algorithm.

These observations suggest that analyzing and maximizing link utilization provides a unifying and systematic approach for improving collective communication algorithms across a wide range of network topologies. This perspective not only explains the limitations of existing algorithms but also offers clear guidance for the design of more

[7]The proof in Paragraph 20 can be generalized to the n-dimensional case to obtain a similar result

efficient, topology-aware collectives.

4.6 Summary

This chapter focused on improving the performance of collective communication, system growth and installtion cost of 2dfc network topology.

Although DO is bandwidth-optimal, its sequential use of links across dimensions leads to resource underutilization. Meanwhile, MULTITREE algorithm fully utilizes resources and maintaining bandwidth optimality It suffers from a large number of communication steps, resulting in significant latency, particularly for small message sizes or in networks with high startup costs. By contrast, the MD algorithm addresses the above limitations by fully utilizing the network resources in parallel while maintaining a small number of communication steps. The algorithm achieves this by splitting data into two parts and transferring them simultaneously across dimensions efficiently.

Our analytical and simulation evaluation results yield the following findings.

- MD consistently achieves the best theoretical communication time compared to DO and MULTITREE, possessing both the smallest latency and bandwidth coefficients.

- Topology-aware algorithms (DO, MULTITREE, and MD) demonstrate substantial performance gains over conventional topology-unaware algorithms (e.g., Pairwise, Bruck, Ring) for alltoall, allgather, and allreduce benchmarks. These gains steam from their efficient use of network-topology information and contention avoidance.

- The MD algorithm consistently outperforms both DO and MULTITREE across various message sizes and network scales: For alltoall, MD was up to 1.58x faster than MULTITREE and 1.66x faster than DO on a 16×16 2dfc. For allgather, MD achieved up to 1.5x and 1.2x faster performance compared to DO on 4×4 and 8×8 2dfc, respectively. It significantly outperformed conventional algorithms (e.g., 6.83x faster than Bruck and 16.3x faster than Pair on 16×16 with a 4MB message size). For allreduce, MD demonstrated superior performance for large

message sizes, speeding up 1.38x, 1.30x, and 1.17x over DO on 4×4, 8×8, and 16×16 2dfc, respectively, at a 32MB message size.

- While increasing link bandwidth improves performance for all algorithms, MD remains the best performer. However, its relative advantage over other algorithms becomes smaller at higher bandwidths, because the startup latency (α) contribution becomes more prominent as message transfer time ($m\beta$) decreases.

The analysis of network size growth shows that the 2dfc network topology is comparable with state-of-the-art topologies in terms of network size. In addition, the system install cost is also reasonable for building large systems when compare with state-of-the-art topologies such as Dragonfly.

5 Conclusions and Future Works

5.1 Conclusions

This dissertation has investigated topology-aware multi-port message-combine collective communication for high-performance computing (HPC) networks, focusing on Kautz with a diameter of two and two-dimensional fully connected (2dfc) network topologies. The main findings and contributions are summarized as follows:

Multi-port communication on Kautz topology We adopted the Kautz graph as the basis for our study. We proposed multi-port message-combine collective communications that exploit all available outgoing links of each node. Simulation results demonstrated that these methods effectively increase aggregate bandwidth by enabling parallel communication paths and reduce latency through pre-scheduled contention-free operations.

Extension to two-dimensional fully connected (2dfc) topology Recognizing the limited practicality of unidirectional Kautz networks in real-world HPC systems, we extended our approach to bi-directional 2dfc networks. The proposed multi-port message-combine operations were shown to improve throughput by up to 1.38x over a competitor at 32MB data size. These results confirm that the method is well suited for practical deployment in state-of-the-art systems, including HyperX and Dragonfly.

Comprehensive evaluation Using the SimGrid framework, we evaluated the proposed algorithms under both micro-benchmarks (Allgather, Alltoall, Allreduce) and deep learning workloads (ResNet50, VGG16). The results consistently showed significant reductions in execution time compared with conventional algorithms, confirming both the efficiency and scalability of our approach.

5.2 Future Works

While the proposed methods are effective for diameter-two networks, the evaluation revealed limitations when extending to higher-dimensional or more complex network topologies. Message-combine operations introduce non-negligible latency overheads, particularly for small messages, which diminish performance advantages in such settings. Based on the findings, promising directions for future research include:

- Designing lightweight message-combine techniques to mitigate overhead in higher-dimensional networks.
- Extending the proposed algorithms to heterogeneous HPC systems, where network bandwidth and latency characteristics vary across components.
- Investigating fault tolerance and adaptive routing strategies to ensure robustness under dynamic workloads and failures.
- Investigating the performance evaluation of collective communication under the condition that interaction and interferences from the other applications can affect the load balance of parallel computing systems.

Among these directions, the development of lightweight message-combine techniques is particularly important, as it could enable the scalable deployment of topology-aware multi-port communication. More broadly, the analytical framework presented in this dissertation offers a general guideline for designing and optimizing collective communication algorithms in next-generation high-radix interconnection networks.

Bibliography

[1] "Top500 november 2019," https://www.top500.org/lists/top500/2019/11/.

[2] Argonne Leadership Computing Facility, "Mira: Alcf' s 10-petaflops ibm blue gene/q supercomputer," https://www.alcf.anl.gov/alcf-resources/mira, 2013.

[3] S. Chunduri, S. Parker, P. Balaji, K. Harms, and K. Kumaran, "Characterization of mpi usage on a production supercomputer," in *International Conference for High Performance Computing, Networking, Storage, and Analysis (SC)*, 2018.

[4] C. E. Leiserson, "Fat-trees: universal networks for hardware-efficient supercomputing," *IEEE transactions on Computers*, vol. 100, no. 10, pp. 892–901, 1985.

[5] T. T. Nguyen and M. Wahib, "An allreduce algorithm and network co-design for large-scale training of distributed deep learning," in *IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, 2021, pp. 396–405.

[6] J. Domke, S. Matsuoka, I. R. Ivanov, Y. Tsushima, T. Yuki, A. Nomura, S. Miura, N. McDonald, D. L. Floyd, and N. Dubé, "Hyperx topology: First at-scale implementation and comparison to the fat-tree," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2019, pp. 1–23.

[7] J. Kim, W. J. Dally, S. Scott, and D. Abts, "Technology-driven, highly-scalable dragonfly topology," in *2008 International Symposium on Computer Architecture*. IEEE, 2008, pp. 77–88.

[8] J. H. Ahn, N. Binkert, A. Davis, M. McLaren, and R. S. Schreiber, "Hyperx: topology, routing, and packaging of efficient large-scale networks," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, 2009, pp. 1–11.

[9] J. Domke, S. Matsuoka, I. Radanov, Y. Tsushima, T. Yuki, A. Nomura, S. Miura, N. McDonald, D. L. Floyd, and N. Dubé, "The first supercomputer with hyperx topology: A viable alternative to fat-trees?" in *2019 IEEE Symposium on High-Performance Interconnects (HOTI)*. IEEE, 2019, pp. 1–4.

[10] M. Besta and T. Hoefler, "Slim fly: A cost effective low-diameter network topology," in *SC'14: proceedings of the international conference for high performance computing, networking, storage and analysis*. IEEE, 2014, pp. 348–359.

[11] G. E. Moore *et al.*, "Cramming more components onto integrated circuits," 1965.

[12] M. Miller and J. Sirán, "Moore graphs and beyond: A survey of the degree/-diameter problem," *The electronic journal of combinatorics*, pp. DS14–May, 2012.

[13] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey, "Jellyfish: Networking data centers randomly," in *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, 2012, pp. 225–238.

[14] S. Hoory, N. Linial, and A. Wigderson, "Expander graphs and their applications," *Bulletin of the American Mathematical Society*, vol. 43, no. 4, pp. 439–561, 2006.

[15] A. Valadarsky, M. Dinitz, and M. Schapira, "Xpander: Unveiling the secrets of high-performance datacenters," in *Proceedings of the 14th ACM Workshop on Hot Topics in Networks*, 2015, pp. 1–7.

[16] Bhuyan and Agrawal, "Generalized hypercube and hyperbus structures for a computer network," *IEEE Transactions on Computers*, vol. C-33, no. 4, pp. 323–333, 1984.

[17] W. J. Dally and B. P. Towles, *Principles and practices of interconnection networks*. Elsevier, 2004.

[18] J. Duato, S. Yalamanchili, and L. Ni, *Interconnection networks*. Elsevier, 2002.

[19] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*. Elsevier, 2011.

[20] J. Kim, W. J. Dally, and D. Abts, "Flattened butterfly: a cost-efficient topology for high-radix networks," in *Proceedings of the 34th annual international symposium on Computer architecture*, 2007, pp. 126–137.

[21] A. Singla, A. Singh, K. Ramachandran, L. Xu, and Y. Zhang, "Proteus: a topology malleable data center network," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010, pp. 1–6.

[22] K. Chen, A. Singla, A. Singh, K. Ramachandran, L. Xu, Y. Zhang, X. Wen, and Y. Chen, "Osa: An optical switching architecture for data center networks with unprecedented flexibility," *IEEE/ACM Transactions on Networking*, vol. 22, no. 2, pp. 498–511, 2013.

[23] K. Wen, P. Samadi, S. Rumley, C. P. Chen, Y. Shen, M. Bahadori, K. Bergman, and J. Wilke, "Flexfly: Enabling a reconfigurable dragonfly through silicon photonics," in *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2016, pp. 166–177.

[24] M. Y. Teh, Z. Wu, and K. Bergman, "Flexspander: augmenting expander networks in high-performance systems with optical bandwidth steering," *Journal of Optical Communications and Networking*, vol. 12, no. 4, pp. B44–B54, 2020.

[25] G. Liu, R. Proietti, M. Fariborz, P. Fotouhi, X. Xiao, and S. B. Yoo, "Architecture and performance studies of 3d-hyper-flex-lion for reconfigurable all-to-all hpc networks," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2020, pp. 1–16.

[26] W. M. Mellette, R. Das, Y. Guo, R. McGuinness, A. C. Snoeren, and G. Porter, "Expanding across time to deliver bandwidth efficiency and low latency," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, 2020, pp. 1–18.

[27] H. Ballani, P. Costa, R. Behrendt, D. Cletheroe, I. Haller, K. Jozwik, F. Karinou, S. Lange, K. Shi, B. Thomsen *et al.*, "Sirius: A flat datacenter network with nanosecond optical switching," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 782–797.

[28] M. Koibuchi, H. Matsutani, H. Amano, D. F. Hsu, and H. Casanova, "A case for random shortcut topologies for hpc interconnects," in *2012 39th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2012, pp. 177–188.

[29] R. V. Tomic, "Network throughput optimization via error correcting codes," *arXiv preprint arXiv:1301.4177*, 2013.

[30] I. Fujiwara, M. Koibuchi, H. Matsutani, and H. Casanova, "Skywalk: A topology for hpc networks with low-delay switches," in *2014 IEEE 28th International Parallel and Distributed Processing Symposium*. IEEE, 2014, pp. 263–272.

[31] M. Flajslik, E. Borch, and M. A. Parker, "Megafly: A topology for exascale systems," in *International Conference on High Performance Computing*. Springer, 2018, pp. 289–310.

[32] W. M. Mellette, R. McGuinness, A. Roy, A. Forencich, G. Papen, A. C. Snoeren, and G. Porter, "Rotornet: A scalable, low-complexity, optical datacenter network," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, 2017, pp. 267–280.

[33] M. Y. Teh, Z. Wu, M. Glick, S. Rumley, M. Ghobadi, and K. Bergman, "Performance trade-offs in reconfigurable networks for hpc," *Journal of Optical Communications and Networking*, vol. 14, no. 6, pp. 454–468, 2022.

[34] S. B. Akers and B. Krishnamurthy, "A group-theoretic model for symmetric interconnection networks," *IEEE transactions on Computers*, vol. 38, no. 4, pp. 555–566, 1989.

[35] B. D. McKay, M. Miller, and J. Širáň, "A note on large graphs of diameter two and given maximum degree," *Journal of Combinatorial Theory, Series B*, vol. 74, no. 1, pp. 110–118, 1998.

[36] N. Blach, M. Besta, D. De Sensi, J. Domke, H. Harake, S. Li, P. Iff, M. Konieczny, K. Lakhotia, A. Kubicek *et al.*, "A high-performance design, implementation, deployment, and evaluation of the slim fly network," in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, 2024, pp. 1025–1044.

[37] W. Kautz, "Bounds on directed (d, k) graphs," *Theory of cellular logic networks and machines*, pp. 20–28, 1968.

[38] W. H. Kautz, "Design of optimal interconnection networks for multiprocessors," *Architectures and Design of Digital Computers, NATO Advanced Summer Inst.*, pp. 249–272, 1969.

[39] J.-C. Bermond, N. Homobono, and C. Peyrat, "Connectivity of kautz networks," *Discrete Mathematics*, vol. 114, no. 1-3, pp. 51–62, 1993.

[40] A. J. Hoffman and R. R. Singleton, "On moore graphs with diameters 2 and 3," *IBM Journal of Research and Development*, vol. 4, no. 5, pp. 497–504, 1960.

[41] NERSC, "Aries interconnect," 2014, [Figure page 7]. Accessed: 2025-03-18. [Online]. Available: https://www.nersc.gov/assets/pubs_presos/NUG2014Aries.pdf

[42] T. Hoefler, T. Bonato, D. De Sensi, S. Di Girolamo, S. Li, M. Heddes, J. Belk, D. Goel, M. Castro, and S. Scott, "Hammingmesh: A network topology for large-scale deep learning," in *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2022, pp. 1–18.

[43] B. Alverson, E. Froese, L. Kaplan, and D. Roweth, "Cray xc series network," *Cray Inc., White Paper WP-Aries01-1112*, 2012.

[44] Swiss National Supercomputing Centre (CSCS), "Piz Daint Supercomputer," https://www.cscs.ch/computers/piz-daint, 2013–2023, accessed: December 9th 2023.

[45] LANL, "Trinity: Advanced technology system," https://www.lanl.gov/projects/trinity/, accessed: 12 October, 2023.

[46] R. Clapp, M. Dimitrov, K. Kumar, V. Viswanathan, and T. Willhalm, "Quantifying the performance impact of memory latency and bandwidth for big data workloads," in *2015 IEEE International Symposium on Workload Characterization*. IEEE, 2015, pp. 213–224.

[47] R. W. Hockney, "The communication challenge for mpp: Intel paragon and meiko cs-2," *Parallel computing*, vol. 20, no. 3, pp. 389–398, 1994.

[48] A. Geist, W. Gropp, S. Huss-Lederman, A. Lumsdaine, E. Lusk, W. Saphir, T. Skjellum, and M. Snir, "Mpi-2: Extending the message-passing interface," in *European Conference on Parallel Processing*. Springer, 1996, pp. 128–135.

[49] W. Gropp, "Mpich2: A new start for mpi implementations," in *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*. Springer, 2002, pp. 7–7.

[50] Message Passing Interface Forum, "Mpi: A message-passing interface standard, version 4.1," University of Tennessee, Knoxville, Tech. Rep., November 2023, version 4.1. [Online]. Available: https://www.mpi-forum.org/docs/mpi-4.1/mpi41-report.pdf

[51] A. Weingram, Y. Li, H. Qi, D. Ng, L. Dai, and X. Lu, "xccl: A survey of industry-led collective communication libraries for deep learning," *Journal of Computer Science and Technology*, vol. 38, no. 1, pp. 166–195, 2023.

[52] R. Rabenseifner, "Optimization of collective reduction operations," in *Computational Science-ICCS 2004: 4th International Conference, Kraków, Poland, June 6-9, 2004, Proceedings, Part I 4*. Springer, 2004, pp. 1–9.

[53] Z. Cai, Z. Liu, S. Maleki, M. Musuvathi, T. Mytkowicz, J. Nelson, and O. Saarikivi, "Synthesizing optimal collective algorithms," in *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2021, pp. 62–75.

[54] A. Kohler, M. Radetzki, P. Gschwandtner, and T. Fahringer, "Low-latency collectives for the intel scc," in *2012 IEEE International Conference on Cluster Computing*. IEEE, 2012, pp. 346–354.

[55] K. T. Pham, T. T. Nguyen, H. Yamaguchi, Y. Urino, and M. Koibuchi, "Scalable low-latency inter-fpga networks," in *2022 IEEE International Parallel & Distributed Processing Symposium*. IEEE, 2022, pp. 234–245.

[56] D. De Sensi, T. Bonato, D. Saam, and T. Hoefler, "Swing: Short-cutting rings for higher bandwidth allreduce," in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, 2024, pp. 1445–1462.

[57] P. Basu, L. Zhao, J. Fantl, S. Pal, A. Krishnamurthy, and J. Khoury, "Efficient all-to-all collective communication schedules for direct-connect topologies," in *Proceedings of the 33rd International Symposium on High-Performance Parallel and Distributed Computing*, 2024, pp. 28–41.

[58] M. Wilkins, H. Wang, P. Liu, B. Pham, Y. Guo, R. Thakur, P. Dinda, and N. Hardavellas, "Generalized collective algorithms for the exascale era," in *2023 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2023, pp. 60–71.

[59] A. Shah, V. Chidambaram, M. Cowan, S. Maleki, M. Musuvathi, T. Mytkowicz, J. Nelson, O. Saarikivi, and R. Singh, "{TACCL}: Guiding collective algorithm synthesis using communication sketches," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, 2023, pp. 593–612.

[60] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 6th ed., ser. The Morgan Kaufmann Series in Computer Architecture and Design. Morgan Kaufmann, 2019.

[61] D. Scott, "Efficient all-to-all communication patterns in hypercube and mesh topologies," in *The Distributed Memory Computing Conference*, May 1991, pp. 398–403. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/DMCC.1991.633174

[62] B. L. Payne, M. Barnett, R. Littlefield, D. G. Payne, and R. V. D. Geijn, "Global combine on mesh architectures with wormhole routing," in *Int. Parallel Proc. Symp*, 1993.

[63] N. Karonis, B. de Supinski, I. Foster, W. Gropp, E. Lusk, and J. Bresnahan, "Exploiting hierarchy in parallel computer networks to optimize collective operation performance," in *International Parallel and Distributed Processing Symposium, IPDPS*, 2000, pp. 377–384.

[64] R. Thakur, R. Rabenseifner, and W. Gropp, "Optimization of collective communication operations in mpich," *International Journal of High Performance Computing Applications*, vol. 19, no. 1, pp. 49–66, 2005.

[65] "MVAPICH: MPI over InfiniBand, 10GigE/iWARP and RoCE." [Online]. Available: http://mvapich.cse.ohio-state.edu/

[66] MVAPICH Project, *MVAPICH2 2.3.7 User Guide*, Network-Based Computing Laboratory, The Ohio State University, 2021, accessed: 2025-05-06. [Online]. Available: https://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-userguide.pdf

[67] E. Chan, M. Heimlich, A. Purkayastha, and R. Van De Geijn, "Collective communication: theory, practice, and experience," *Concurrency and Computation: Practice and Experience*, vol. 19, no. 13, pp. 1749–1783, 2007.

[68] P. Sanders, J. Speck, and J. L. Träff, "Two-tree algorithms for full bandwidth broadcast, reduction and scan," *Parallel Computing*, vol. 35, no. 12, pp. 581–594, 2009.

[69] J. Bruck, C.-T. Ho, S. Kipnis, E. Upfal, and D. Weathersby, "Efficient algorithms for all-to-all communications in multiport message-passing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 8, no. 11, pp. 1143–1156, 1997.

[70] P. Patarasuk and X. Yuan, "Bandwidth efficient all-reduce operation on tree topologies," in *Parallel and Distributed Processing Symposium, IPDPS*, 2007, pp. 1–8.

[71] C. Ying, S. Kumar, D. Chen, T. Wang, and Y. Cheng, "Image classification at supercomputer scale," *arXiv preprint arXiv:1811.06992*, 2018.

[72] C.-H. Chu, P. Kousha, A. A. Awan, K. S. Khorassani, H. Subramoni, and D. K. Panda, "Nv-group: link-efficient reduction for distributed deep learning on modern dense gpu systems," in *ACM International Conference on Supercomputing*, 2020, pp. 1–12.

[73] K. S. Khorassani, C.-H. Chu, Q. G. Anthony, H. Subramoni, and D. K. Panda, "Adaptive and hierarchical large message all-to-all communication algorithms for large-scale dense gpu systems," in *IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, 2021, pp. 113–122.

[74] J. Dong, Z. Cao, T. Zhang, J. Ye, S. Wang, F. Feng, L. Zhao, X. Liu, L. Song, L. Peng *et al.*, "Eflops: Algorithm and system co-design for a high performance distributed training platform," in *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020, pp. 610–622.

[75] J. Huang, P. Majumder, S. Kim, A. Muzahid, K. H. Yum, and E. J. Kim, "Communication algorithm-architecture co-design for distributed deep learning," in *ACM/IEEE International Symposium on Computer Architecture (ISCA)*, 2021, pp. 181–194.

[76] R. L. Graham, B. W. Barrett, G. M. Shipman, T. S. Woodall, and G. Bosilca, "Open mpi: A high performance, flexible implementation of mpi point-to-point communications," *Parallel Processing Letters*, vol. 17, no. 01, pp. 79–88, 2007.

[77] "Intel® oneAPI MPI Library," https://www.intel.com/content/www/us/en/developer/tools/oneapi/mpi-library.html, Intel Corporation, accessed: 2025-07-11.

[78] J. Bruck, C.-T. Ho, S. Kipnis, and D. Weathersby, "Efficient algorithms for all-to-all communications in multi-port message-passing systems," in *Proceedings of the sixth annual ACM symposium on Parallel algorithms and architectures*, 1994, pp. 298–309.

[79] NVIDIA Corporation, "NVIDIA Collective Communications Library (NCCL)," urlhttps://developer.nvidia.com/nccl, NVIDIA, 2025, accessed: 2025-07-11.

[80] A. Sergeev and M. Del Balso, "Horovod: fast and easy distributed deep learning in tensorflow," *arXiv preprint arXiv:1802.05799*, 2018.

[81] P. Patarasuk and X. Yuan, "Bandwidth optimal all-reduce algorithms for clusters of workstations," *Journal of Parallel and Distributed Computing*, vol. 69, no. 2, pp. 117–124, 2009.

[82] A. Faraj, P. Patarasuk, and X. Yuan, "Bandwidth efficient all-to-all broadcast on switched clusters," *International Journal of Parallel Programming*, vol. 36, pp. 426–453, 2008.

[83] A. Ferrero, U. Pisani, and K. J. Kerwin, "A new implementation of a multiport automatic network analyzer," *IEEE Transactions on Microwave Theory and Techniques*, vol. 40, no. 11, pp. 2078–2085, 1992.

[84] R. Abbott, "Adaptive computer systems," in *Proceedings, IEEE Aerospace Conference*, vol. 4. IEEE, 2002, pp. 4–4.

[85] A. Gibiansky and J. Hestness, "baidu-research/tensorflow-allreduce," https://github.com/baidu-research/tensorflow-allreduce, 2017, [Online; accessed 4-November-2019].

[86] A. Gibiansky, "Bringing hpc techniques to deep learning," https://andrew.gibiansky.com/blog/machine-learning/baidu-allreduce/, 2017, accessed: 2025-05-06.

[87] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter, "Versatile, Scalable, and Accurate Simulation of Distributed Applications and Platforms," *Journal of Parallel and Distributed Computing*, vol. 74, no. 10, pp. 2899–2917, 2014.

[88] K. Mizutani, H. Yamaguchi, Y. Urino, and M. Koibuchi, "Accelerating parallel data processing using optically tightly coupled fpgas," *Journal of Optical Communications and Networking (JOCN), IEEE/OPTICA*, vol. 14, no. 2, pp. A166–A179, Feb 2022.

[89] A. N. Kahira, T. T. Nguyen, L. B. Gomez, R. Takano, R. M. Badia, and M. Wahib, "An oracle for guiding large-scale model/hybrid parallel training of convolutional neural networks," in *Proceedings of the 30th International Symposium on High-Performance Parallel and Distributed Computing*, 2021, pp. 161–173.

[90] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[91] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[92] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[93] J. Doi and Y. Negishi, "Overlapping methods of all-to-all communication and fft algorithms for torus-connected massively parallel supercomputers," in *SC'10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2010, pp. 1–9.

[94] K. T. Pham, T. T. Nguyen, H. Yamaguchi, Y. Urino, and M. Koibuchi, "Scalable low-latency inter-fpga networks," in *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2022, pp. 234–245.

[95] T. T. Nguyen, K. T. Pham, H. Yamaguchi, Y. Urino, and M. Koibuchi, "Effective switchless inter-fpga memory networks," *Journal of Parallel and Distributed Computing*, vol. 179, p. 104713, 2023.

[96] M. Barile. (2025) Taxicab metric. https://mathworld.wolfram.com/TaxicabMetric.html. MathWorld – A Wolfram Resource. Accessed: 2025-10-20.

[97] (2025) Colfax direct: Adapters, switches, cables, & interconnects. Colfax Direct. Accessed: 2025-10-01. [Online]. Available: https://www.colfaxdirect.com/

[98] K. Harms, V. Vishwanath, N. Mateev, T. Peterka, J. Davis, and M. E. Papka, "Theta: Rapid installation and acceptance of an xc40 knl system," *Concurrency and Computation: Practice and Experience*, vol. 30, no. 13, p. e4336, 2018. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1002/cpe.4336

[99] J. Peng, J. Liu, Y. Dai, M. Xie, and C. Gong, "Optimizing all-to-all collective communication on tianhe supercomputer," in *2022 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*. IEEE, 2022, pp. 402–409.

[100] R. Thakur and W. D. Gropp, "Improving the performance of collective operations in mpich," in *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*. Springer, 2003, pp. 257–267.

[101] R. Zong, J. Zhang, Z. Tang, W. Zhang, and K. Li, "Topology-aware interleaved all-reduce communication for dragonfly network," *IEEE Transactions on Networking*, 2025.

[102] D. De Sensi, S. Di Girolamo, K. H. McMahon, D. Roweth, and T. Hoefler, "An in-depth analysis of the slingshot interconnect," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. IEEE, 2020, pp. 1–14.

[103] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, "A high-performance, portable implementation of the mpi message passing interface standard," *Parallel computing*, vol. 22, no. 6, pp. 789–828, 1996.

[104] D. Xiang and X. Liu, "Deadlock-free broadcast routing in dragonfly networks without virtual channels," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 9, pp. 2520–2532, 2015.

[105] J. Ma, D. Dong, C. Li, K. Wu, and L. Xiao, "Paard: Proximity-aware all-reduce communication for dragonfly networks. in 2021 ieee intl conf on parallel & distributed processing with applications, big data & cloud computing, sustainable

computing & communications, social computing & networking (ispa/bdcloud/socialcom/sustaincom)," *Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*, pp. 255–262, 2021.