EVALUATION OF COLLECTIVE COMMUNICATION ALGORITHMS TARGETING IN-NETWORK COMPUTATION-ENABLED HIGH-PERFORMANCE NETWORKS

AĞ İÇİ HESAPLAMA ÖZELLİKLİ YÜKSEK PERFORMANSLI AĞLARI HEDEFLEYEN TOPLU İLETİŞİM ALGORİTMALARININ DEĞERLENDİRİLMESİ

ÇAĞLAYAN DÖKME

ASSOC. PROF. DR. KAYHAN MUSTAFA İMRE

Supervisor

Submitted to

Graduate School of Science and Engineering of Hacettepe University

as a Partial Fulfillment to the Requirements

for the Award of the Degree of Master of Science

in Computer Engineering

January 2025

ABSTRACT

EVALUATION OF COLLECTIVE COMMUNICATION ALGORITHMS TARGETING IN-NETWORK COMPUTATION-ENABLED HIGH-PERFORMANCE NETWORKS

Çağlayan DÖKME

Master of Science, Computer Engineering

Supervisor: Assoc. Prof. Dr. Kayhan Mustafa İMRE

January 2025, 90 pages

This thesis explores the impact of in-network computing (INC) on the performance of collective communication operations in parallel computing systems utilizing a fat-tree network topology. By integrating computational tasks within network switches, INC offers a method to enhance communication efficiency in data-intensive environments. This study compares the performance of key collective operations such as barrier synchronization, broadcast, scatter, gather, and reduce under traditional and INC-enabled approaches, evaluating metrics such as timing cost, bandwidth usage, and synchronization quality. The results reveal that INC significantly reduces timing cost and bandwidth usage in collective operations, leading to improved scalability and synchronization consistency, especially in large-scale networks. This work underscores INC's potential to optimize collective communication in parallel computing systems, providing a promising avenue for improving performance and scalability in high-performance computing infrastructures.

Keywords: in-network computing, parallel computing, collective communication

ÖZET

AĞ İÇİ HESAPLAMA ÖZELLİKLİ YÜKSEK PERFORMANSLI AĞLARI HEDEFLEYEN TOPLU İLETİŞİM ALGORİTMALARININ DEĞERLENDİRİLMESİ

Çağlayan DÖKME

Yüksek Lisans, Bilgisayar Mühendisliği

Danışman: Doç. Dr. Kayhan Mustafa İMRE

Ocak 2025, 90 sayfa

Bu tez, ağ içi bilgi işlemenin (INC), Fat-Tree ağ topolojisi kullanan paralel bilgi işlem sistemlerinde toplu iletişim operasyonlarının performansı üzerindeki etkisini araştırmaktadır. INC, bilgi işleme görevlerini ağ anahtarlarına aktararak, veri yoğun ortamlarda iletişim verimliliğini artıracak bir yöntem sunar. Bu çalışma, geleneksel metot ve INC'nin etkin olduğu metot altında bariyer senkronizasyonu, yayınlama, dağıtma, toplama ve indirgeme gibi temel kolektif operasyonların performansını karşılaştırarak zamanlama maliyeti, bant genişliği kullanımı ve senkronizasyon kalitesi gibi ölçümleri değerlendirmektedir. Sonuçlar, INC'nin kolektif operasyonlarda zamanlama maliyetini ve bant genişliği kullanımını önemli ölçüde azalttığını ve özellikle büyük ölçekli ağlarda gelişmiş ölçeklenebilirlik ve senkronizasyon tutarlılığını sağladığını ortaya koyuyor. Bu çalışma, INC'nin paralel bilgi işlem sistemlerinde kolektif iletişimi optimize etme potansiyelini öne çıkarıyor ve yüksek performanslı bilgi işlem altyapılarında performansı ve ölçeklenebilirliği artırmak için umut verici bir yol sağlıyor.

Anahtar Kelimeler: ağ içi hesaplama, paralel işleme, toplu iletişim

ACKNOWLEDGEMENTS

This thesis is lovingly dedicated to my mother, Hasibe, whose unwavering support and endless encouragement have been a guiding light throughout my journey. To my wife-to-be, Şirin, whose patience, love, and belief in me have been a constant source of strength and inspiration. And to my entire family, whose sacrifices, wisdom, and kindness have shaped me into the person I am today. This work stands as a testament to your profound impact on my life and my heartfelt gratitude for your presence every step of the way.

CONTENTS

Page

TABLES

Page

FIGURES

ABBREVIATIONS

INC : **I**n-**N**etwork **C**omputing

1. INTRODUCTION

Parallel computing systems are designed to perform complex computations by dividing tasks into smaller subtasks that can be executed simultaneously across multiple processing units. This approach significantly enhances computational speed and efficiency, making parallel computing an essential paradigm for high-performance computing applications. By leveraging the collective power of multiple processors, parallel computing systems can tackle large-scale problems that would be infeasible for a single processor to handle.

In parallel computing systems, collective communication operations are crucial for coordinating and synchronizing the tasks performed by different processors. These operations, such as barrier synchronization, broadcast, scatter, and gather, facilitate data exchange and ensure that all processors work cohesively towards a common goal. Efficient collective communication is vital for maintaining the performance and scalability of parallel computing systems.

Network topology plays a pivotal role in the performance of parallel computing systems. Fat-tree networks are a popular choice due to their hierarchical structure and high bisection bandwidth, which help mitigate congestion and ensure efficient data transfer. In a fat-tree network, switches are arranged in a tree-like structure with multiple paths between any two nodes, providing redundancy and improving fault tolerance. Figure 1.1 shows an example fat-tree constructed with 4-port switches.

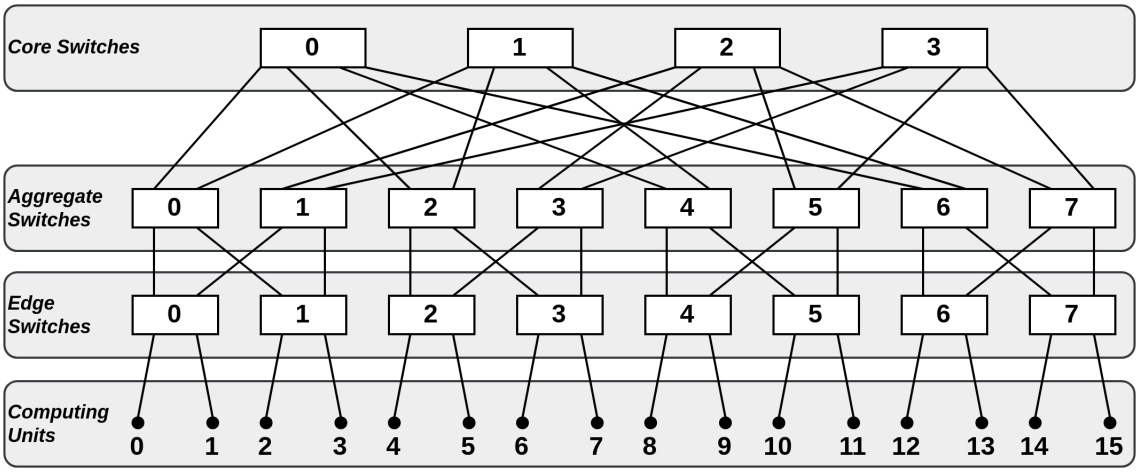

Figure 1.1 4-port Fat-tree

Despite the advantages of parallel computing, these systems often encounter bottleneck problems due to dense data transfer between computing units. As the number of processors increases, the volume of data exchanged during collective communication operations can lead to network congestion, increased latency, and reduced overall system performance. Addressing these bottlenecks is critical for optimizing the efficiency and scalability of parallel computing systems.

1.1. Scope Of The Thesis

This thesis aims to explore the potential of in-network computing to enhance the performance of parallel computing systems. By integrating computational tasks within the network fabric, the study hypothesizes that significant improvements in system throughput, latency, and overall efficiency can be achieved. A custom-designed simulator will be used to emulate parallel computing units and programmable network switches interconnected through a fat-tree network topology. The study will delve into the advantages of in-network computing techniques for handling collective communication operations.

1.2. Contributions

While previous studies have extensively explored various approaches to optimize collective communication operations, our work diverges by specifically focusing on the implementation and evaluation of in-network computed collective operations within a detailed simulator environment. Unlike traditional methods that primarily address improvements at the node level or propose algorithmic enhancements, our research leverages the processing capabilities of network switches to distribute the computational load, thereby mitigating bottlenecks inherent in root-based synchronization schemes. Furthermore, while some recent works have touched upon the potential of in-network computing, they often lack comprehensive performance evaluations or detailed comparisons with traditional methods. Our study fills this gap by providing a thorough comparative analysis, supported by simulation results, that highlights the tangible benefits of in-network computation in terms of synchronization time, message overhead, and overall system efficiency. This focus on in-network computation, coupled with a robust methodological framework, sets our research apart and underscores its potential impact on the design of future high-performance computing systems.

In the context of this thesis, **traditional methods** refer to the conventional approach to collective communication in parallel computing systems, where the network serves solely as a data forwarding medium without any computational capability. In these methods, all data processing, aggregation, or synchronization tasks are carried out exclusively by the computing nodes. For example, in a scatter or gather operation, each node either sends or receives data directly to or from other nodes or a designated root node, without assistance from intermediate network devices. This approach often leads to increased network congestion and higher latency, especially as the number of nodes or the volume of data increases. The limitations of these traditional methods form the basis for exploring in-network computing as a more efficient alternative.

1.3. Organization

This thesis is structured as follows:

- **Chapter 2: Background Overview** provides foundational concepts and relevant technical details essential to understanding the proposed approach, including key aspects of in-network computing and parallel processing.
- **Chapter 3: Related Work** surveys existing research in the field, focusing on approaches that have previously explored in-network computing for collective operations and performance improvements in parallel and distributed systems.
- **Chapter 4: Proposed Method** introduces the core of this research. It begins with the Simulator Design and its components, including Ports, Switches, and Computing Nodes, followed by Network Setup and Time Advancement strategies. Subsections detailing operational flows—such as Direct Message, Barrier, Broadcast, Scatter, Gather, Reduce, and All-reduce—demonstrate how each collective communication function is integrated within the simulator.
- **Chapter 5: Experimental Results** presents performance metrics collected from experimental simulations, divided into Timing Costs, Bandwidth Usage, and Completion Time Difference for each collective communication operation, as well as Direct Message and Matrix Multiplication. Each category assesses how the proposed method compares to traditional techniques, using specific metrics to highlight improvements in latency, bandwidth, and overall completion time.
- **Chapter 6: Conclusion** summarizes the findings, discusses their significance, and suggests potential directions for future work based on the insights gained from this research.

This organizational structure ensures a logical flow from background concepts and literature review to detailed descriptions of the proposed method, experimental analysis, and concluding insights.

2. BACKGROUND OVERVIEW

Parallel computing is a high-performance computing paradigm where large-scale problems are solved by dividing tasks across multiple processing units or computing nodes allowing simultaneous execution. This approach enables faster processing of data-intensive tasks, making it essential for applications in fields such as scientific simulations, machine learning, and real-time data analysis. By spreading the workload across several nodes, parallel computing increases computational efficiency and reduces processing times. However, as the number of nodes increases, efficient communication between them becomes critical to maintain performance, especially when nodes must frequently share intermediate results.

Collective communication operations are fundamental in parallel computing, allowing data exchange and synchronization across nodes. Key operations include **broadcast**, **scatter**, **gather**, **reduce**, and **barrier synchronization**. Each of these operations facilitates specific data transfer and processing tasks essential for parallel algorithms. For instance, a **broadcast** operation shares data from one node to all others, while **gather** collects data from multiple nodes to a single destination. These operations, particularly in large systems, face challenges such as **latency**, **synchronization overhead**, and **bandwidth limitations**. As more nodes interact, these factors can cause bottlenecks, slowing down overall performance.

In-Network Computing (INC) addresses some of these communication challenges by enabling network devices, such as switches and routers, to process data. Traditionally, network switches serve only as data-forwarding devices; however, INC integrates computation into the network layer itself. This approach reduces the number of communication steps and minimizes data movement across nodes by performing certain computations as data travels through the network. By reducing data traffic and communication delays, INC enhances the performance of collective communication operations, particularly in data-intensive environments. INC is especially beneficial in applications requiring frequent, high-volume data exchanges, where it reduces both the latency and the overall processing load on compute nodes.

In traditional parallel systems, all computation occurs within computing nodes, while network devices focus solely on data forwarding. As a result, communication costs increase with the number of nodes due to higher data volumes moving through the network. In contrast, **INC-enabled systems** introduce active computation within network switches, transforming them into intermediate computation points for collective operations. This shift allows INC-enabled systems to handle specific parts of data aggregation and transformation tasks directly within the network. For example, during a **reduce operation**, INC-enabled switches may combine partial results from multiple nodes, reducing the amount of data reaching the final destination. This change enhances efficiency and minimizes communication overhead.

Network topology plays a critical role in shaping communication patterns and efficiency in parallel systems. Standard topologies include **torus**, **hypercube**, and **fat-tree**. Each topology offers unique trade-offs between data path redundancy, latency, and communication load balancing.

The **fat-tree topology**, often used in large-scale data centers and high-performance computing (HPC) systems, is a hierarchical network structure with multiple levels of switches, organized in layers: edge, aggregate, and core. This topology minimizes network congestion by providing multiple paths for data transmission between nodes. Each layer serves a specific function, from connecting nodes at the **edge** level to directing traffic across **aggregate** and **core** layers. Fat-tree networks are well-suited for INC, as they offer multiple data paths, enabling efficient data routing and handling even in large systems. Figure 1.1 shows an example of fat-tree topology.

INC fundamentally changes the way collective communication operations are executed. Each operation benefits differently from INC's ability to distribute computation across network layers.

1. **Scatter and Gather:** In a traditional scatter, each node sends data directly to others. With INC, edge switches handle data distribution, reducing the number of messages

in the network. For gather operations, INC enables intermediate aggregation, allowing switches to collect data on its way to the target, reducing data transfer costs.

2. **Reduce:** INC also optimizes reduce operations by allowing intermediate reductions within network switches. Edge switches combine data from multiple nodes before forwarding, while core and aggregate switches continue to reduce data as it moves toward the destination.

3. **Barrier Synchronization:** Barriers are required to ensure all nodes reach synchronization points before proceeding. INC-enabled barriers allow edge and aggregate switches to handle local synchronization, reducing the overall time for all nodes to reach the barrier.

Recent research in INC has explored advanced ways to optimize data processing within network switches, such as programmable data planes and switch-level aggregation algorithms. Studies show that INC can significantly improve performance in network-bound applications, particularly in reducing the time and bandwidth required for collective communication. However, several challenges remain. For example, integrating INC into existing network infrastructures can be complex, requiring specialized hardware and software support. Furthermore, programming INC-capable devices poses new challenges, as current development frameworks for switch-based processing are less mature than traditional programming environments. Future directions include enhancing software-defined networking (SDN) integration to control and optimize INC functions dynamically and expanding INC's applicability to a broader range of network topologies and applications.

3. RELATED WORK

In recent years, in-network computing (INC) has emerged as a transformative concept in computing architectures, primarily applied within data centers and distributed applications but gradually expanding to parallel computing systems. INC promises enhanced performance, reduced latency, and optimized resource utilization, providing significant advantages for various computational domains. This section reviews existing research on in-network computing and its application to collective operations, highlighting how the present work extends these contributions.

Several well-established libraries, such as MPI (Message Passing Interface) and Open-SHMEM, form the foundation for collective communication in parallel processing clusters. Researchers have incorporated INC concepts to enhance the efficiency of these libraries. For example, Venkata et al. advanced the OpenSHMEM library by integrating INC, achieving faster collective operations within parallel computing frameworks, Chen et al. introduced a hybrid method that combines INC with point-to-point messages, optimizing MPI collectives by balancing computational loads across network nodes [1] [2].

Collection operations like all-reduce are essential for data-intensive applications. Liu et al. designed a scalable hardware architecture specifically for all-reduce operations in INC settings, showing performance benefits achievable with dedicated hardware optimizations [3]. Yang et al. explored aggregation for INC, providing insights into performance gains with hardware-supported data handling techniques [4]. While these studies focus on optimizations, they underscore the need for INC techniques tailored to software-defined environments in high-performance computing.

Thakur et al.'s work on collective communication optimization is a foundational study, identifying efficient communication strategies that enhance scalability—a goal that aligns with INC-enabled systems designed to offload computations across network layers. Their findings illustrate key strategies that can be leveraged within INC frameworks to reduce bottlenecks and maintain synchronization consistency [5].

Given that one of the primary purposes of in-network computing is to reduce network traffic, a focus on optimizing frequent requests becomes inevitable. Tokusashi et al. elucidate the advantages and intricacies of leveraging in-network computing for caching mechanisms, offering insights into enhancing data retrieval and access patterns through strategically implemented in-network caching strategies [6].

While in-network computing enhances system performance by incorporating on-path packet analysis during collective communication, its overall impact on system performance remains a subject of exploration. Yang et al. introduce a comprehensive performance model tailored for in-network computing systems. Through meticulous case studies, the authors elucidate the underlying factors influencing system performance and provide a structured framework for evaluating and optimizing in-network computing architectures [7].

Sapio et al. argue that although constrained by network architecture, INC can be highly effective when applied to specific tasks. They propose Daiet, a system that achieves significant data reduction and reduced worker computation time, especially for machine learning and graph analytics. Their results highlight the potential for in-network aggregation to reduce congestion while achieving high throughput [8].

Kumar et al. focus on optimizing MPI collective operations —specifically MPI Broadcast, MPI Reduce, and MPI AllReduce— for fat-tree networks in InfiniBand clusters. They introduce multi-color k-ary trees with a unique mapping scheme, achieving improved link utilization and overall performance compared to traditional methods. Their hybrid approach and Multi-Leader techniques further optimize clusters with SMP nodes, showing significant gains in micro-benchmarks on POWER8 and X86 clusters, as well as application performance improvements in PARATEC and QBOX [9].

Imre's simulation work on programmable networks in distributed highlights the scalability of INC-enabled systems through the Greatest Available Logical Time (GALT) algorithm. His findings indicate that INC can offload time synchronization tasks within a network, presenting a unique use case for INC in distributed time-sensitive applications [10].

This thesis builds on these studies by investigating INC's impact on broadly used collective operations in parallel systems, using a fat-tree topology to handle large-scale computations effectively. Unlike previous works that focus on either hardware-only solutions or specific algorithms, this work emphasizes a software-defined, scalable approach to INC, demonstrated through a custom simulator and detailed comparative analysis. This focus on INC's software applications fills a notable gap, illustrating the practical benefits and challenges of INC in parallel computing environments.

4. PROPOSED METHOD

This section details the methodology used to implement in-network computing within a fat-tree topology simulator, aiming to evaluate its impact on collective communication operations in parallel computing systems. Each component of the simulator is described, including its function and relevance to the overall system design.

The methodology of this study revolves around the development of a custom simulator designed to evaluate the impact of in-network computing on parallel computing systems. This section details the design and implementation of the simulator, including bi-directional ports, switch types, computing nodes, and the interconnections within a fat-tree network topology. Furthermore, it describes the creation and processing of messages that represent collective communication operations.

4.1. Simulator Design

The simulator emulates a fat-tree network architecture with INC capabilities to replicate and analyze the behavior of parallel computing systems during collective operations. Each component of the simulator, including ports, switches, and computing nodes, is designed to support efficient data transmission and processing, thereby providing a realistic environment to evaluate the effects of INC.

The interconnection of switches and computing nodes follows the fat-tree topology, which provides a scalable and efficient network structure. The design involves creating a hierarchical arrangement where each layer of switches is interconnected, and edge switches are connected to computing nodes. The implementation requires configuring the network layout, establishing connections between ports, and ensuring that data paths are correctly established. Figure 1.1 visualizes the most basic fat-tree in which each switch has 4 ports.

4.1.1. Ports

Network ports are essential for representing the connections between different layers of the fat-tree network, either between switches or between a switch and a computing node. They facilitate the exchange of data. Ports are designed to be bi-directional and have message queues for both directions. Messages carried through these ports are delayed proportional to their size, simulating realistic transmission times. The implementation of these ports includes mechanisms for handling data packets, managing queues efficiently, and ensuring data integrity during transmission. In Figure 1.1, ports are represented as lines between units.

4.1.2. Switches

Switches form the core of the INC-enabled network. Divided into edge, aggregate, and core layers, switches handle data routing and, in the case of INC, perform partial computation tasks to reduce traffic and processing loads on end nodes. This structure enables efficient aggregation and distribution of data, essential for collective operations. Each switch type serves a specific role within the fat-tree topology and is designed with unique functionalities to handle various communication tasks. Each switch includes ports that are bi-directionally connected to the ports of either another switch or a computing node. The number of ports in a switch must be even and greater than 2, *e.g. 4, 6, .., 2k*.

- **Core switches** are positioned at the top layer and establish connection between groups. They only have down-ports.
- **Aggregate switches** are located in the intermediate layer and connect core switches to edge switches. They act as intermediaries, facilitating efficient data flow between the upper and lower layers of the network.
- **Edge switches** are situated at the bottom layer and interface directly with computing nodes.

Figure 1.1 visualizes the layers formed by different types of switches.

4.1.3. Computing Nodes

Computing nodes are the primary processing units in the simulator, representing the individual processors in a parallel computing system. Each node is designed to execute computational tasks, send and receive messages, and participate in collective communication operations. Each node has only one port connected to the corresponding Edge switch. Computing nodes can be equipped with algorithms to simulate computational workloads and measure performance metrics such as execution time and communication overhead. The number of computing nodes depends on the port amount on switches.

Messages are central to simulating collective communication operations in the network. Each message represents a specific operation such as barrier synchronization, broadcast, scatter, or gather. The design includes defining message types with attributes such as source, destination, payload, and operation type. Message processors are implemented on both switches and computing nodes to handle incoming messages, perform necessary computations, and forward messages as required. The implementation involves creating message objects and corresponding processor methods to simulate the flow of data and control signals within the network. These processors are designed to handle collective communication efficiently, ensuring that operations are executed correctly and performance metrics are accurately recorded.

4.2. Network Setup

The network setup is a crucial step in configuring the simulator to accurately reflect the architecture and behavior of a fat-tree network topology. The primary parameter driving the setup is the number of ports per switch, denoted as k. This parameter must be an even number greater than 2, and it determines the number of switches at each layer of the network and the number of computing nodes. Using the aforementioned parameter, the number of switches and computing nodes in the network can be derived as follows:

- Core Switches: $\frac{k^2}{4}$

- Aggregate Switches: $\frac{k^2}{2}$
- Edge Switches: $\frac{k^2}{2}$
- Computing Nodes: $\frac{k^3}{4}$

The interconnection of switches and computing nodes follows a systematic process to ensure the fat-tree topology is correctly established:

1. Each computing node is connected to the down port of a corresponding edge switch. This connection establishes the base layer of the network, where the computing nodes interact directly with the edge switches.
2. The up-ports of edge switches are connected to the down-ports of aggregate switches. This step creates the intermediate layer, enabling communication between the computing nodes and the higher layers of the network.
3. The up-ports of aggregate switches are connected to the down-ports of core switches. Core switches, positioned at the top layer, only have down-ports and serve as the primary routing points for data across different groups of switches.
4. After establishing all connections, the network setup process includes a validation step to ensure that all switches and computing nodes are correctly interconnected. This involves checking each port to confirm that it is properly linked to the intended switch or computing node.

4.3. Time Management Technique

The simulator employs a time-increment-based time management technique to coordinate the operations of the network and computing nodes in a synchronized manner. Unlike event-based techniques, which advance time only when significant events occur, the time-increment-based approach progresses in fixed, discrete time steps, or ”ticks.” This method is particularly suitable for the simulator, as it minimizes the likelihood of idle *(i.e.*

jobless) ticks during execution, ensuring that the simulation remains active and productive at every step.

Main Network Thread: The core of the simulator's timing mechanism is a tick counter maintained in the main network thread. This counter advances in discrete time steps, or ticks, and drives the simulation forward. At each tick, the simulator calls the tick method for every switch object and computing node object in the network. This periodic invocation ensures that all components of the system are given the opportunity to process messages and perform their respective tasks in a coordinated manner.

Switch Tick Method: During each tick, the tick method in each switch object is responsible for checking the ports for new messages. If a message is detected on any port, the switch processes the message according to its type and the switch's role within the fat-tree topology. This processing may involve routing the message to another port or performing a computation.

Computing Node Tick Method: Similarly, the tick method for each computing node object checks for incoming messages on its connected ports. When a message is detected, it is enqueued into an internal queue managed by the computing node's separate thread. This design allows computing nodes to handle messages asynchronously, ensuring that the parallel computation tasks can proceed without being blocked by network communication delays.

Parallel Execution and Synchronization: The separation of the network and computing nodes into different threads allows the simulator to realistically model the parallel execution of tasks. The main network thread advances the global tick counter, while the individual computing node threads execute their respective tasks and process messages concurrently. Synchronization mechanisms ensure that the state of the network and computing nodes remains consistent, enabling accurate simulation of collective communication operations.

Message Handling and Processing: The processing of messages within the tick methods is central to the simulator's operation. Switches can either/both route or/and process messages based on the network topology and the specific communication operation being simulated

such as aggregating data during a gather operation or distributing data during a scatter operation. Computing nodes perform the necessary computations associated with each message.

This time advancement mechanism, combining a single-threaded network with multi-threaded computing nodes, provides a robust framework for simulating the performance and efficiency of parallel computing systems enhanced by in-network computing techniques.

4.4. Operational Flows

The integration of computing capabilities into the network significantly influences the message flow during collective operations. By leveraging in-network processing, data can be processed and routed more efficiently, reducing latency and optimizing resource utilization. In this section, we detail the operational flows for key collective communication operations, highlighting how the network's computational power reshapes traditional message patterns and enhances overall system performance. The collective operations covered include:

- Barrier
- Broadcast
- Scatter
- Gather
- Reduce
- All-Reduce

4.4.1. Barrier

This section describes the flow of a barrier operation in both traditional and in-network computing (INC) contexts. A barrier operation ensures synchronization across all

participating computing nodes, preventing any node from proceeding until all nodes reach the barrier point. This operation is essential in parallel computing systems to maintain execution consistency across distributed tasks.

In traditional parallel computing systems, a barrier operation typically involves the following steps:

1. **Request Collection:** Barrier request signals propagate in a hierarchical, tree-like fashion, starting from the leaf nodes and moving towards a designated root node. The left-side nodes aggregate and forward the barrier requests, while the root node, upon receiving all requests, determines when to release the barrier. Figure 4.1 illustrates this step.

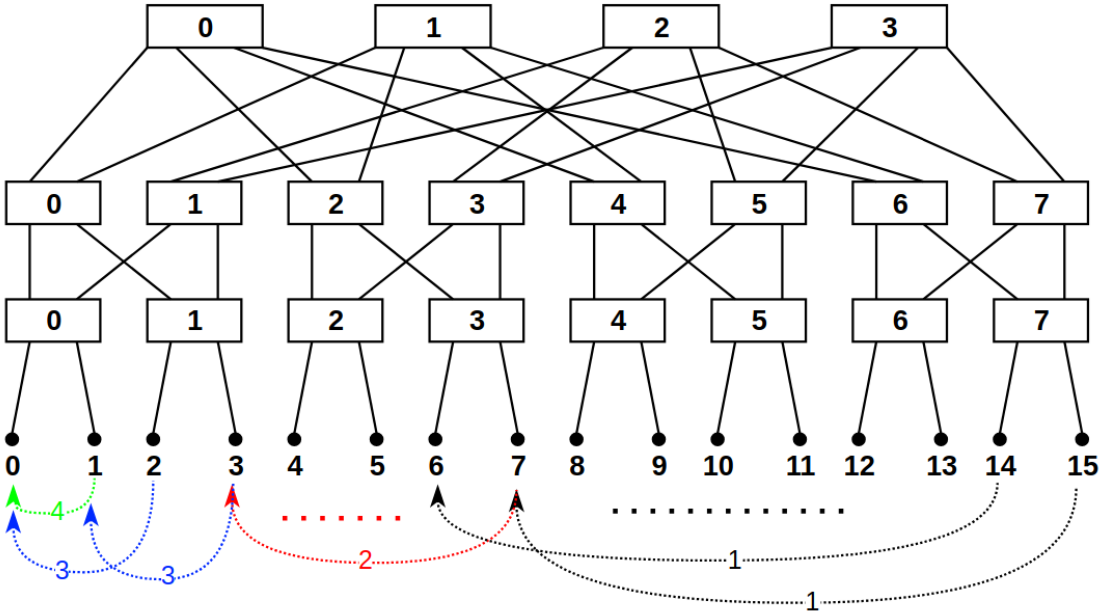

Figure 4.1 Traditional Barrier Request Collection

2. **Barrier Release:** Once all barrier requests are collected at the root, it sends a release signal to all nodes, allowing them to continue. The release signal may also propagate in a tree-like manner to optimize network traffic, with intermediate nodes re-transmitting the signal. Figure 4.2 demonstrates this release process.

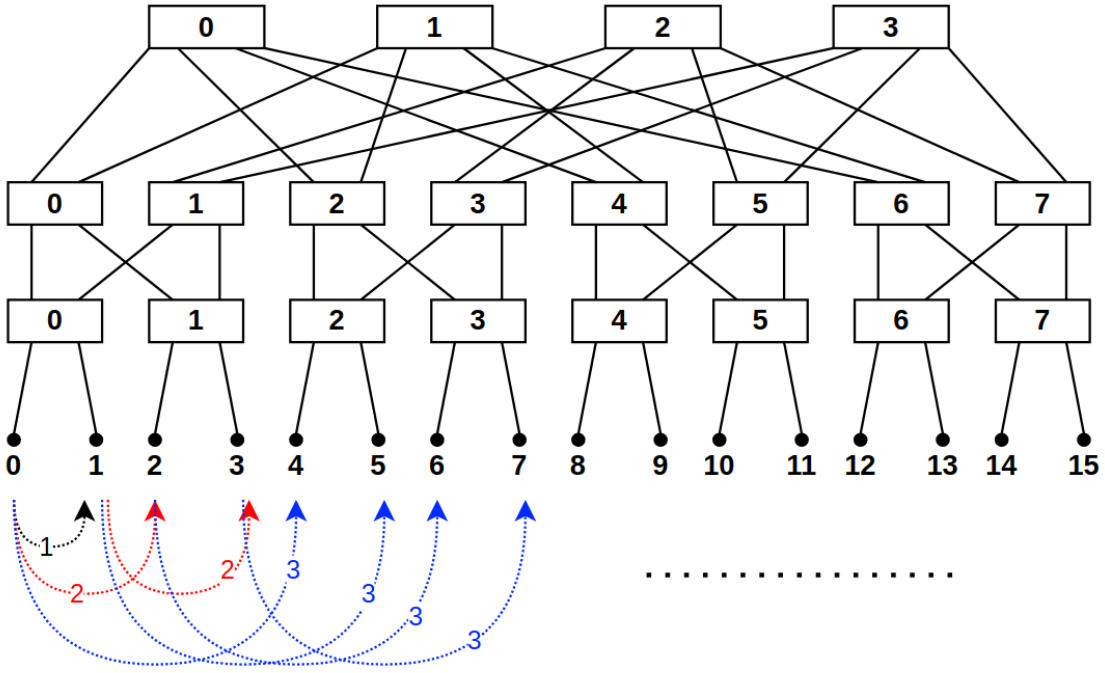

Figure 4.2 Traditional Barrier Release

With in-network computing, the barrier operation leverages the processing capabilities of network switches to reduce bottlenecks and improve efficiency. Here's how the message flow is handled in the simulator:

1. **Local Barrier Requests:** Each computing node sends its barrier request to its connected edge switch.

2. **Switch-level Aggregation:** Edge switches wait to receive barrier requests from all their down-ports (connected computing nodes). Once all requests are collected, the edge switch forwards a single aggregated request to its up-port, connecting to an aggregate switch. Aggregate switches follow the same process, collecting requests and forwarding them upward.

3. **Core Switch Processing:** Core switches, positioned at the top of the hierarchy, collect barrier requests from all their ports.

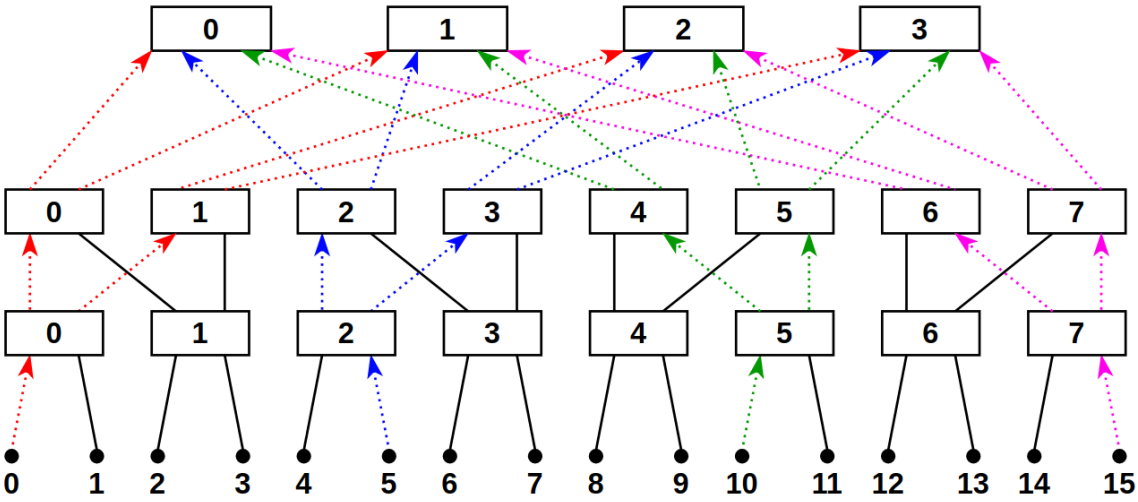

Figure 4.3 In-network Computed Request Collection

4. **Barrier Release:** Upon receiving requests from all ports, core switches initiate the release process by broadcasting a barrier release signal to all connected aggregate switches.

5. **Cascading Release:** Aggregate switches wait until they receive release signals from all their up-ports. Once received, they propagate the release signal down to the edge switches, which in turn send the signal to their connected computing nodes.

6. **Completion:** Each computing node waits for the barrier release signal from its edge switch before resuming operations. The flow of the release signal is visualized in Figure 4.4.

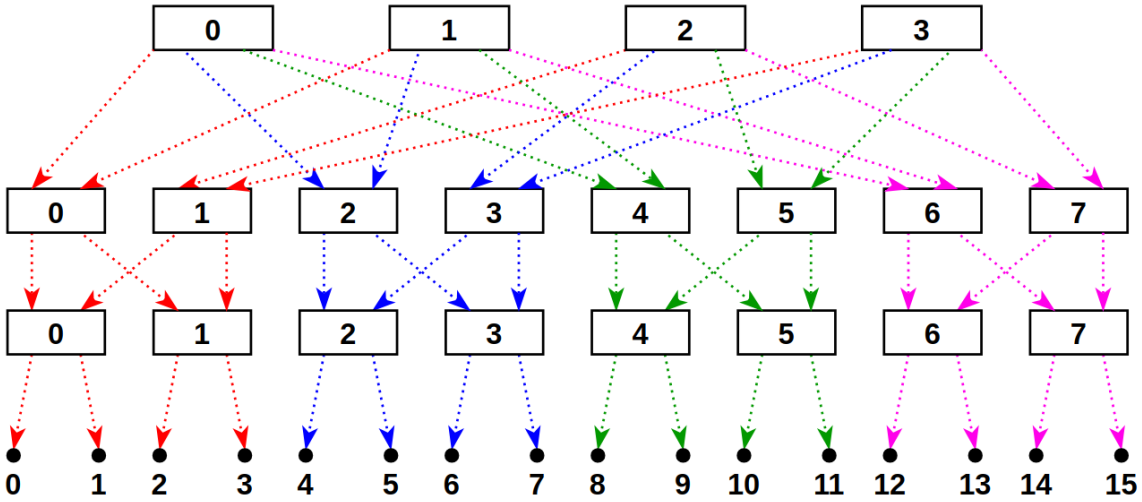

Figure 4.4 In-network Computed Barrier Release

By utilizing in-network computing capabilities, the barrier operation is significantly optimized. Switches handle message aggregation and forwarding at each layer, reducing the communication overhead typically associated with routing all requests to a single node. This approach reduces bottlenecks and improves synchronization, as each level of the network can release the barrier independently, ensuring faster and more efficient communication across the entire system.

4.4.2. Broadcast

The broadcast operation is a fundamental communication primitive used in parallel computing systems to disseminate data from a single source node to all other participating nodes. This operation ensures that a message, such as a control signal or a data payload, is efficiently propagated throughout the network. In traditional systems, this involves multiple message transmissions across the network, often following a tree-like structure. In contrast, in-network computing (INC) enhances the broadcast process by allowing switches to take on part of the data distribution workload, reducing transmission delays and minimizing network congestion. This section describes the flow of the broadcast operation in both traditional and INC-enabled scenarios.

In traditional parallel computing systems, the source node is responsible for sending data directly to all target nodes. This process may involve multiple message transmissions across the network, typically following a tree-based structure, where intermediate nodes forward the data until it reaches all destination nodes. While effective, this method can generate high network traffic, particularly as the number of nodes increases, leading to potential bottlenecks. Figure 4.5 visualizes the operational flow of broadcast operation in traditional systems.

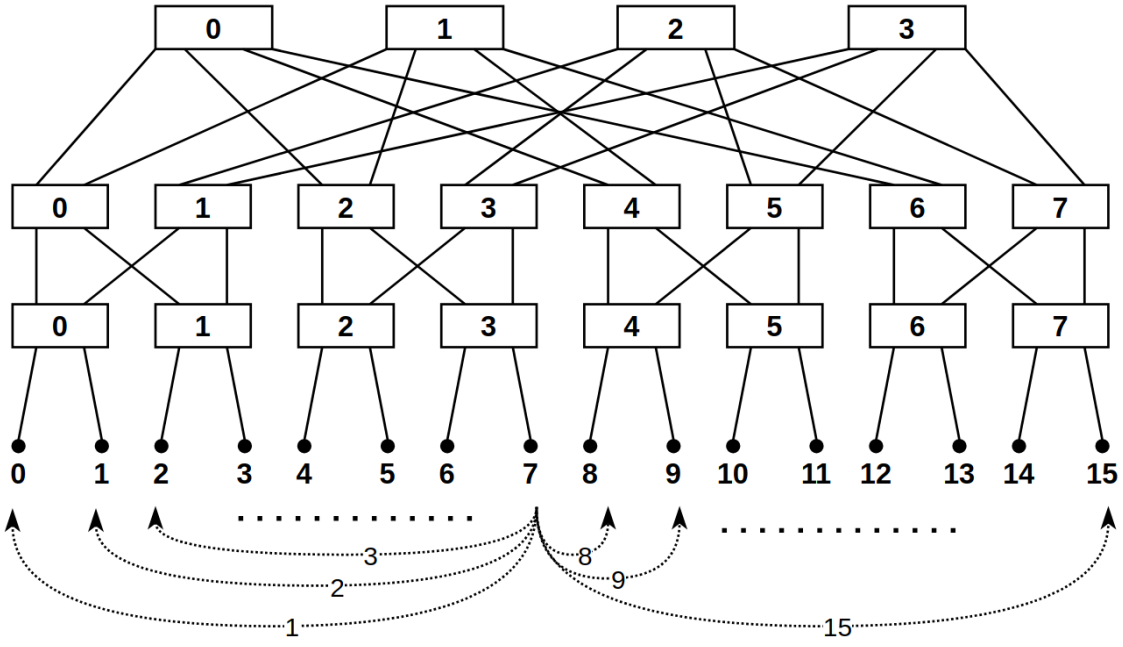

Figure 4.5 Traditional Broadcast

In INC-enabled systems, the broadcast operation leverages the processing capabilities of network switches to improve efficiency. The source node sends the data into the network, and the switches handle most of the distribution, significantly reducing the load on the source node and minimizing network congestion. The flow proceeds as follows:

- **Edge Switches:** When the data is received from a down-port (a connected computing node), the edge switch forwards the data to other nodes connected to its down-ports, as well as to the most available up-port, to propagate the data upward in the network. Conversely, if the data arrives from an up-port (i.e., an aggregate switch), the edge

switch broadcasts it to all down-ports, ensuring that all connected computing nodes receive the data.

- **Aggregate Switches:** Upon receiving data from a down-port (an edge switch), the aggregate switch forwards the data to the other down-ports (other edge switches) and to the most available up-port, sending the data up towards the core switches. If the data arrives from an up-port, the aggregate switch broadcasts it to all down-ports, distributing the data to all edge switches connected below it.
- **Core Switches:** At the top of the hierarchy, core switches receive the data from one of their ports and immediately broadcast it to all other ports, ensuring the data reaches every part of the network.

Figure 4.6 depicts the broadcast operation in in-network computing-enabled systems.

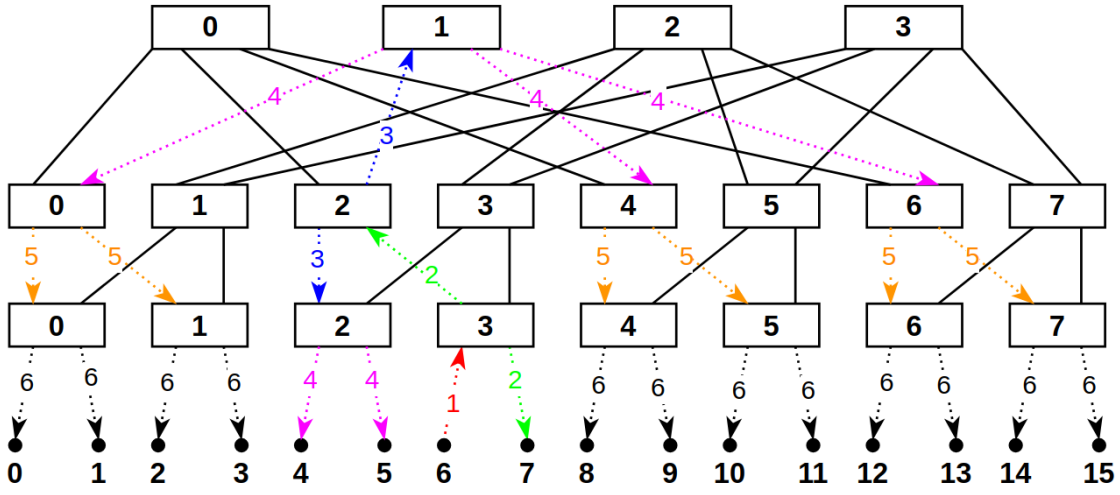

Figure 4.6 In-network Computed Broadcast

By offloading much of the data distribution work to the network switches, the INC-enabled broadcast reduces the overall number of transmissions and the load on the source node. This approach helps avoid network congestion and speeds up the dissemination of information, particularly in large-scale systems.

4.4.3. Scatter

The scatter operation is used to distribute distinct chunks of data from a single source node to multiple target nodes. It is commonly employed in parallel computing when the source node has a large dataset that must be divided and sent to different nodes for processing. The flow of this operation varies significantly between traditional and in-network computed (INC) systems.

In traditional systems, the source node is responsible for sending each chunk of data directly to its corresponding target node. The source node sequentially sends out data chunks, and each target node waits to receive its specific portion. While simple in concept, this approach can result in bottlenecks at the source node, as it needs to manage multiple outgoing transmissions, especially in large-scale systems. Figure 4.7 illustrates the operational flow of a scatter operation in traditional parallel computing systems.

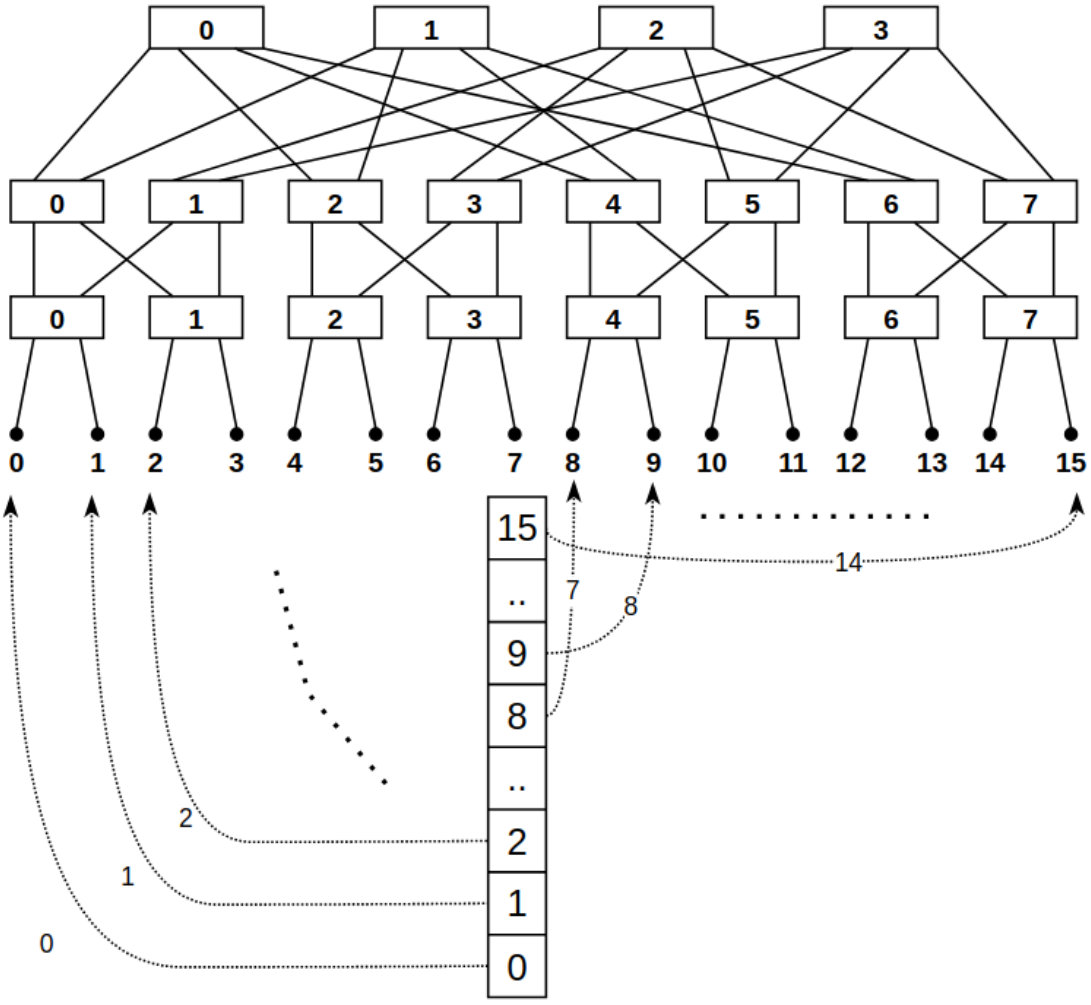

Figure 4.7 Traditional Scatter

In INC-enabled systems, the scatter operation offloads much of the data distribution to the network switches. The source node sends the entire dataset to its edge switch, and the switches take over the task of forwarding the data chunks to the appropriate target nodes, improving efficiency. The flow proceeds as follows:

- **Edge Switches:** If the data is received from a down-port (a connected computing node), the edge switch distributes the corresponding chunks to other nodes connected to its down-ports and forwards the remaining data to the most available aggregate switch. If the data arrives from an up-port (an aggregate switch), the edge switch

scatters the data to all its down-ports, ensuring each connected computing node receives its chunk.

- **Aggregate Switches:** When the data arrives from a down-port (an edge switch), the aggregate switch distributes the corresponding chunks to its down-ports (other edge switches) and sends the rest of the data to the most available core switch. If the data comes from an up-port (a core switch), the aggregate switch scatters it to all down-ports, distributing the data to the edge switches below.
- **Core Switches:** At the core of the network, the core switches receive the data from one of their ports and divide the dataset, sending each chunk to its corresponding port based on the target node's location. This ensures efficient distribution of data across the network.

Figure 4.8 depicts the operational flow of a scatter operation in in-network computation-enabled parallel computing systems.

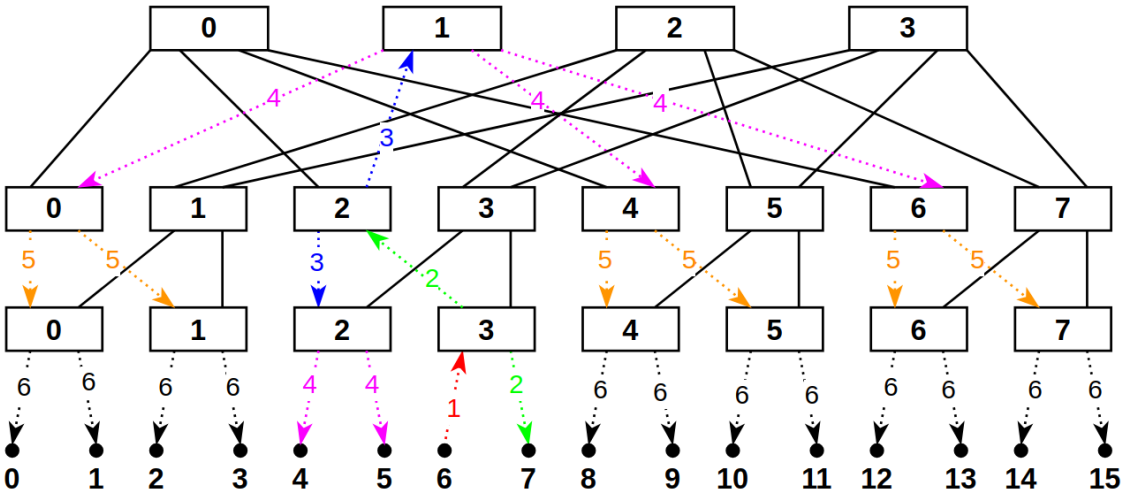

Figure 4.8 In-network Computed Scatter

In this INC-enabled approach, the burden of scattering the data is distributed across the network, reducing the workload on the source node and improving overall performance. By leveraging the processing capabilities of the switches, data is transmitted more efficiently,

minimizing network congestion and accelerating the scattering process, especially in large-scale deployments.

4.4.4. Gather

The gather operation is the reverse of scatter, where data from multiple nodes is collected and sent to a single target node. It is frequently used in parallel computing systems where results from distributed computations need to be centralized for further processing. The operation's flow differs significantly between traditional and in-network computed (INC) systems.

In traditional systems, each computing node sends its data directly to the target node. The target node is responsible for receiving and collecting all the data chunks. While this method is simple, it often creates a bottleneck at the target node, as it must handle multiple incoming transmissions, leading to potential delays as the scale of the system grows. Figure 4.9 shows the corresponding flow.

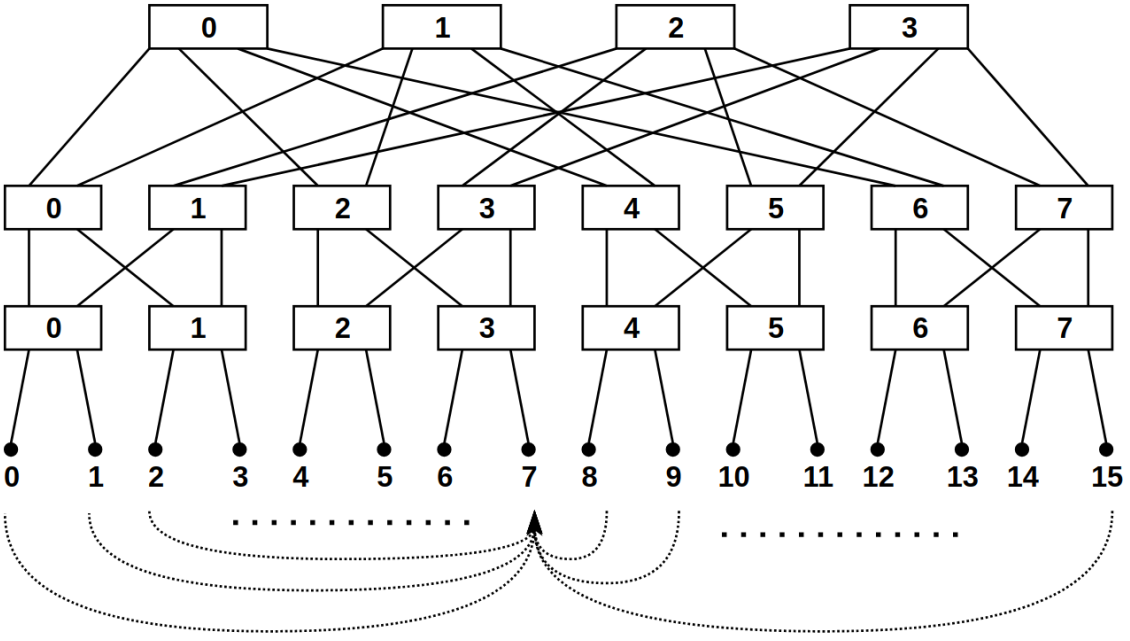

Figure 4.9 Traditional Gather

In INC-enabled systems, the Gather operation is optimized by distributing the aggregation tasks across the edge and aggregate switches to reduce the communication burden on the nodes and minimize network congestion. The INC-enabled Gather operation proceeds as follows:

- **Edge switches** gathers all the received data.
  - If it isn't directly connected to the target node and if all nodes of it have sent their data, the gathered data will be sent to the same-column aggregate switch.
  - If it is directly connected to the target node, it waits for data from the same-group aggregate switches and other directly connected computing nodes. When received and gathered all the data, it sends the result to the target node.
- **Aggregate switches** decide on the action with the destination of the data.
  - If destined to another group, then it just redirects the data without any manipulation on it. Note that only the same-column edge switch can send this type of data.
  - Otherwise, it gathers the incoming data packets.
    - If it's placed on the same column as the target node, it receives from the up-ports only. Notice that it can only receive from the same sub-column indexed computing nodes.
    - Else, it will receive from up-port(s) and from the same column edge switch *(i.e. computing nodes)*.
- Core switches don't contribute to the Gather operation. They only re-direct the data to the appropriate group.

Figure 4.10 illustrates the described flow for in-network computed Gather operation.

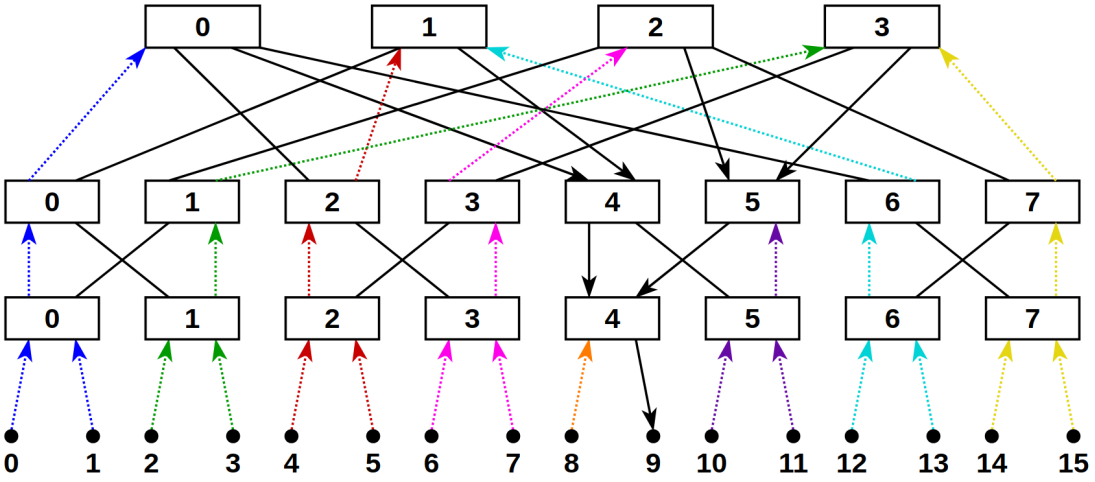

Figure 4.10 In-network Computed Gather

In this INC-enabled approach, the data-gathering task is distributed across the edge switches, reducing the workload on the target node. This results in better performance by preventing bottlenecks and minimizing delays associated with traditional systems. The distributed nature of INC gather allows for faster data aggregation, especially in large-scale systems, as the network itself takes on part of the responsibility for gathering data.

4.4.5. All-Gather

The All-Gather operation is an extension of the Gather operation, where instead of collecting data from all nodes into a single target node, the collected data is shared with all nodes in the network. Each node contributes a portion of the data, and the combined dataset is distributed across all participating nodes. The implementation of this operation varies significantly between traditional parallel computing systems and in-network computed (INC) systems.

In the traditional approach to All-Gather, the operation is typically performed in two sequential steps: gathering and broadcast. First, a root node collects data from all participating nodes, combining them into a single dataset. In the second step, the root node broadcasts the combined dataset back to all nodes in the network. While this approach is more efficient than having all nodes send their data to every other node individually, it still

incurs significant communication overhead, especially as the system scales. The two steps of the traditional All-Gather operation are visualized in Figure 4.9 and Figure 4.5.

In INC-enabled systems, the network itself contributes to the gathering and distribution processes, reducing the communication and computation burden on individual nodes. The All-Gather operation in an INC-enabled system proceeds as follows: INC-Based All-Gather Flow

- **Edge Switches** *(Gather Phase)*: Each computing node sends its data to its respective edge switch through its down-port. The edge switch collects data from all connected nodes (down-ports) and aggregates it into a partial dataset. The combined dataset is sent upward through all up-ports to the aggregate switches.
- **Aggregate Switches** *(Gather Phase)*: The aggregate switches receive partial datasets from all edge switches (down-ports). Once data from all corresponding edge switches is received, the aggregate switch combines these datasets into a larger combined dataset. This aggregated data is sent upward to the core switches through the up-ports. The upward flow of data is shown in Figure 4.11.
- **Core Switches** *(Gather and Distribution Phase)*: The core switches collect data from all aggregate switches (down-ports). Once the full dataset is constructed, the core switches initiate the distribution phase. Each core switch sends the complete dataset back down to all its aggregate switches through its down-ports.
- **Aggregate Switches** *(Distribution Phase)*: Upon receiving the complete dataset from the core switches, the aggregate switches forward the data to all their edge switches through their down-ports. No additional data processing is required at this stage, as the dataset is already complete.
- **Edge Switches** *(Distribution Phase)*: Each edge switch propagates the complete dataset to all its connected computing nodes through its down-ports. The distribution phase for the INC-enabled All-Gather operation is visualized in Figure 4.12.

- **Computing Nodes**: Each computing node receives the complete dataset from its corresponding edge switch. At this point, all nodes in the network have access to the same, fully aggregated data.

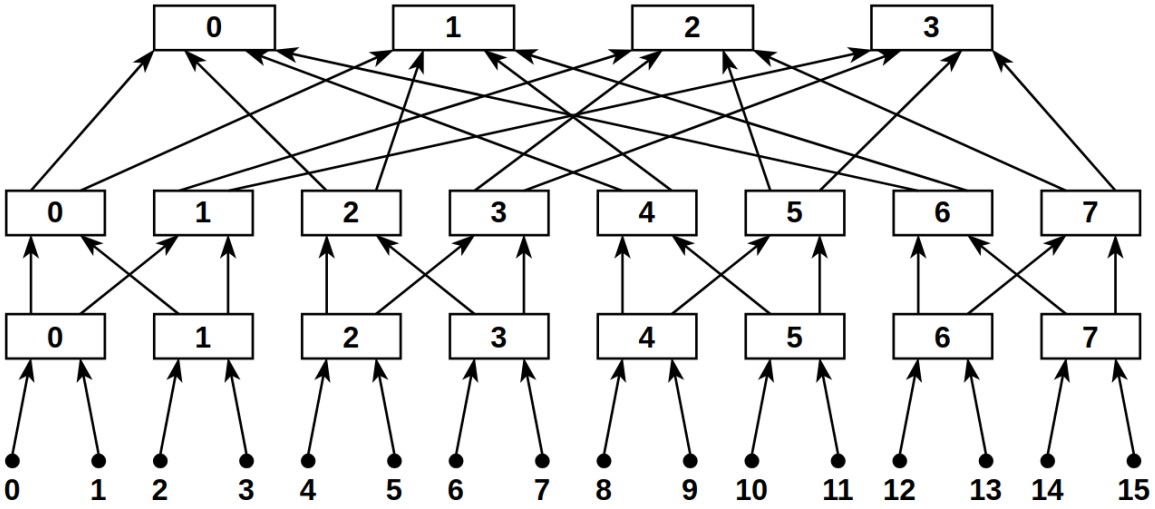

Figure 4.11 In-network Computed All-Gather Gather Phase

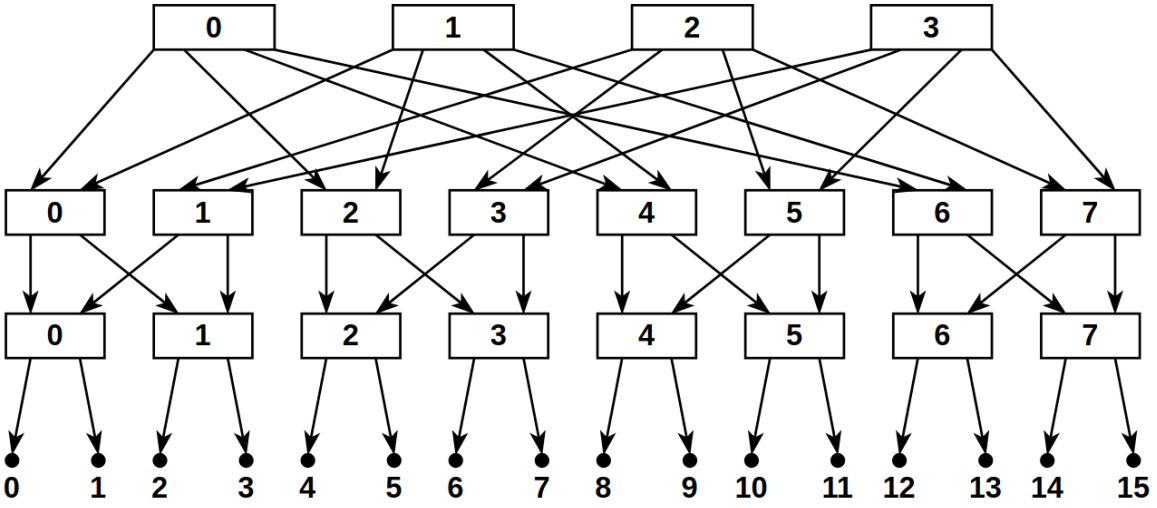

Figure 4.12 In-network Computed All-Gather Distribution Phase

In the INC-enabled All-Gather operation, switches at each network level actively participate in the gathering and distribution processes. This reduces the volume of data transferred at each step and minimizes the communication overhead compared to traditional methods.

By leveraging the hierarchical structure of the fat-tree topology and the computational capabilities of the switches, INC ensures a more efficient and scalable implementation of the All-Gather operation.

4.4.6. Reduce

The Reduce operation is a fundamental collective operation in parallel computing that combines data from multiple nodes using a specified operation *(e.g., sum, min, max)* and sends the result to a target node. The method by which this operation is performed varies significantly between traditional systems and in-network computed *(INC)* systems, with the latter optimizing the process by offloading some computational work to the network switches.

In a traditional Reduce operation, each computing node sends its data directly to the target node, along with the operation type (e.g., sum or average). The target node is responsible for receiving all the data chunks and performing the reduction operation to generate the final result. This centralized approach can create performance bottlenecks, especially as the number of participating nodes increases, due to the volume of data converging on a single node. Figure 4.13 illustrates the traditional operational flow for a Reduce operation.

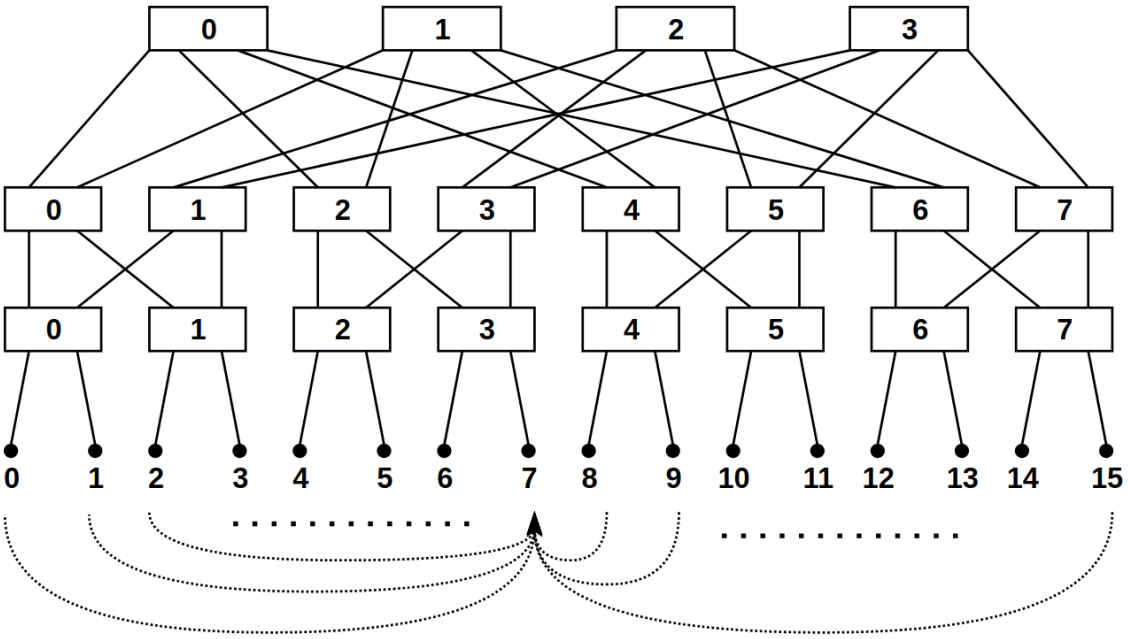

Figure 4.13 Traditional Reduce

In INC-enabled systems, the Reduce operation is optimized by distributing the aggregation tasks across the network switches. This reduces the communication burden on the nodes and minimizes network congestion. The INC-enabled Reduce operation proceeds as follows:

- **Edge switches** reduces all the received data.
  - If it isn't directly connected to the target node and if all nodes of it have sent their data, the reduced data will be sent to the same-column aggregate switch.
  - If it is directly connected to the target node, it waits for data from the same-group aggregate switches and other directly connected computing nodes. When all data is received and reduced, the result is sent to the target node.
- **Aggregate switches** decide on the action with the destination of the data.
  - If destined to another group, then it just redirects the data without any manipulation on it. Note that only the same-column edge switch can send this type of data.

- Otherwise, it reduces the incoming data packets.
    - If it's placed on the same column as the target node, it receives from the up-ports only. Notice that it can only receive from the same sub-column indexed computing nodes.
    - Else, it will receive from up-port(s) and from the same column edge switch *(i.e. computing nodes)*.

- Core switches don't contribute to the Reduce operation. They only re-direct the data to the appropriate group.

Figure 4.14 visualizes the explained flow for in-network computed Reduce operation.

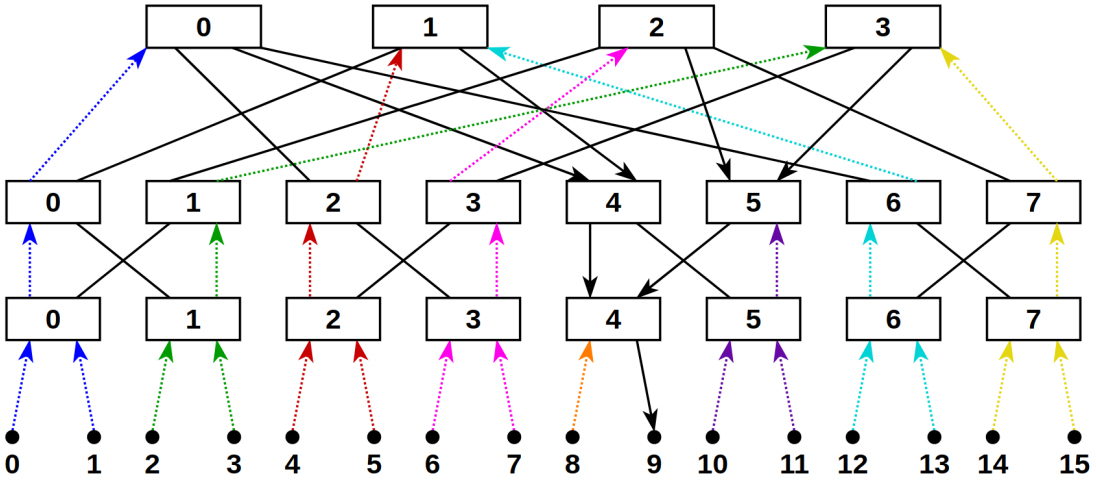

Figure 4.14 In-network Computed Reduce

In this INC-enabled flow, each level of the network contributes to the reduction operation, offloading some of the computational work from the target node and distributing it across multiple switches. This approach leads to better performance by minimizing data congestion and reducing latency, especially in large-scale parallel systems. By performing reductions at each stage of the network, the overall efficiency of the operation is improved, and bottlenecks that are common in traditional systems are significantly mitigated.

4.4.7. All-Reduce

The All-Reduce operation is an extension of the Reduce operation, where instead of sending the reduced result to a single target node, the result is shared with all nodes participating in the operation. Each node contributes data, and the result of the reduction operation *(e.g., sum, min, max)* is made available to all nodes. The implementation of this operation differs significantly between traditional parallel computing systems and in-network computed (INC) systems.

In the traditional approach to All-Reduce, the operation is conducted in two sequential steps: reduction and broadcast. First, a root node is determined, where all participating nodes send their data for reduction. The root node aggregates the data according to the specified operation *(e.g., summing or finding the maximum value)*. In the second step, the reduced result is broadcasted from the root node to all other nodes. This approach is efficient compared to an alternative, less optimal method where each node individually sends its data to all other nodes and performs the reduction locally. The latter method generates significant communication overhead as the number of nodes increases, due to the N2N2 messages exchanged among NN nodes. In Figure 4.13 and Figure 4.5, the two steps of the process are visualized.

In INC-enabled systems, the network itself participates in the reduction process, reducing the communication and computation burden on the individual nodes. The reduction operation is progressively carried out by switches at various levels, ensuring that the reduced data is distributed back to all nodes efficiently. The process can be broken down as follows:

- **Edge Switches:** Each computing node sends its data to its respective edge switch. The edge switch waits for data from all of its connected nodes *(down-ports)*. Once all data has been received, the edge switch performs the reduction operation and sends the reduced result to all up-ports *(i.e., to the aggregate switches)*.
- **Aggregate Switches:** The aggregate switches receive reduced data from all edge switches *(down-ports)*. Once data from all corresponding edge switches is collected,

the aggregate switch performs another reduction on the received data. It then forwards the reduced data to all up-ports *(i.e., to the core switches)*.

- **Core Switches:** The core switches are responsible for reducing the data from all aggregate switches. Once data is received from all down-ports *(i.e., corresponding aggregate switches)*, the core switch performs a final reduction operation. At this point, all core switches hold the same reduced result, ensuring consistency across the network. Figure 4.15 shows the reduction phase for the in-network computed All-Reduce operation. The fully reduced data is then sent back down to all aggregate switches through the corresponding down-ports.

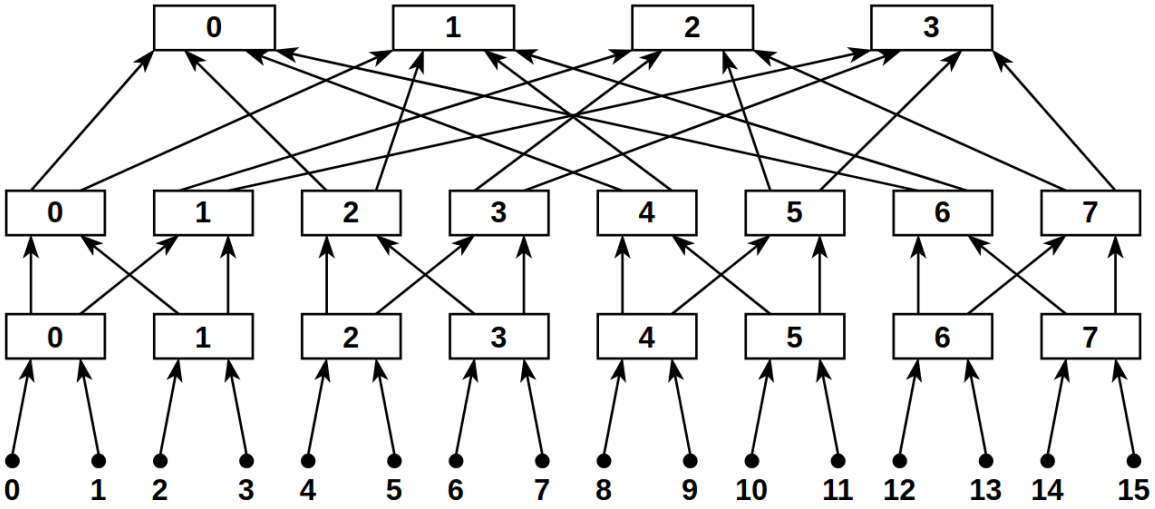

Figure 4.15 In-network Computed All-Reduce Reduction Phase

- **Aggregate Switches *(Downward Flow)*:** Upon receiving the same reduced data from all core switches *(up-ports)*, the aggregate switches propagate the reduced result down to all their edge switches. No further reduction is required at this stage as the data has already been fully reduced.

- **Edge Switches *(Downward Flow)*:** Similarly, the edge switches wait for the same reduced result from all aggregate switches *(up-ports)*. Once the data is received, the edge switches send the final reduced data to all their connected computing nodes *(down-ports)*.

- **Computing Nodes:** Each node receives the fully reduced data from its corresponding edge switch. The result of the All-Reduce operation is now available to all nodes in the network. Figure 4.16 shows the distribution phase for the in-network computed All-Reduce operation.

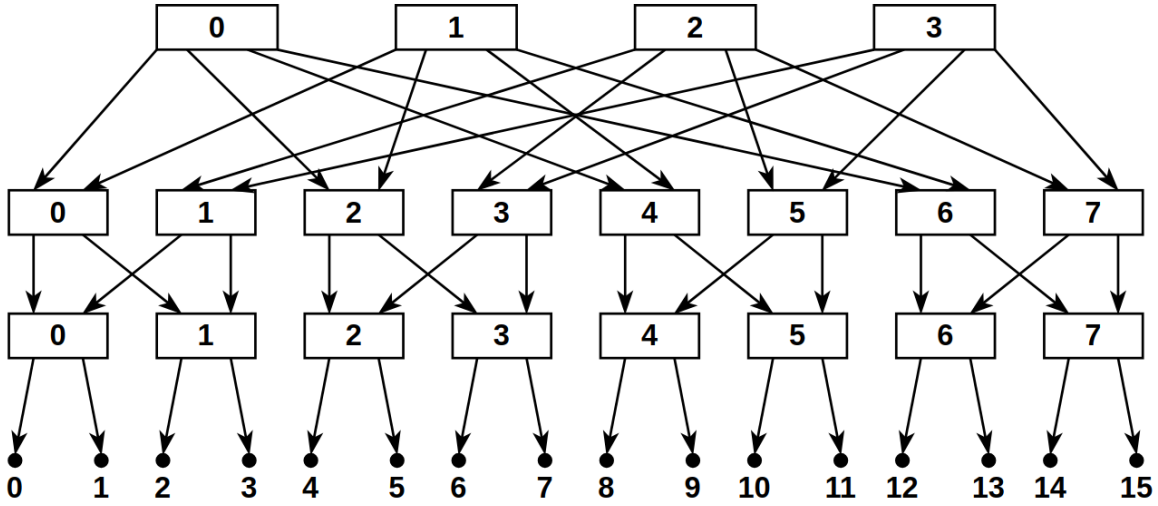

Figure 4.16 In-network Computed All-Reduce Distribution Phase

In this INC-enabled approach, the switches at each level of the network contribute to the reduction and distribution process. This distributed reduction minimizes the volume of data being transferred and ensures that all nodes receive the final reduced result efficiently. By leveraging the network infrastructure for data reduction, the overall operation is significantly optimized, reducing both latency and communication overhead compared to traditional methods.

4.4.8. Direct Message

The flow of direct messages remains unchanged in both traditional and INC-based methods. Direct messages involve simple, point-to-point communication between nodes, where network switches act solely as forwarding devices without performing any computation or processing. Since INC's benefits arise from its ability to process and aggregate data within the network, it does not impact the performance or behavior of direct messages.

4.5. Simulator Implementation

The simulator was developed in C++20 with the full source code accessible at GitHub repository [11]. It utilizes modern syntax and Standard Template Library *(STL)* containers to ensure efficient data management and ease of maintenance. The design incorporates several advanced C++ features and best practices, resulting in a robust and flexible simulation environment tailored for in-network computing research. Key implementation details are as follows:

- **Memory Management:** The simulator uses smart pointers for automatic and efficient memory management. This approach minimizes memory leaks and allows safe, efficient handling of dynamically allocated resources, which is especially critical for simulating network topologies and operations involving numerous interconnected objects.
- **Concurrency Control:** Given the parallel nature of the simulator, concurrency is handled using standard C++ library utilities. These constructs help manage shared resources, ensuring thread safety when multiple computing nodes and switches interact concurrently.
- **Logging and Debugging:** The simulator includes a flexible logging system with various logging levels for traceability, debugging, and performance monitoring. This structure allows for fine-grained control over log output, aiding in the identification of issues and ensuring efficient performance tuning during development and experimentation.
- **Command Line Customization:** Users can modify the simulator's parameters through command-line arguments, enabling easy customization of simulation settings such as network size, number of ports, and operation types. This flexibility allows for rapid configuration changes, facilitating experiments across different network topologies and collective operations without code modifications.

- **Cross-Platform Compatibility:** The simulator is designed to be compatible with both Linux and Windows environments, making it accessible to a wide range of users and adaptable to various system configurations.
- **Standalone Parallel Processing:** The simulator is entirely self-contained, with no external dependencies required for parallel processing simulation. This independence ensures portability and ease of deployment, allowing the simulator to run in environments where dependency management might be challenging.
- **MPI-Like API:** To facilitate migration and usability, the simulator provides an API that mimics the widely used MPI interface, making it intuitive for users familiar with MPI-based parallel programming. This API simplifies the adaptation of existing MPI-based code to the simulator, enabling researchers to leverage the INC functionalities within familiar MPI-like operations.

This technical foundation makes the simulator a powerful tool for exploring the potential of in-network computing in parallel computing environments, offering flexibility, efficiency, and a high degree of customization.

5. EXPERIMENTAL RESULTS

This section presents an in-depth analysis of the simulated performance of various collective communication operations, comparing traditional and in-network computing (INC) methods across different port configurations. Each operation—Barrier, Broadcast, Scatter, Gather, and Reduce—is evaluated based on critical performance metrics, including timing cost, bandwidth usage, and synchronization quality. By examining these metrics under both INC-enabled and traditional setups, this section highlights the effectiveness of INC in enhancing efficiency, reducing network congestion, and improving synchronization consistency. The insights provided here will support a deeper understanding of how INC can optimize communication patterns in parallel computing environments.

Table 5.1 provides an overview of the relationship between the number of computing nodes and switches across different port configurations. This information serves as a reference point to help interpret the results in the following sections. As the port count increases, the number of computing nodes and network switches scales accordingly, impacting both timing and bandwidth metrics. By reviewing this table alongside the graphs, readers can better understand the network structure and node-switch distribution, which are integral to analyzing the effects of in-network computing (INC) and traditional methods on each collective communication operation.

Ports	Computing Nodes	Edge Switches	Agg. Switches	Core Switches
4	16	8	8	4
6	54	18	18	9
8	128	32	32	16
10	250	50	50	25
12	432	72	72	36
14	686	98	98	49
16	1024	128	128	64
18	1458	162	162	81
20	2000	200	200	100
22	2662	242	242	121
24	3456	288	288	144
26	4394	338	338	169
28	5488	392	392	196
30	6750	450	450	225
32	8192	512	512	256
34	9826	578	578	289
36	11664	648	648	324
38	13718	722	722	361
40	16000	800	800	400
42	18522	882	882	441
44	21296	968	968	484
46	24334	1058	1058	529
48	27648	1152	1152	576

Table 5.1 Amount of objects compared under different port configurations

The performance results presented in this section were obtained using a simulation environment running on a machine with the following specifications: an **Intel i7-12700H** processor with a maximum clock frequency of **4700 MHz** and a caching hierarchy consisting

of **544 KiB of L1d** cache, **704 KiB of L1i** cache, **11.5 MiB of L2** cache, and **24 MiB of L3** cache. The system is equipped with **64 GB of RAM**, providing ample memory to support large-scale simulations without bottlenecks. This hardware configuration ensures that the reported results are free from resource constraints and accurately reflect the performance differences between traditional and INC-enabled methods. However, the impact of hardware-specific factors, such as processor speed and memory access times, should be considered when interpreting these results.

5.1. Timing Costs

The Timing Costs section presents a comparison of the time taken to complete various collective communication operations in parallel computing. This metric, often referred to as synchronization or execution time, measures the duration from the start to the completion of each operation. By assessing the timing costs of both traditional and INC-based methods, this analysis highlights the impact of in-network computing (INC) on reducing latency across different operations. Each following sub-section details the timing performance for a specific communication operation—ranging from simple broadcasts to complex matrix multiplications—to demonstrate how INC can optimize synchronization in distributed systems.

5.1.1. Barrier Synchronization

As shown in Figure 5.1, when INC is disabled, the synchronization cost escalates considerably as the number of ports increases. In contrast, enabling INC to maintain a lower and more consistent synchronization cost across varying port counts, underscoring the efficiency of INC in minimizing the overhead of barrier synchronization.

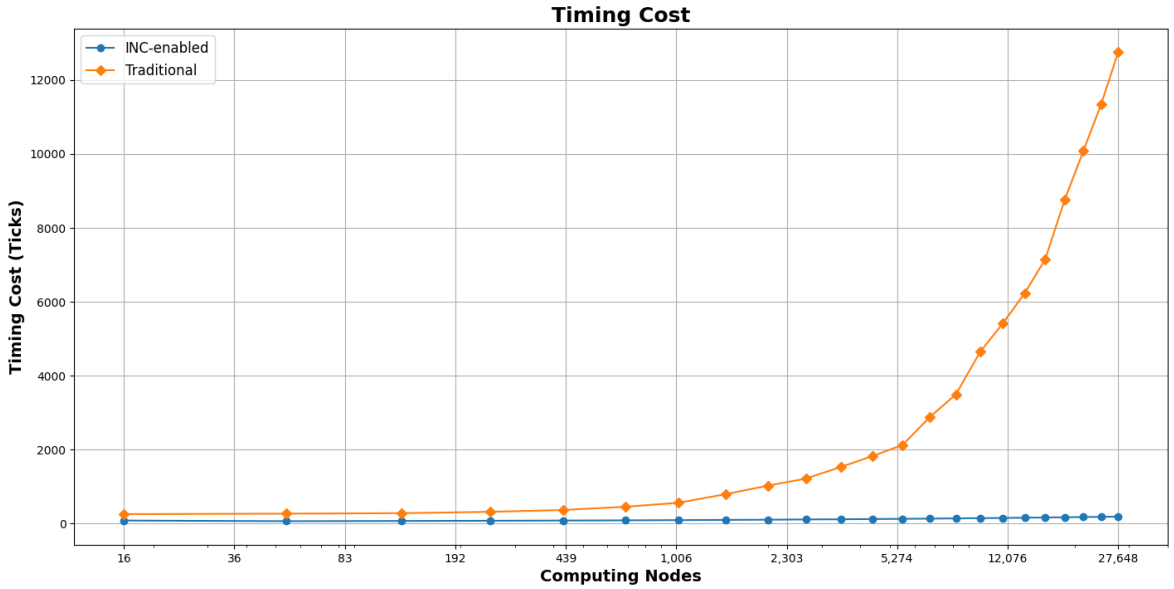

Figure 5.1 Barrier Synchronization Timing Cost

5.1.2. Broadcast

For Broadcast operations, in terms of timing cost, the traditional method exhibits a notably higher cost as it requires the source node to send data directly to each target node individually. By contrast, the INC-enabled approach achieves lower timing costs by leveraging network switches to disseminate data efficiently.

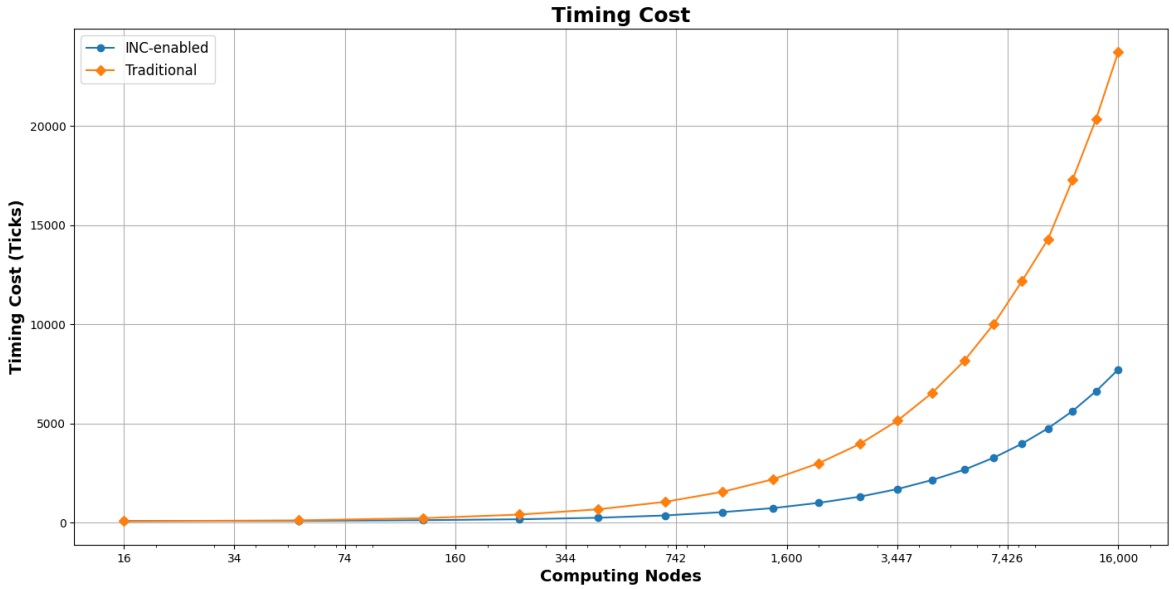

Figure 5.2 Broadcast Timing Cost

Figure 5.2 highlights the reduction in timing cost achieved with INC, which streamlines data transmission across the network by reducing the reliance on the source node alone for data delivery.

5.1.3. Scatter

The scatter operation exhibits a significantly lower timing cost when using INC compared to the traditional method. As shown in Figure 5.3, INC consistently achieves timing costs that are approximately half those of the traditional approach across various port configurations. This dramatic reduction highlights the efficiency of in-network computing in optimizing data dispersion, as switches actively participate in directing data to the appropriate nodes, minimizing delays. Moreover, INC demonstrates remarkable stability in timing cost, with only minor increases as the number of computing nodes grows. In contrast, the traditional method shows a steep rise in timing cost with increasing port counts and node numbers, exacerbated by the higher communication overhead and less efficient data distribution. The divergence between the two methods becomes more pronounced at larger scales,

emphasizing the scalability and effectiveness of INC in reducing operational delays in scatter operations

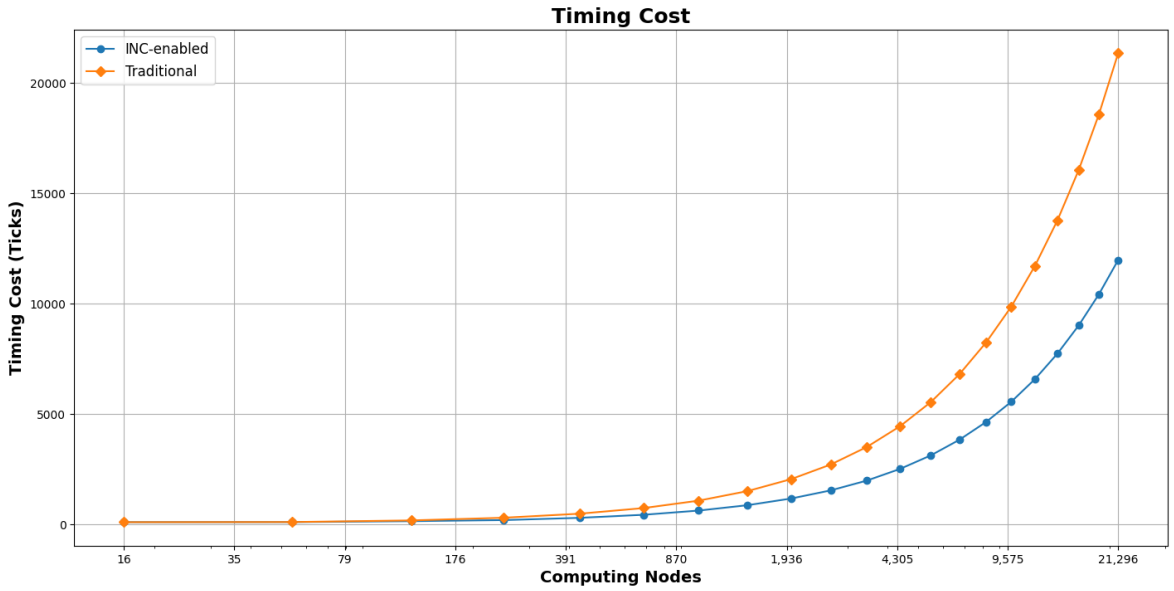

Figure 5.3 Scatter Timing Cost

5.1.4. Gather

As seen in Figure 5.4, with INC enabled, the timing cost grows moderately as the number of ports increases, thanks to in-network data aggregation. In contrast, the traditional method sees a more substantial increase in timing cost, as each computing node sends its data individually to the target, leading to increased latency and operational delays. This improvement highlights the efficiency of INC in managing large-scale data aggregation.

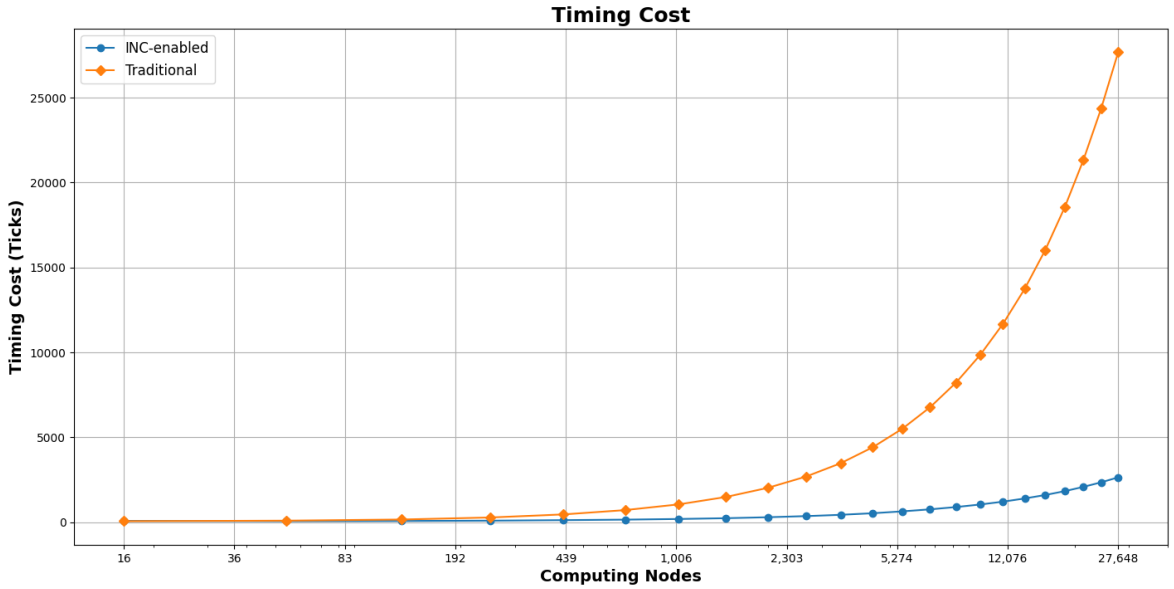

Figure 5.4 Gather Timing Cost

5.1.5. All-gather

Illustrated in Figure 5.5, the timing cost of the All-Gather operation shows a steep increase with larger port counts and node numbers in traditional systems. In contrast, the INC-enabled approach demonstrates a more controlled growth in timing costs, showcasing its ability to efficiently manage the increased communication demands. By utilizing the computational capabilities of network switches, the INC approach ensures that data aggregation and distribution occur with minimal delays, even as the system scales. This efficiency highlights INC's effectiveness in maintaining lower operation latencies compared to traditional methods.

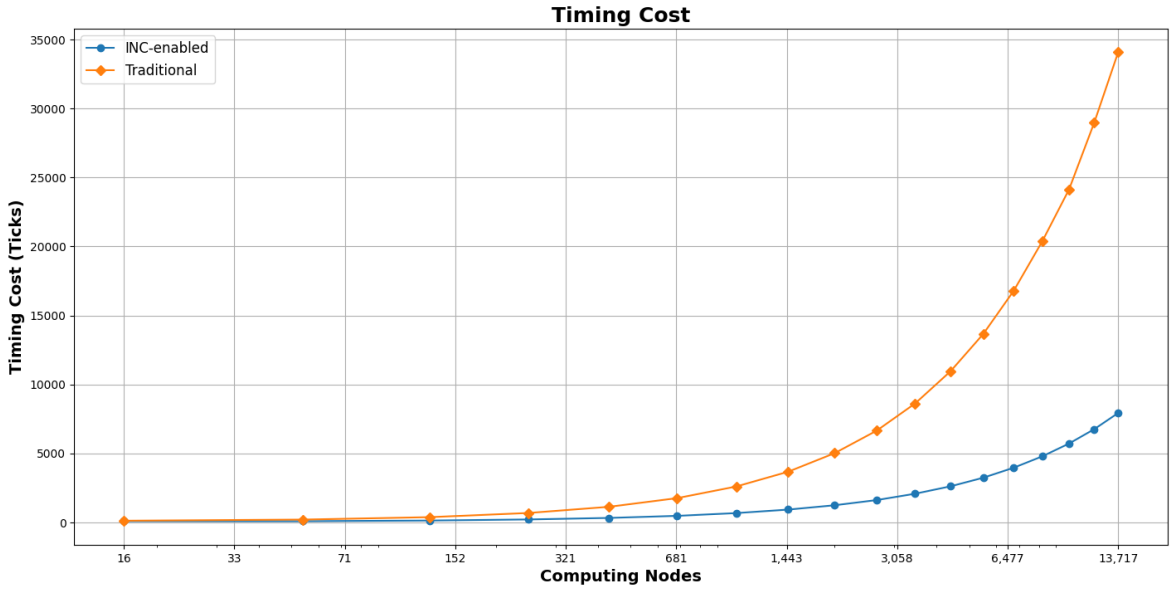

Figure 5.5 All-Gather Timing Cost

5.1.6. Reduce

When using INC, timing cost is consistently lower across various port configurations, with only minor increases as the number of computing nodes grows. This stability highlights INC's ability to effectively manage and carry out data reduction by leveraging network resources to optimize the process. In contrast, the traditional approach shows a marked increase in timing cost with the number of ports, especially as node count rises. The difference between INC and traditional methods becomes increasingly pronounced with larger port counts, demonstrating INC's efficiency in reducing operational delays. The corresponding results are displayed in Figure 5.6.

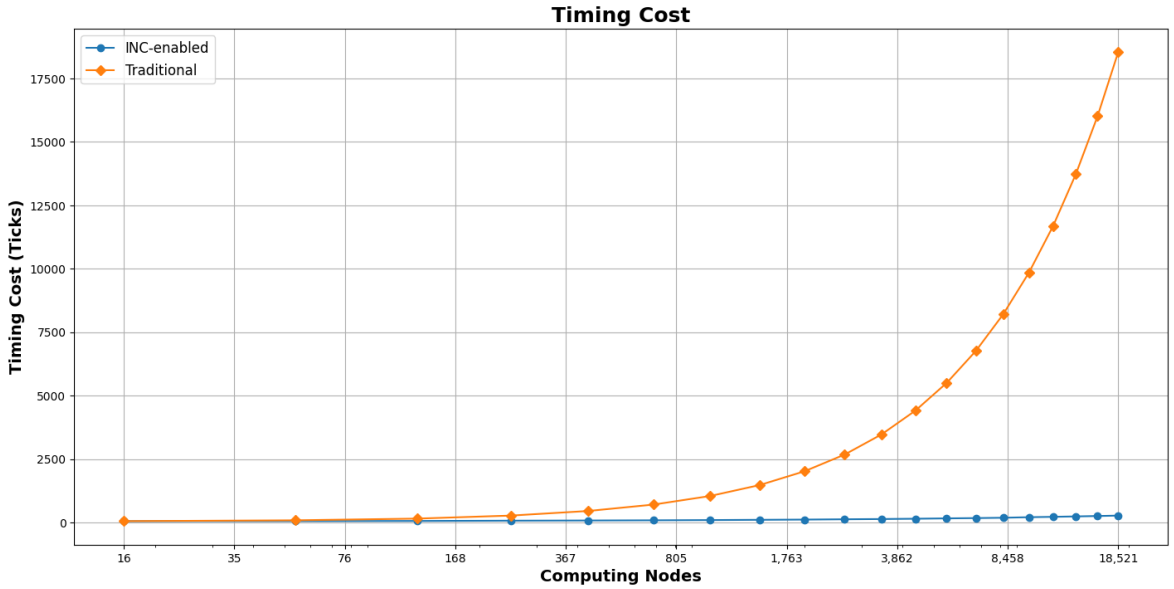

Figure 5.6 Reduce Timing Cost

5.1.7. All-reduce

Delineated in Figure 5.7, with INC-enabled networking, timing costs for the all-reduce operation scale minimally as the number of ports and computing nodes increase, whereas the traditional method exhibits a significantly steeper rise in timing costs under similar conditions. This highlights the effectiveness of in-network processing in handling the increased communication demands of all-reduce operations, particularly by aggregating partial results at intermediate switches. As the system grows in scale, INC's ability to reduce processing delays becomes increasingly evident, providing substantial performance benefits over traditional approaches.

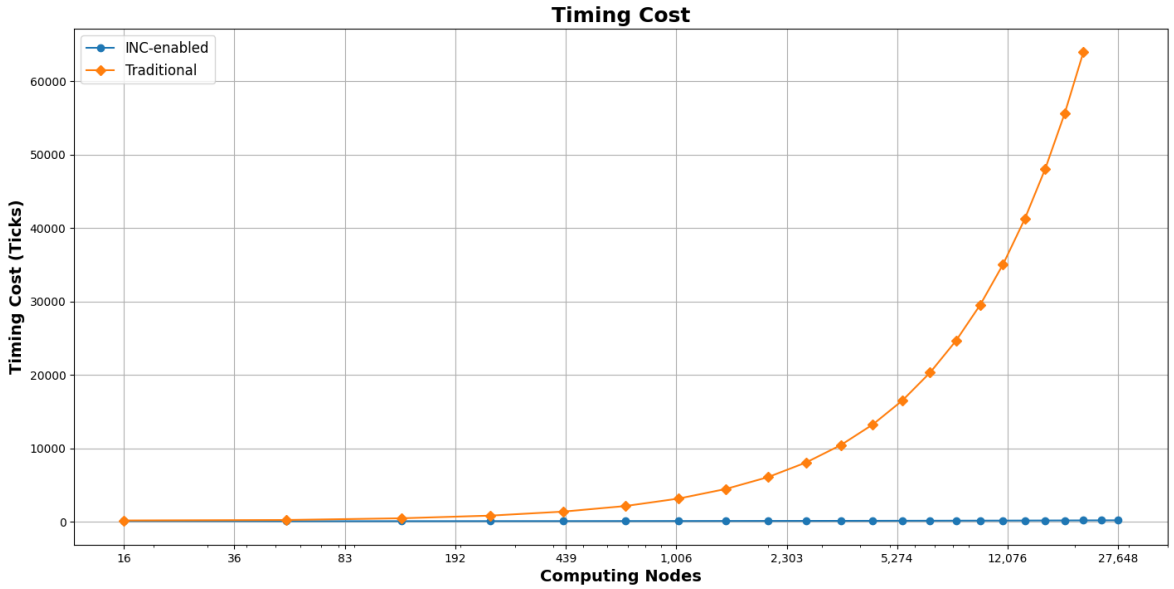

Figure 5.7 All-Reduce Timing Cost

5.2. Bandwidth Usage

The Bandwidth Usage section evaluates the amount of network bandwidth consumed by each communication operation. This metric is essential for understanding the efficiency of data movement within the network. High bandwidth usage can lead to network congestion, reducing overall system performance. By comparing bandwidth usage between traditional and INC-based approaches, this section illustrates how INC can reduce data transmission overhead. Each sub-subsection presents the results for individual operations, allowing for a direct comparison of network efficiency improvements provided by INC across various collective operations.

5.2.1. Barrier Synchronization

In Figure 5.8, we observe a steeper increase in network message count as the port count rises without INC. With the INC-enabled approach, as expected, fewer messages are transmitted in the network.

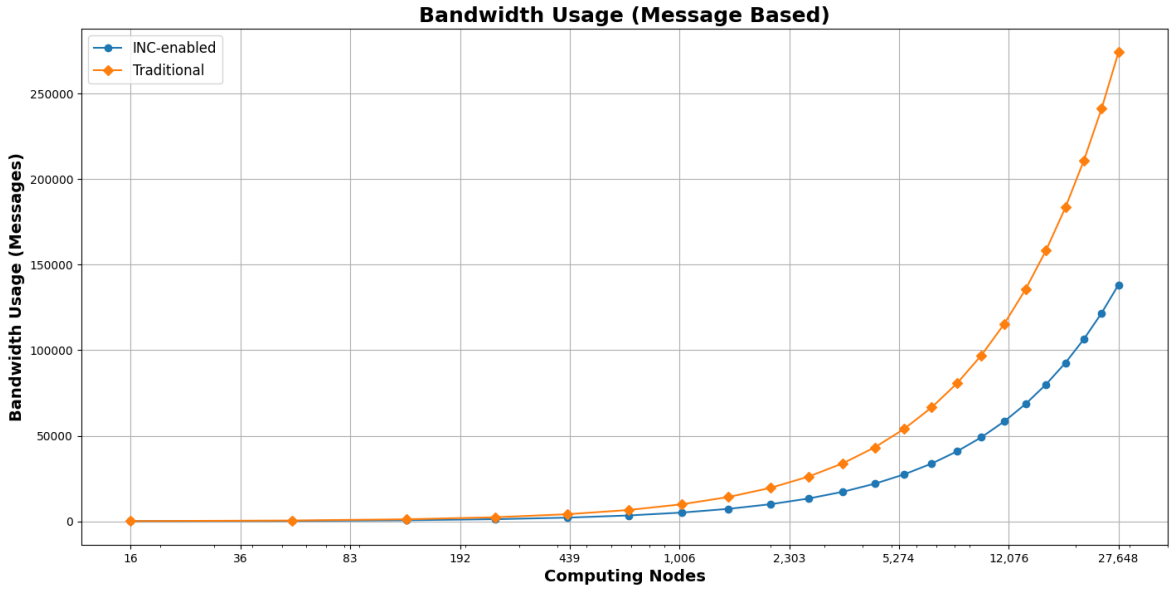

Figure 5.8 Barrier Synchronization Bandwidth Usage (Message)

Figure 5.9, along with Figure 5.8, proves the efficiency of the INC-enabled approach by showing that this approach not only reduces the total transmitted message amount, but it also reduces the message sizes.

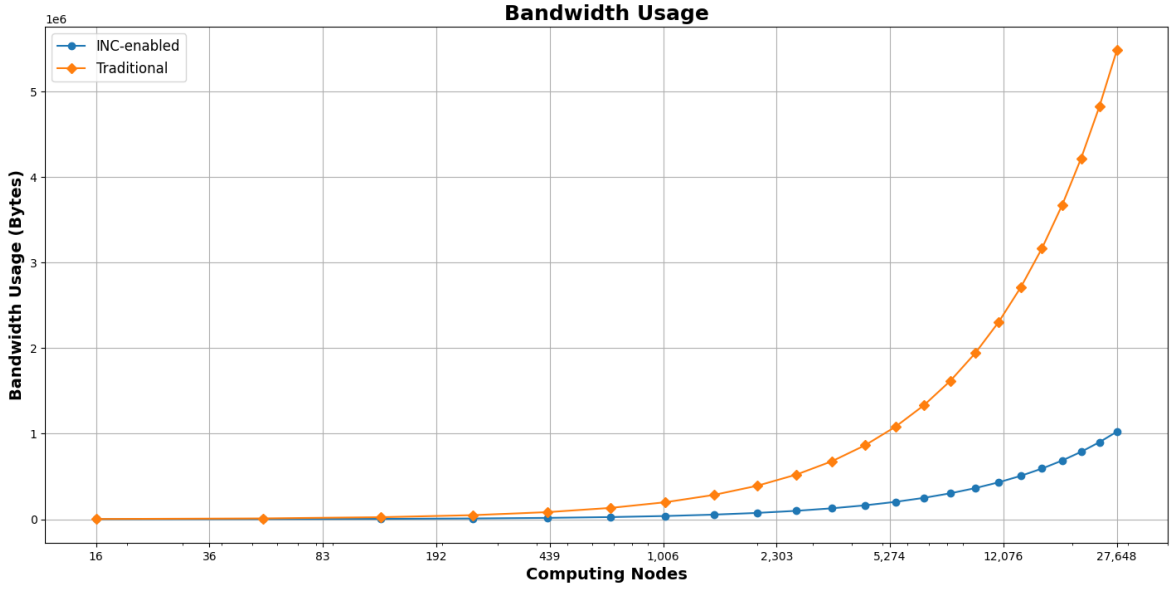

Figure 5.9 Barrier Synchronization Bandwidth Usage (Byte)

5.2.2. Broadcast

In a Broadcast operation, the bandwidth usage is significantly higher than that of the traditional method due to the high volume of individual messages sent from the source to each target node. INC, on the other hand, reduces bandwidth demands by directing data more efficiently via network switches.

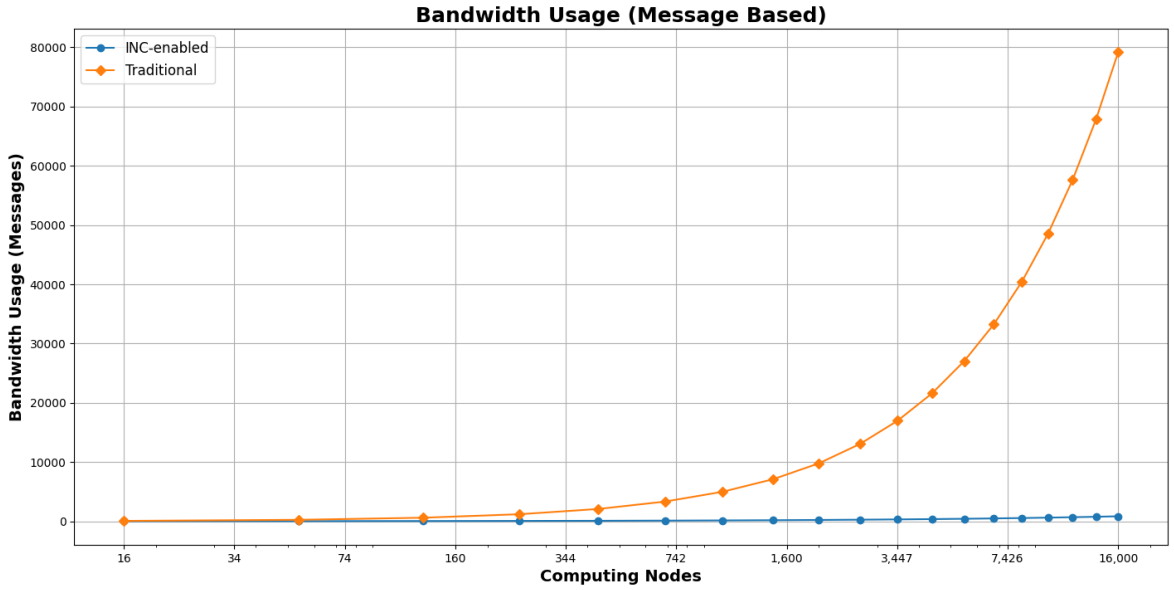

Figure 5.10 Broadcast Bandwidth Usage (Message)

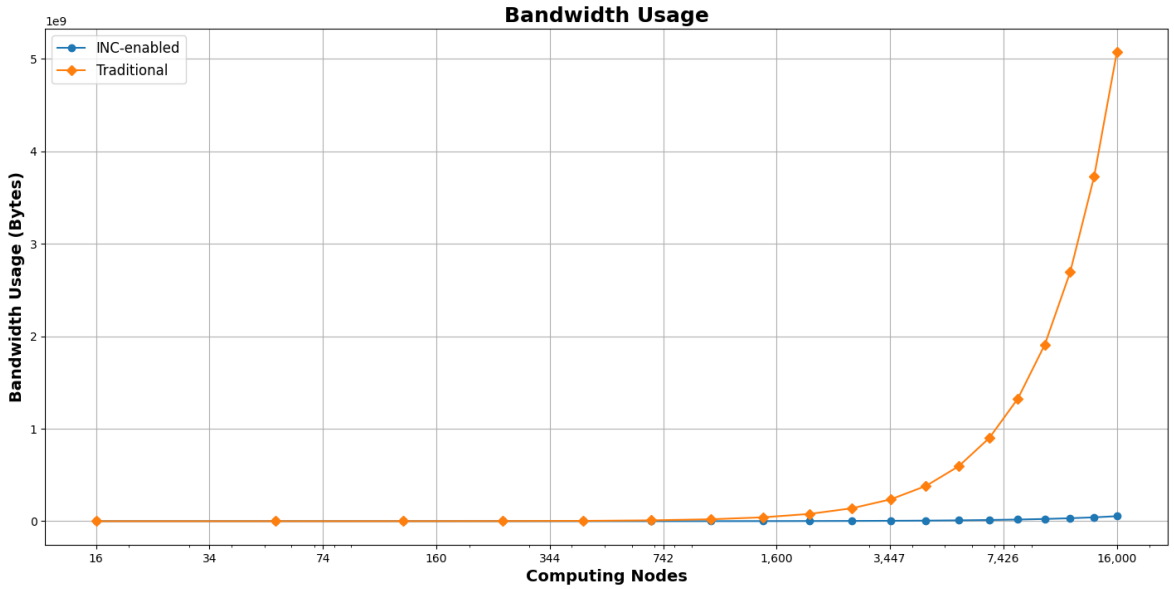

Figure 5.11 Broadcast Bandwidth Usage (Byte)

As shown in Figure 5.10 and Figure 5.11, an INC-enabled Broadcast operation requires fewer messages and message sizes, conserving network bandwidth and reducing the likelihood of congestion.

5.2.3. Scatter

A Scatter operation is quite similar to a Broadcast operation as described in section 4.. The only difference is that when using the INC-enabled approach the transmitted data gets smaller at each layer of the network. Therefore, the results of the Scatter operation are also similar to the Broadcast operation. With the INC-enabled approach, bandwidth usage is significantly lower than in the traditional setup, suggesting that INC manages message flow more efficiently. This efficiency minimizes congestion, as INC enables data to be distributed more selectively, reducing the overall volume of transmitted messages. As seen in the Figure 5.12 and Figure 5.13, bandwidth usage scales drastically with port count, showing a sharp increase in message volume and byte volume, emphasizing the heavier network burden associated with standard communication methods.

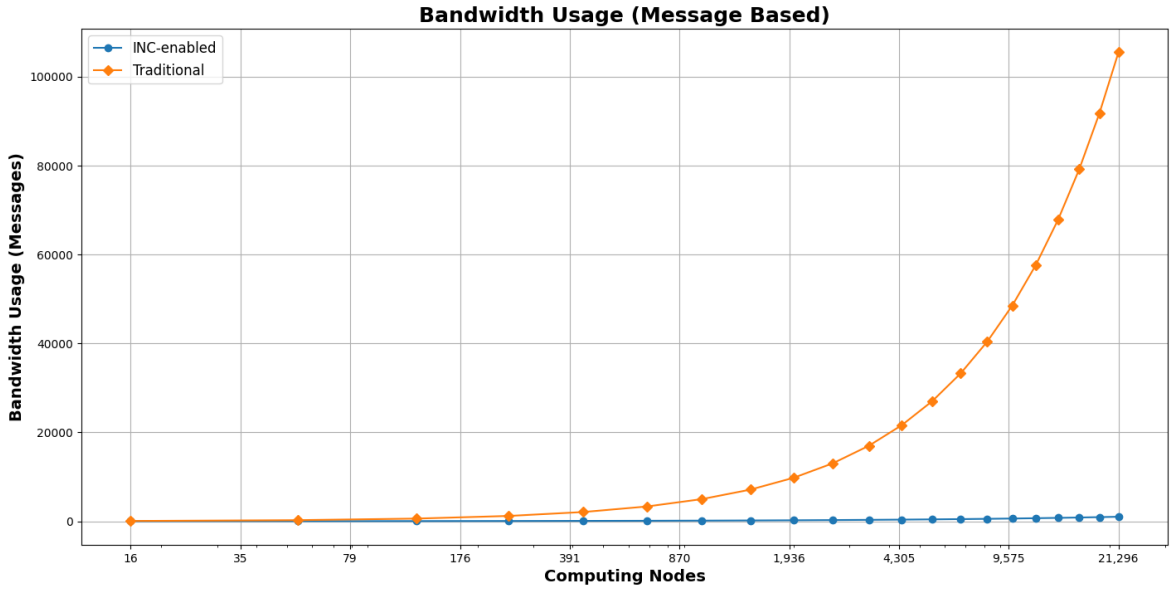

Figure 5.12 Scatter Bandwidth Usage (Message)

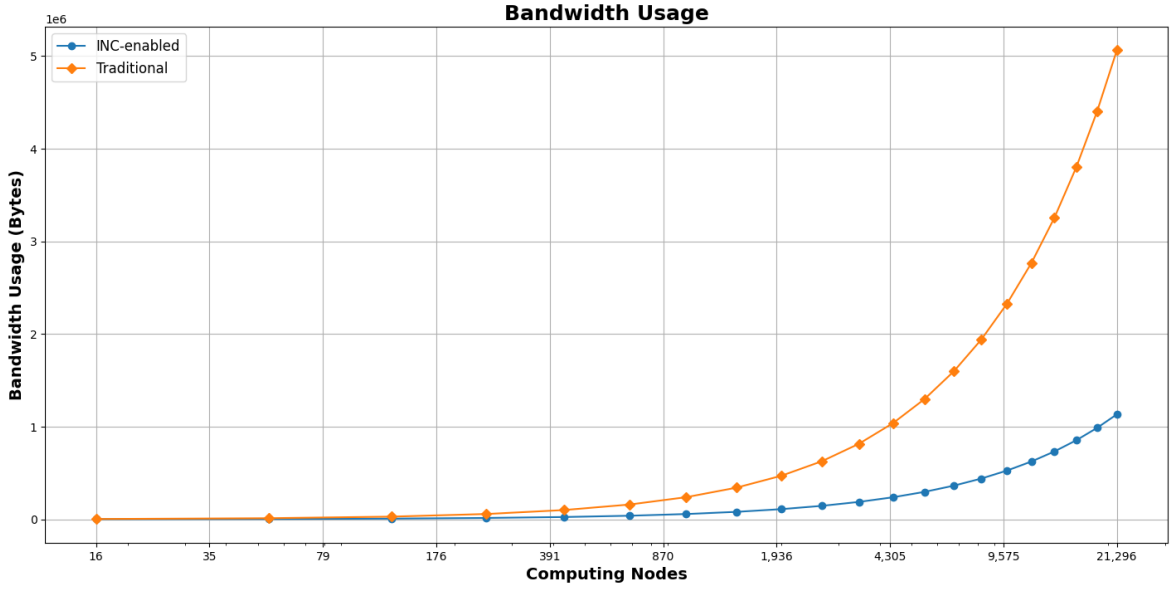

Figure 5.13 Scatter Bandwidth Usage (Byte)

5.2.4. Gather

The Gather operation implemented in the INC-enabled approach prioritizes aggregating the data at each layer of the network. As data from different contributing nodes is gathered at

each layer and sent as a single message, the bandwidth usage in terms of both message and byte amounts are lower compared to the traditional approach.

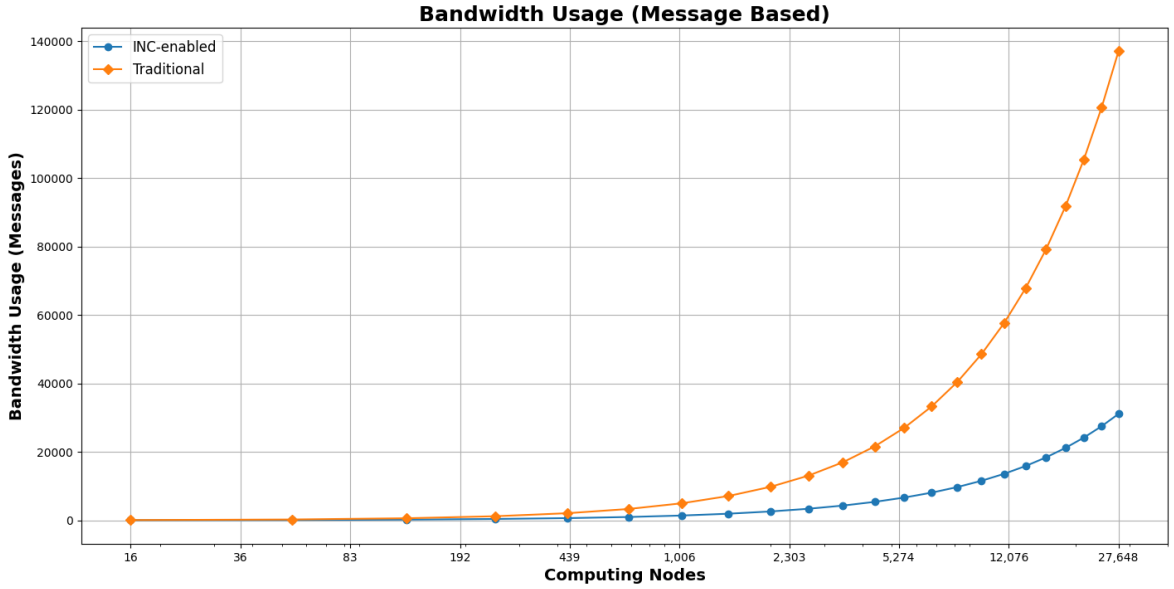

Figure 5.14 Gather Bandwidth Usage (Message)

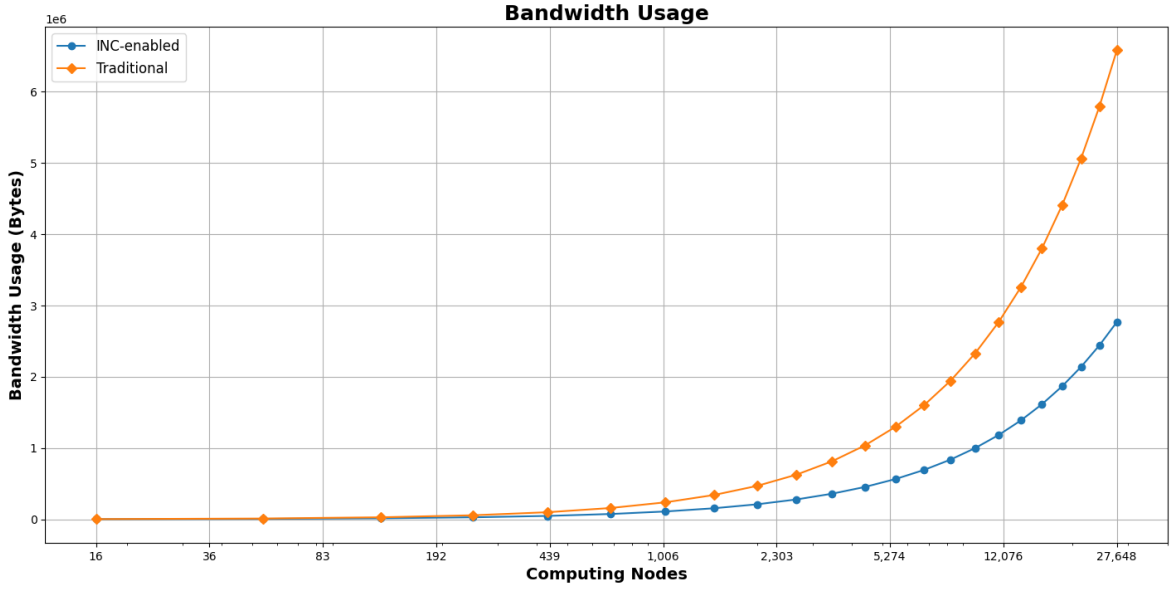

Figure 5.15 Gather Bandwidth Usage (Byte)

As highlighted in Figure 5.14 and Figure 5.15, in the traditional method, each node transmits data directly to the target, which significantly increases message volume as node numbers

grow. This demonstrates INC's effectiveness in reducing message overhead, which is crucial in scenarios with higher port counts.

5.2.5. All-gather

Depicted in Figure 5.16, the bandwidth usage in terms of message count for the All-Gather operation is significantly reduced in the INC-enabled setup compared to the traditional approach. Traditional methods involve a greater number of message exchanges during data collection and distribution as each node communicates with the other, leading to higher network congestion. The INC-enabled system, however, leverages switch-level aggregation and distribution to streamline data flow, reducing the total volume of transmitted messages.

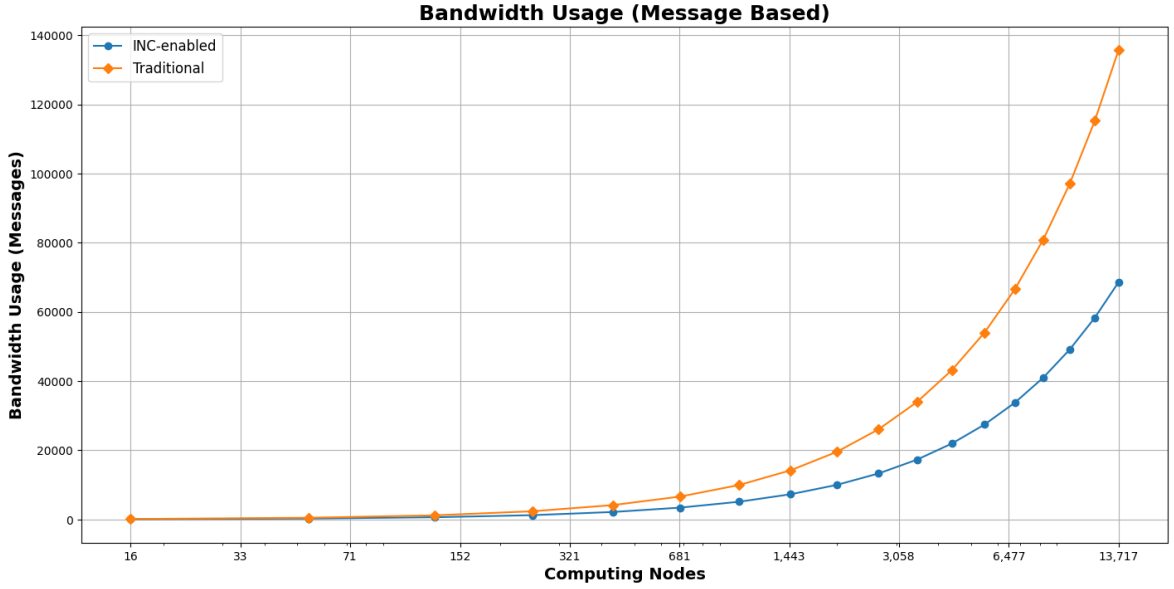

Figure 5.16 All-Gather Bandwidth Usage (Message)

Although the network traffic based on message amount is less in the INC-enabled approach, it has slightly higher bandwidth usage in terms of total transmitted bytes. This is due to the redundancy in messages that occurs by design. To better synchronize the operation and ease the implementation, we preferred to collect the gathered data at each core switch and this caused the total bandwidth usage to be slightly higher.

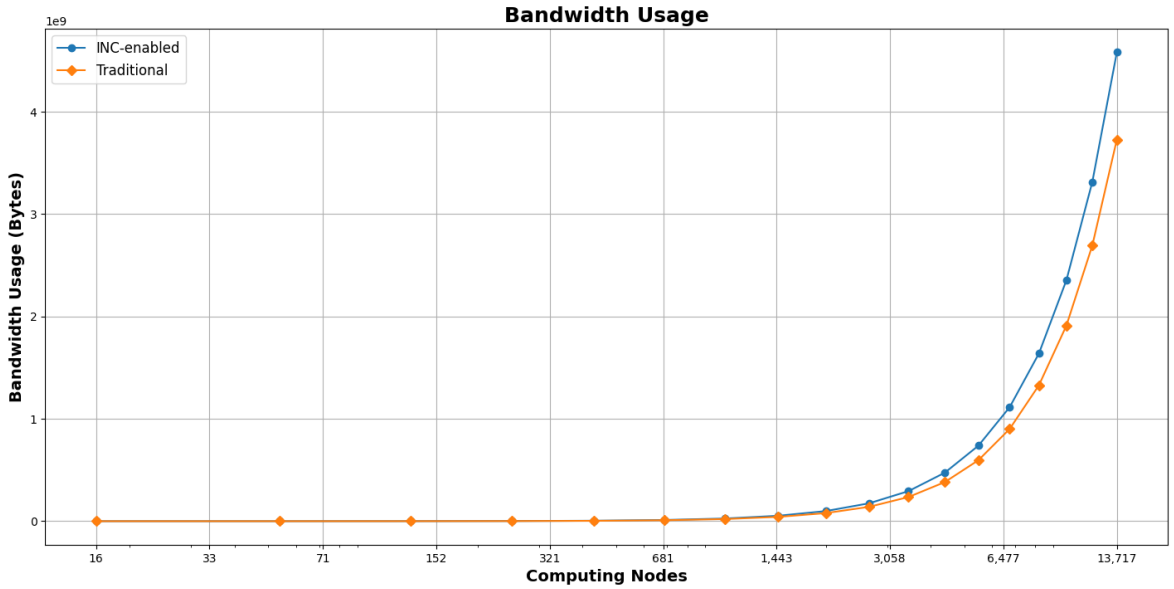

Figure 5.17 All-Gather Bandwidth Usage (Byte)

The possible implementations of this operation comes with a design dilemma in which the designer should either sacrifice bandwidth usage or timing cost. An alternative to this approach could be similar to the single Gather operation where we preferred the column-based approach and didn't utilize the processing capability of the core switches.

5.2.6. Reduce

In a Reduce operation, the INC-enabled approach shows substantial savings in bandwidth usage, both in terms of total messages and bytes, compared to the traditional method. Demonstrated in Figure 5.18 and Figure 5.19, with fewer messages and bytes needed to complete the operation, the network is less congested in the INC-enabled setup, leading to improved performance and reduced overhead in data communication.

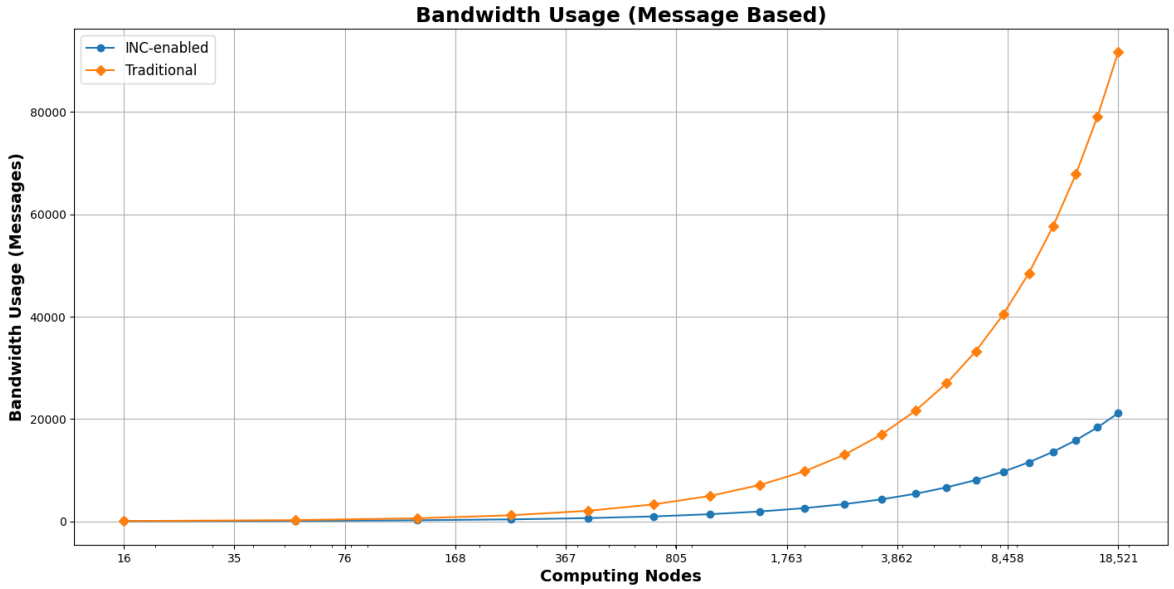

Figure 5.18 Reduce Bandwidth Usage (Message)

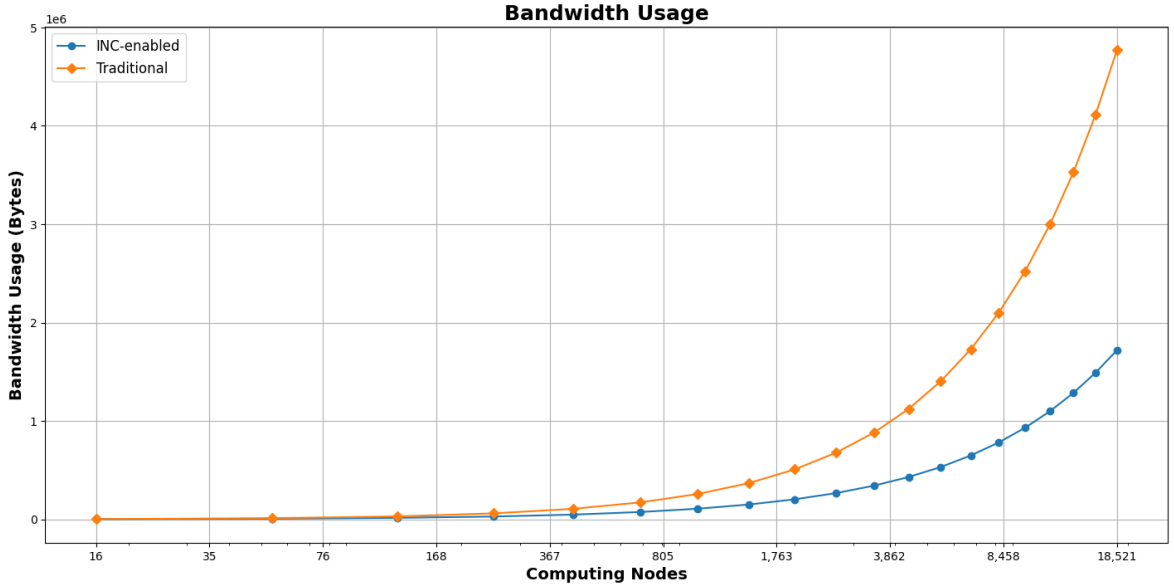

Figure 5.19 Reduce Bandwidth Usage (Byte)

5.2.7. All-reduce

The INC-enabled approach for all-reduce operations also demonstrates significant savings in bandwidth usage compared to the traditional method. As shown in Figure 5.20 and

Figure 5.21, the bandwidth usage required to complete the operation is notably lower in the INC-enabled setup. This reduction alleviates network congestion and improves communication efficiency, especially as the number of computing nodes and ports increases. By leveraging in-network aggregation, the INC-based approach minimizes data transmission overhead, further enhancing the scalability and performance of the All-Reduce operation.

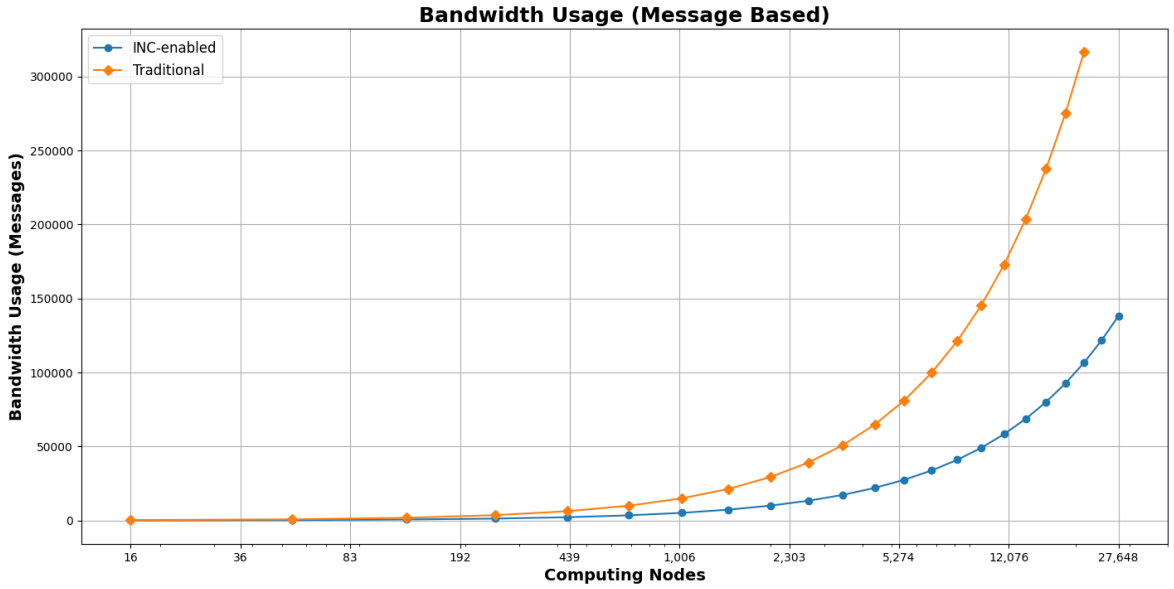

Figure 5.20 All-Reduce Bandwidth Usage (Message)

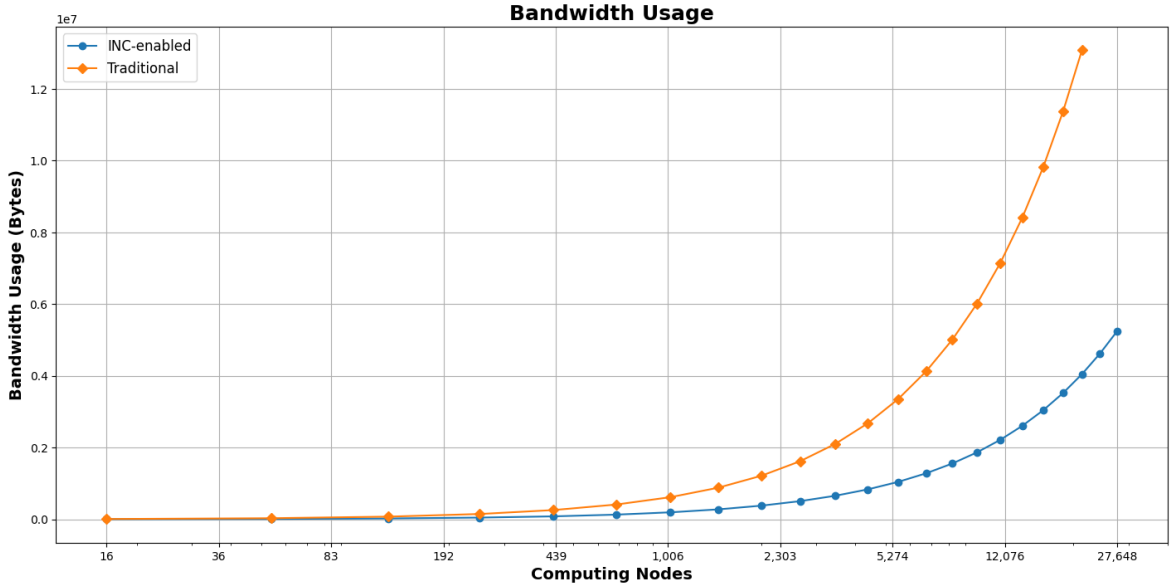

Figure 5.21 All-Reduce Bandwidth Usage (Byte)

5.3. Completion Time Difference

The Completion Time Difference section examines the variance in completion times among computing nodes for each operation, which serves as a measure of synchronization consistency. This metric, often referred to as time variation or skew, indicates how evenly synchronized nodes are during the execution of collective operations. Lower completion time differences imply higher synchronization quality, which is critical for parallel computing performance. This section compares the consistency achieved with traditional methods versus INC, highlighting how INC affects synchronization quality across various operations. Each sub-subsection addresses a specific collective communication operation, offering insight into how INC reduces time disparities and enhances uniformity in task completion across nodes.

5.3.1. Barrier Synchronization

Synchronization quality is assessed by measuring the maximum variation in tick counts across computing nodes at the barrier release moment, revealing the consistency of synchronization.

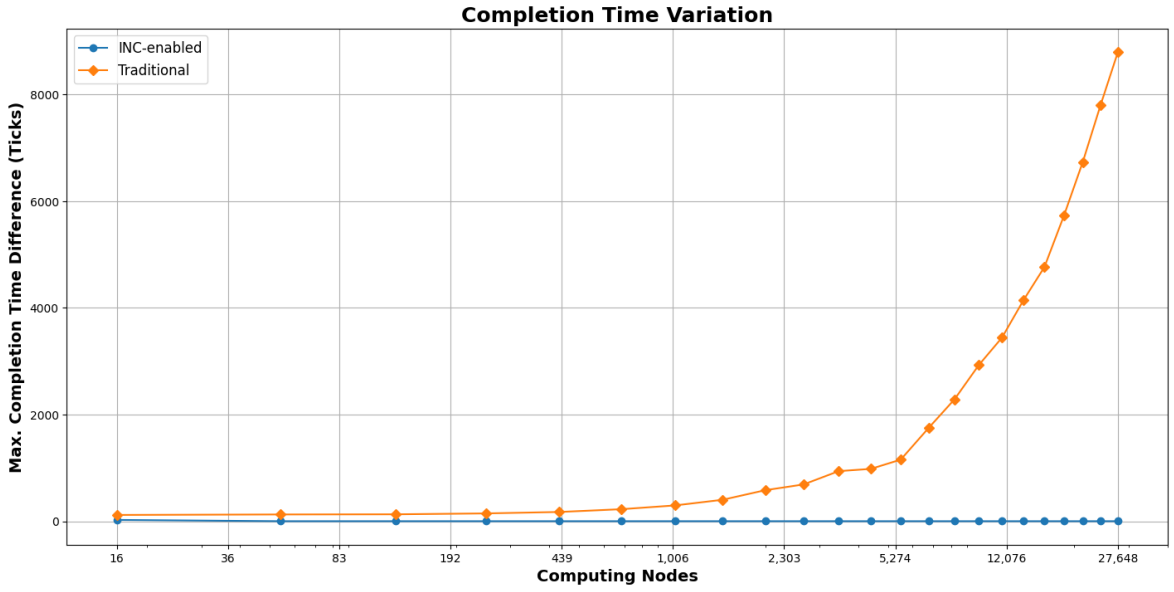

Figure 5.22 Barrier Synchronization Quality

As depicted in Figure 5.22, INC maintains a low and stable maximum time variation, ensuring high synchronization consistency. Without INC, however, the time variation increases substantially with higher port counts, indicating less uniform synchronization across nodes.

5.3.2. Broadcast

The completion time difference, or the disparity in broadcast completion times across nodes, is notably higher in the traditional method. With INC, the network actively participates in directing data to target nodes, ensuring more synchronized delivery times.

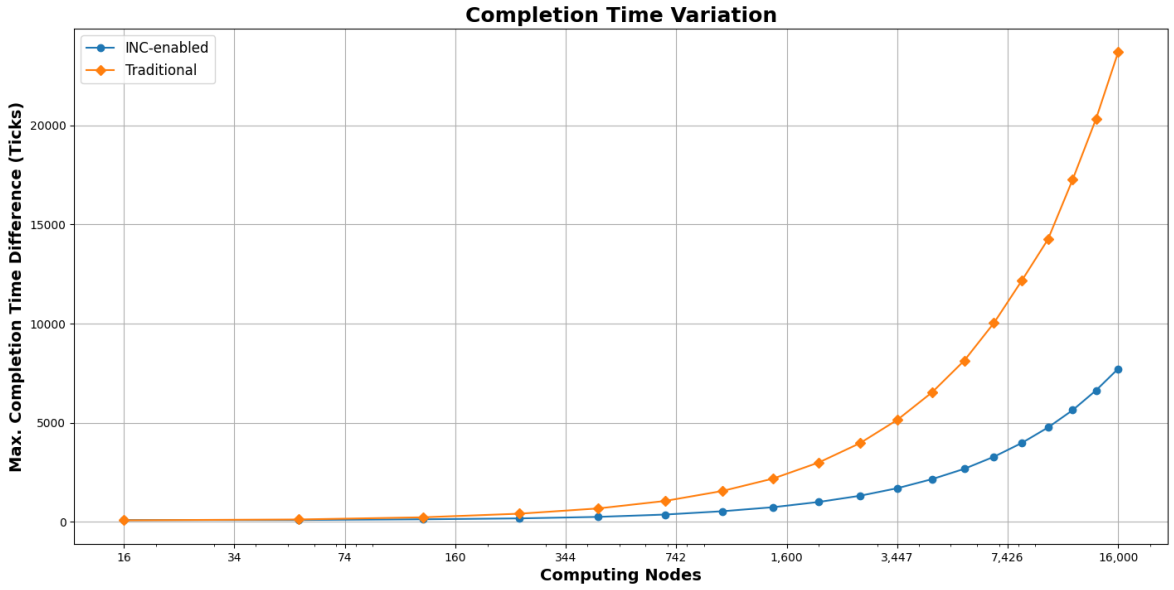

Figure 5.23 Broadcast Completion Time Difference

In Figure 5.23, we observe that INC minimizes completion time differences across nodes, resulting in a more consistent data reception time across the network.

5.3.3. Scatter

The completion time difference represents the variance in operation completion times across computing nodes. Using INC minimizes this variance, maintaining more synchronized

completion times regardless of the number of ports involved. However, Traditional communication exhibits a substantial increase in completion time difference as port numbers grow, indicating less consistent data arrival times. This inconsistency can impact overall system performance, particularly in applications requiring strict timing coordination across nodes. Figure 5.24 presents the situation.

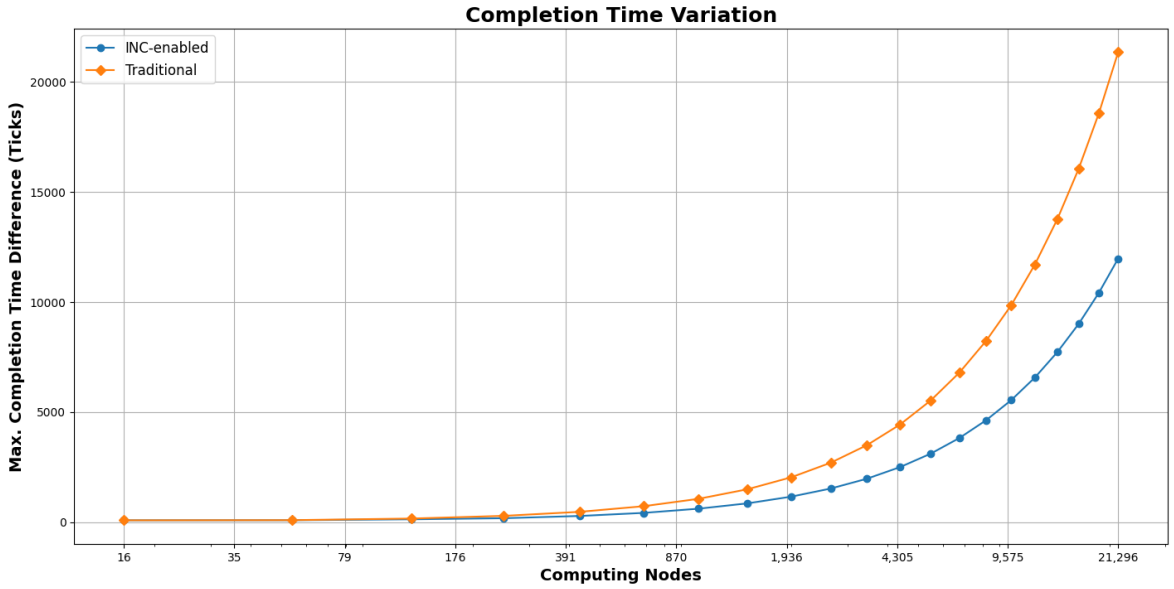

Figure 5.24 Scatter Completion Time Difference

5.3.4. Gather

As demonstrated in Figure 5.25, under INC, completion time differences stay relatively low, demonstrating more uniform synchronization across nodes. In the traditional method, these differences widen with port count, indicating less consistent data aggregation and greater synchronization delays. The consistency provided by INC highlights its benefit in ensuring prompt, synchronized aggregation across nodes

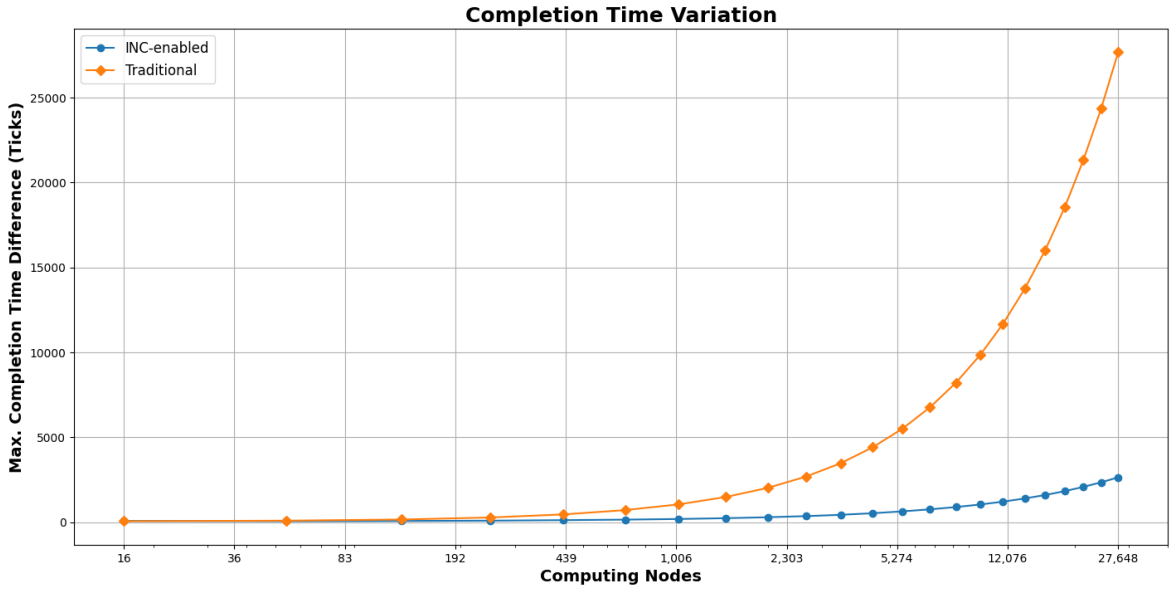

Figure 5.25 Gather Completion Time Difference

5.3.5. All-gather

As shown in Figure 5.26, the INC-enabled approach achieves a more balanced and uniform completion time across nodes for the All-Gather operation. Traditional methods exhibit greater variability, particularly as the network grows in size, indicating a lack of synchronization among nodes. By distributing the workload and utilizing the hierarchical structure of the network, INC ensures that all nodes receive the aggregated data within a narrow time window. This synchronization consistency underscores the robustness of INC in handling large-scale data sharing operations efficiently.

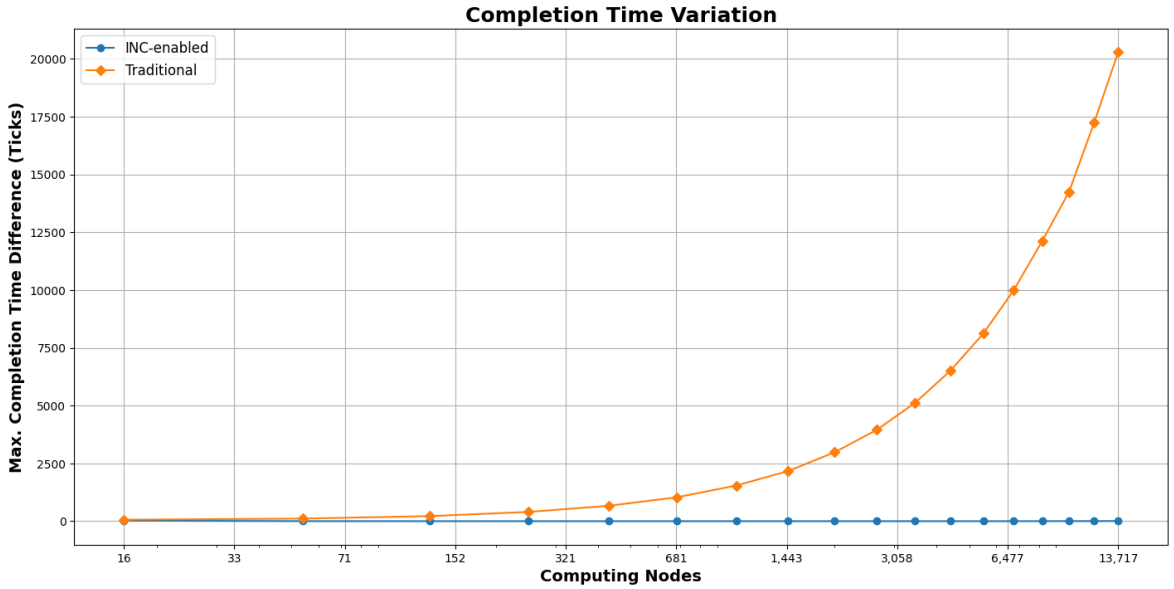

Figure 5.26 All-Gather Completion Time Difference

5.3.6. Reduce

As shown in Figure 5.27, in-network computing shows a more uniform distribution of completion times, as indicated by smaller time differences. Conversely, the traditional approach shows a higher variation, particularly as the network scales, suggesting that INC enables more consistent and synchronized completion across nodes.

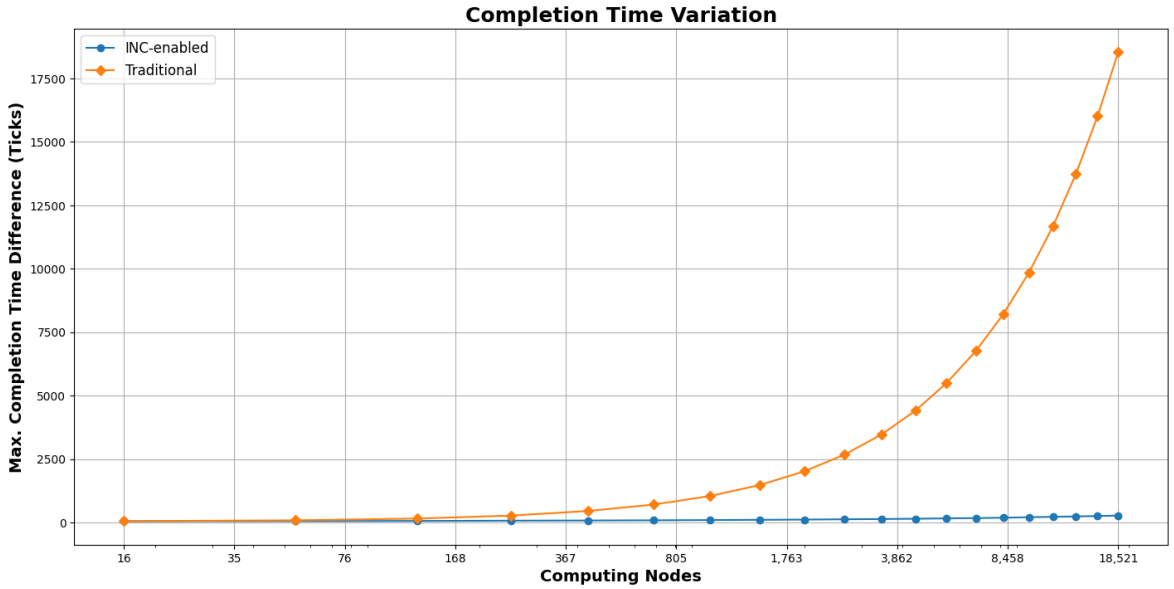

Figure 5.27 Reduce Completion Time Difference

5.3.7. All-reduce

As shown in Figure 5.28, in-network computing enables a more uniform distribution of completion times for the all-reduce operation, as evidenced by smaller time differences between nodes. In contrast, the traditional approach exhibits a higher variation in completion times, particularly as the number of ports and computing nodes increases. This consistency provided by INC highlights its ability to maintain synchronized operation across nodes, reducing disparities and ensuring reliable performance in large-scale systems.

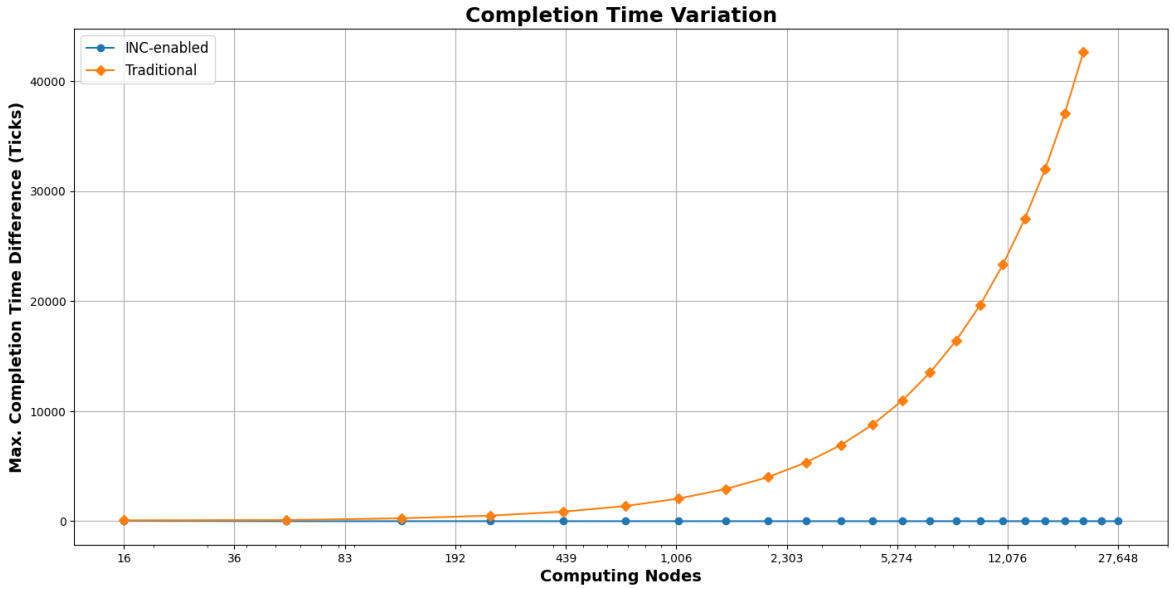

Figure 5.28 All-Reduce Completion Time Difference

5.4. Direct Message

Enabling in-network computing (INC) significantly enhances the performance of collective communication operations in parallel computing systems by allowing intermediate switches to actively participate in data processing. This minimizes the data volume and reduces synchronization costs, which improves both timing and bandwidth efficiency for operations like broadcast, scatter, and reduce. However, INC has minimal to no effect on the performance of direct, point-to-point messages. In direct communication, switches function as passive relays, simply forwarding data between specific nodes without performing any data manipulation. Since no computation is required from the switches for these messages, INC's benefits do not extend to this type of communication, leaving the direct message performance largely unchanged. Thus, while INC optimizes collective operations, its influence on direct message exchanges is neutral, maintaining consistent message latency and throughput for unprocessed traffic.

5.5. Matrix Multiplication

Matrix multiplication is a fundamental operation in many parallel computing applications, including scientific simulations, machine learning, and graphics processing. This section evaluates the performance of matrix multiplication under both traditional and In-Network Computing (INC) enabled configurations. The analysis focuses on three key metrics: timing cost, bandwidth usage, and completion time difference, as illustrated in the accompanying graphs.

5.5.1. Algorithm

The matrix multiplication algorithm follows a distributed approach, executed in several steps:

1. **Matrix Initialization:**
    - If the node's ID (denoted as `m_ID`) is 0 (i.e., the root node), it initializes two matrices, `matrixA` and `matrixB`, each of size `computingNodeAmount` × `computingNodeAmount`.
    - The root node populates these matrices with values based on row and column indices:
        - `matrixA[i][j] = i + j`
        - `matrixB[i][j] = i - j`
2. **Matrix Distribution:**
    - The root node scatters rows of `matrixA` across all computing nodes, ensuring each node receives its designated row.
    - The entire `matrixB` is broadcasted to all nodes, allowing every node to access it for multiplication.
3. **Local Matrix Multiplication:**

- Each node initializes a `localRow` vector to store its portion of the result matrix.
- For each column index `colIdxB` in `matrixB`, the node:
  - Sets a running `sum` to 0.
  - Iterates through each index, multiplying the corresponding element from `matrixA` with the appropriate element from `matrixB` and adding the result to `sum`.
  - Stores the resulting `sum` in `localRow` at position `colIdxB`.

4. **Gathering Results:**
   - Once the multiplication is complete, each node sends its `localRow` vector back to the root node using a gather operation.
   - The root node collects and combines these rows to form the final result matrix.

5.5.2. Timing Cost

Timing cost measures the number of ticks required to complete the matrix multiplication operation. This metric is crucial for understanding the efficiency and speed of the computation process.

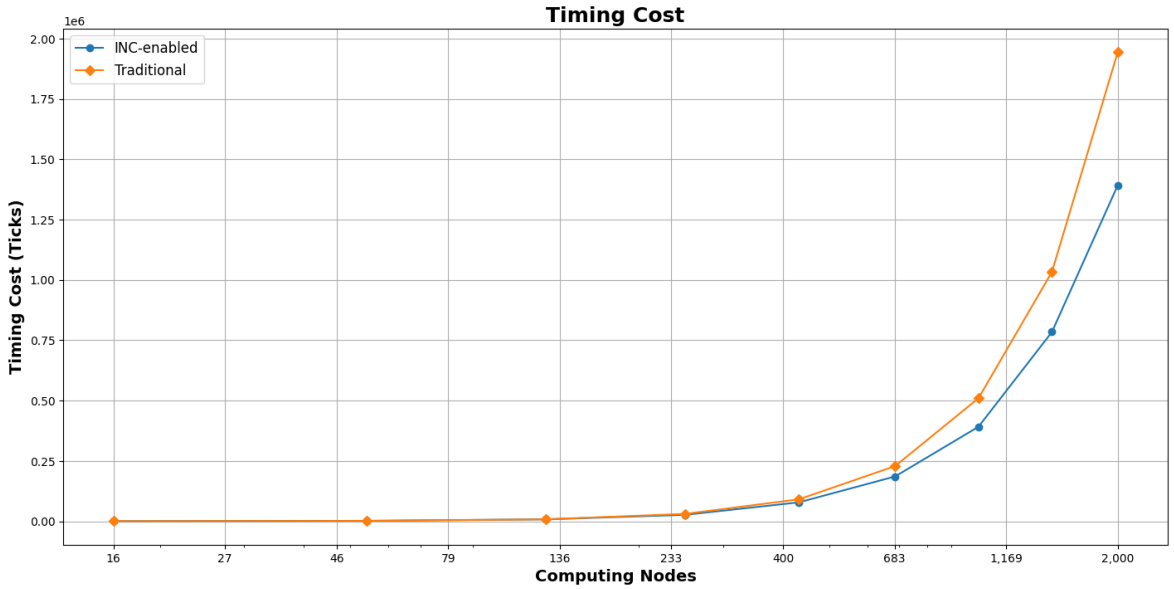

Figure 5.29 Matrix Multiplication Timing Cost

As depicted in Figure 5.29, the timing cost for the traditional method increases significantly with the number of ports. This escalation is due to the higher computational and communication overhead as the system scales. In contrast, the INC-enabled scenario maintains a lower timing cost across all port configurations. This timing improvement demonstrates how INC effectively leverages network resources to optimize computation and data distribution, resulting in faster matrix multiplication operations.

5.5.3. Bandwidth Usage

Bandwidth usage quantifies the total number of messages and the total number of bytes transmitted within the network during the matrix multiplication operation. Efficient bandwidth utilization is essential to minimize network congestion and ensure smooth data flow.

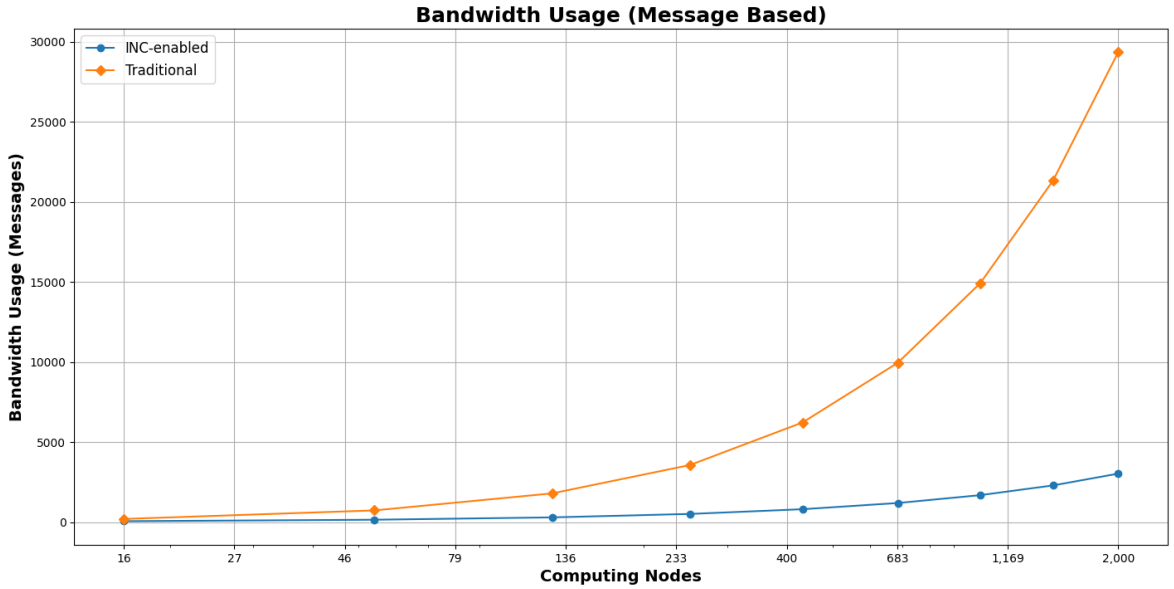

Figure 5.30 Matrix Multiplication Bandwidth Usage (Message)

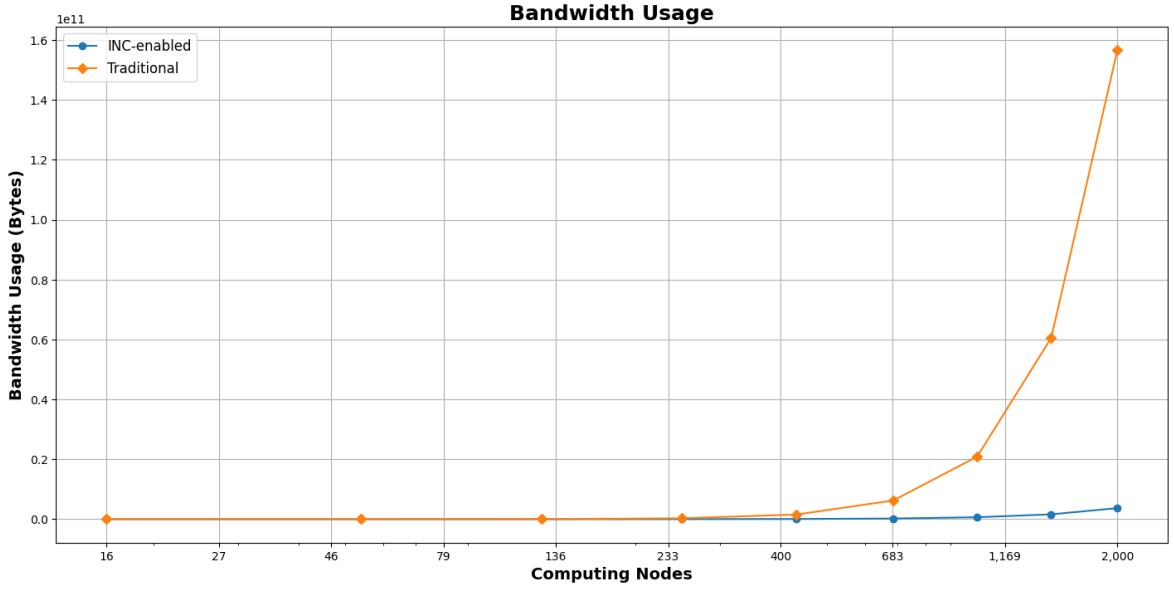

Figure 5.31 Matrix Multiplication Bandwidth Usage (Byte)

Figure 5.30 and Figure 5.31 illustrates that the traditional approach experiences a sharp increase in bandwidth usage as the number of ports grows. This trend is attributed to the direct communication between nodes, which leads to a higher volume of messages being transmitted. Conversely, the INC-enabled configuration significantly reduces bandwidth

usage by offloading data distribution and aggregation tasks to the network switches. This optimization results in fewer messages and bytes being sent across the network, thereby enhancing bandwidth efficiency and reducing the potential for congestion.

5.5.4. Completion Time Difference

Completion time difference measures the variation in operation completion times across different computing nodes, reflecting the consistency of synchronization during the matrix multiplication process.

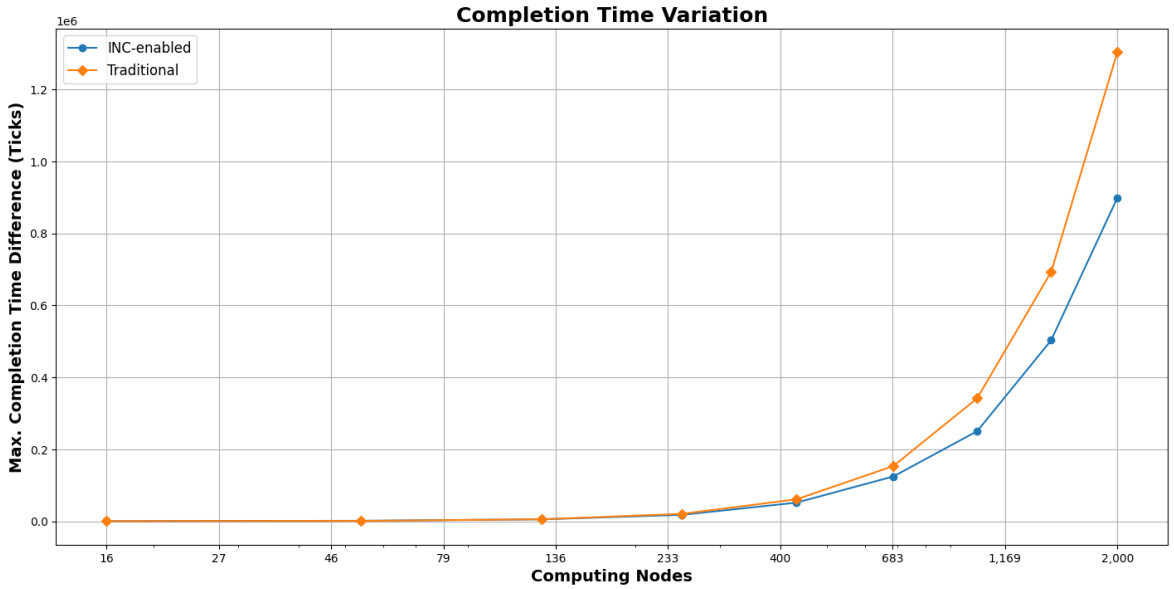

Figure 5.32 Matrix Multiplication Completion Time Difference

As shown in Figure 5.32, the completion time difference remains minimal and stable in the INC-enabled scenario, regardless of the number of ports. This low variation indicates highly consistent synchronization across all computing nodes, which is critical for the accuracy and efficiency of parallel matrix multiplication. In contrast, the traditional method exhibits a significant increase in completion time difference with higher port counts, suggesting less uniform synchronization and potential delays in computation. The improved synchronization quality in the INC-enabled setup underscores its ability to maintain consistent performance across scalable systems.

This sub-section highlights the advantages of using In-Network Computing for matrix multiplication by demonstrating reduced timing costs, lower bandwidth usage, and improved synchronization consistency compared to traditional methods. Let me know if you need further refinements or additional details!

6. CONCLUSION

This thesis has investigated the effects of in-network computing (INC) on the performance of collective communication operations in parallel computing systems using a fat-tree network topology. Through the development and implementation of a custom INC-enabled simulator, we evaluated the impact of INC across a variety of collective operations—including barrier synchronization, broadcast, scatter, gather, reduce, all-reduce, and matrix multiplication—focusing on metrics such as timing cost, bandwidth usage, and completion time difference.

6.1. Key Findings

The results demonstrate that INC substantially improves timing costs and reduces bandwidth usage in most collective operations, showcasing INC's ability to alleviate network congestion and accelerate communication. For example, INC consistently led to lower timing costs in all collective operations by leveraging intermediate switches to process partial data and reduce traffic. These improvements were most pronounced in configurations with larger port counts, where traditional methods typically experience performance degradation due to increased data loads.

In terms of completion time difference, INC-enabled operations exhibited greater synchronization consistency, as reflected in more uniform completion times across computing nodes. This improvement in synchronization quality is crucial for maintaining reliable and consistent performance in parallel systems, where timing variance can impact the overall efficiency of distributed applications. However, INC's impact on direct messages was minimal, as these operations involve straightforward data forwarding that does not benefit from in-network computation.

6.2. Limitations and Future Work

While this study provides promising insights into the potential of INC for parallel computing, several limitations should be acknowledged. First, the research is limited to a fat-tree network topology; future work could expand this analysis to other topologies, such as torus or hypercube, to explore whether INC's benefits are similarly pronounced in different network structures. Additionally, the simulator was designed to operate independently of hardware-specific optimizations, leaving room for future studies to explore hardware-optimized INC implementations, which could provide further performance gains.

Another avenue for future research is to assess the resilience of INC-enabled networks under varying network conditions, such as high fault rates or heavy load fluctuations. This would involve examining how INC performs under real-world scenarios where network reliability can fluctuate, as well as implementing more sophisticated error-handling mechanisms within the network. Furthermore, exploring hybrid INC models that combine software-based INC with hardware acceleration could yield a deeper understanding of how different INC strategies perform under diverse computational workloads.

In conclusion, this thesis underscores the potential of INC to optimize collective communication in parallel computing systems, significantly enhancing scalability, latency, and synchronization quality. By addressing the identified limitations and exploring further applications and network topologies, future research can build on this work to expand INC's role in high-performance computing environments.

REFERENCES

[1] M. G. Venkata, G. Bloch, G. Shainer, and R. Graham. Accelerating openshmem collectives using in-network computing approach. In *2019 31st International Symposium on Computer Architecture and High-Performance Computing (SBAC-PAD)*, pages 212–219. **2019**. doi:10.1109/SBAC-PAD.2019.00042.

[2] S. Chen, W. He, F. Qi, Y. Zheng, and K. Yu. Hybrid approach to optimize mpi collectives by in-network-computation and point-to-point messages. In *2022 7th International Conference on Computer and Communication Systems (ICCCS)*, pages 773–783. **2022**. doi:10.1109/ICCCS55155.2022.9846190.

[3] Y. Liu, J. Zhang, S. Liu, Q. Wang, W. Dai, and R. C. C. Cheung. Scalable fully pipelined hardware architecture for in-network aggregated allreduce communication. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 68(10):4194–4206, **2021**. doi:10.1109/TCSI.2021.3098841.

[4] F. Yang, Z. Wang, X. Ma, G. Yuan, and X. An. Switchagg: A further step towards in-network computing. In *2019 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*, pages 36–45. **2019**. doi:10.1109/ISPA-BDCloud-SustainCom-SocialCom48970.2019.00017.

[5] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. Optimization of collective communication operations in mpich. *The International Journal of High Performance Computing Applications*, 19(1):49–66, **2005**. doi:10.1177/1094342005051521.

[6] Y. Tokusashi, H. Matsutani, and N. Zilberman. Lake: The power of in-network computing. In *2018 International Conference on ReConFigurable Computing*

and FPGAs (ReConFig), pages 1–8. **2018**. doi:10.1109/RECONFIG.2018.8641696.

[7] F. Yang, Z. Wang, X. Ma, G. Yuan, and X. An. Understanding the performance of in-network computing: A case study. In *2019 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*, pages 26–35. **2019**. doi:10.1109/ISPA-BDCloud-SustainCom-SocialCom48970.2019.00016.

[8] Amedeo Sapio, Ibrahim Abdelaziz, Abdulla Aldilaijan, Marco Canini, and Panos Kalnis. In-network computation is a dumb idea whose time has come. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, HotNets '17, page 150–156. Association for Computing Machinery, New York, NY, USA, **2017**. ISBN 9781450355698. doi:10.1145/3152434.3152461.

[9] Sameer Kumar, Sameh S. Sharkawi, and K. A. Nysal Jan. Optimization and analysis of mpi collective communication on fat-tree networks. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1031–1040. **2016**. doi:10.1109/IPDPS.2016.85.

[10] K. Imre. Simulating the programmable networks for hla compatible high-performance simulators. In *2022 International Conference on Modelling and Simulation (ECMS) Proceedings*. **2022**. doi:10.7148/2022-0291.

[11] Caglayan Dokme. In-Network Computing Simulator for Parallel Programming. `https://github.com/CaglayanDokme/In_NetworkComputing/releases/tag/v1.0.0`, **2025**.