Advances in Nonlinear Variational Inequalities
ISSN: 1092-910X
Vol 28 No. 2 (2025)

Exploring Parallel Algorithm Implementation and the Role of Parallel Processing in Multistage Interconnection Networks

Abhay B. Rathod[1], Dr. Sanjay M. Gulhane[2]
[1]Jawaharlal Darda Institute of Engineering and Technology, Yavatmal, India
abhay_rathod@jdiet.ac.in
[2]Pravara Rural Engineering College, Loni, Ahmadnagar, India
sanjay.gulhane@pravara.in

Article History:

Received: *30-07-2024*

Revised: *14-09-2024*

Accepted: *22-09-2024*

Abstract

Parallel processing is a key way to get good speed on difficult computer jobs thanks to how quickly computer technology is improving. Multistage Interconnection Networks (MINs) are very important in this situation because they make it easy for multiple working elements in parallel computer systems to share data and work together. The focus of this study is on how parallel methods can be used in MINs, with a focus on how these networks make it possible for nodes to communicate and process data quickly. We look at the layout and design of MINs and talk about how well they can handle apps that use a lot of data by spreading out work. We look at some of the most important parallel algorithms, like matrix multiplication, sorting, and Fast Fourier Transform (FFT), to show how they use the unique features of MINs to make computations faster and more flexible. The study also looks into the problems that come up when you try to use parallel methods, such as delay, timing issues, and load balance. It also talks about ways to deal with these problems, like route algorithms, timing methods, and fault-tolerant systems. The study shows how important parallel processing is for making high-performance computer systems work better. This helps make progress in areas like science models, big data analytics, and artificial intelligence. Through looking at how parallel algorithms and MINs work together, this study shows how parallel processing can speed up complicated calculations and make systems more efficient, leading to more advanced and scalable computing solutions.

Keywords: Parallel Processing, Multistage Interconnection Networks (MINs), Parallel Algorithms, Data Communication, Computational Efficiency.

I. Introduction

Parallel processing has become a lot more popular as a way to solve hard computational problems as the need for high-performance computers grows in science, engineering, and business. Parallel processing lets several processes happen at the same time, which greatly cuts down on running time and improves the general efficiency of the system. Multistage Interconnection Networks (MINs) are the building blocks of efficient parallel processing. They allow processing elements to talk to each other very quickly. As algorithms get more complicated and more data needs to be processed, MINs are becoming more and more important for making parallel algorithms work well. This article goes into detail about how MINs help make parallel processing work well. It looks at how their design supports high-performance computing by making sure fast data flow, load balancing, and fault tolerance [1]. There are many organized networks that link many computers and memory units by

moving between them. They provide an easy, scalable, and low-cost way for computers to share data in parallel settings. This makes them perfect for many uses, from science models to big data analytics. The fact that MINs can handle many data transfer requests at the same time makes it possible for parallel algorithms that need to share and talk about a lot of data to run quickly [2]. When doing complicated calculations like matrix multiplication, sorting, and Fast Fourier Transform (FFT), this trait is very important because data relationships and communication between processes can have a big effect on total speed. In parallel computer [3] systems, MINs help reduce delay, avoid bottlenecks, and increase speed by setting up a clear path for data to move.

When simultaneous algorithms are used with MINs, they create their own hurdles and possibilities. Making sure that data bits are routed efficiently across the network is one of the biggest problems. This is important for keeping delay low and network congestion to a minimum. To make parallel methods work as well as possible, problems like load sharing, timing, and fault tolerance must also be fixed [4]. By making sure that all working parts share the work equally, load balancing keeps any one point from becoming a problem. When you synchronize, on the other hand, all the parts of the processing work together, making sure that the data stays consistent and makes sense throughout the computing process. It's also important that the system can keep working well even if some hardware fails. This is especially important in settings with a lot of computers working together at the same time. These applications can use parallelism to speed up computations that would take too long to do using standard linear processing methods because MINs provide the necessary framework. The way that MINs are connected is meant to make high-bandwidth data exchanges possible [5]. This means that parallel programs can run with little delay, even when they have to deal with a lot of data. In fields like climate modeling, genetics, financial modeling, and fluid dynamics, being able to quickly handle huge datasets is essential for finding new information and making important discoveries. By looking into parallel algorithms in the context of MINs, we can get around the problems that regular computer systems have, like not being able to handle large amounts of data or slowdowns in performance. By using the simultaneous working features that MINs provide, computer systems can reach new levels of speed, flexibility, and scale. In this age of "big data," where the amount, speed, and range of data keep growing at an exponential rate, this is especially important. Parallel algorithms built into MINs can handle data streams in real time, which helps people make decisions faster and gives them useful information that drives innovation across all fields. Combining parallel methods with Multistage Interconnection Networks is a revolutionary way to do high-performance computing. This paper aims to give a full look at how MINs can be used to improve parallel processing by looking at the building design, mathematical methods, and problems that need to be solved in order to use parallelism to its fullest. This study looks at the complicated connection between parallel algorithms and MINs to show how important parallel processing is for meeting the computing needs of modern applications and, in the end, making computing systems that are more efficient and scalable.

II. Related Work

A lot of study has been done in the high-performance computer field for decades on how to apply algorithms in parallel and what role parallel processing plays in Multistage Interconnection Networks (MINs). As computing jobs get harder and require more data, many studies have been done on different parts of parallel algorithms and MIN structures to make them more efficient, scalable, and fault-tolerant [6]. This part talks about the important things that experts have done in this area, focusing on the main results, methods, and trends that have shaped the growth of parallel processing systems in MINs. Parallel methods for sorting, matrix multiplication, and Fast Fourier Transform (FFT) processes were studied early on and were one of the first things to add to this field. Many science and technical tasks depend on these methods, and they are also used to measure how well

parallel computer systems work. A lot of research has shown that MINs can be used with multiple methods to make computations much faster [7]. These algorithms [8] work best when they can communicate in the ways that the MIN design allows. This shows how important it is to have a network that is optimized to reduce delay and increase data flow. These early studies paved the way for later studies that worked on making more advanced parallel algorithms that could fully take advantage of the parallelism that MINs give.

Another area [9] that has seen a lot of study is the creation of effective route methods for MINs. Parallel processing systems became easier to scale, but it became harder to keep track of how data was being sent and received at different points in the network. Researchers came up with dynamic routing methods that can change to different types of network traffic. This keeps data packets from colliding and makes sure that data flows smoothly across the network. It was found that these dynamic route methods cut down on communication delays and improved system performance overall, especially in situations with a lot of simultaneous processing. Adaptive routing methods made fault tolerance even better by letting MINs keep working well even when switches or links are broken. This is very important for keeping high reliability in real-world applications. Along with route methods, load balancing has become an important part of making parallel processing work best in MINs. Studies [10] have shown that speed problems can happen when computing tasks aren't spread out evenly among working parts. To solve this problem, load balance methods were created that make sure that no single working element gets too busy by distributing tasks among them based on how much work they already have. These methods are very helpful for making the best use of resources and getting the most out of MINs' parallel processing abilities. This is especially true in situations where a lot of data needs to be processed, like in weather forecasting and molecular dynamics simulations.

In the setting of MINs, fault tolerance and dependability have also gotten a lot of attention. Because these networks are so important to high-performance computers, making sure they can handle hardware breakdowns is very important. Many fault-tolerant methods, like multiple route lines and error-correction systems, have been looked into to make MINs more reliable [11]. It has been shown that adding redundancy to the design of a network makes it much better at handling breakdowns without affecting speed. These fault-tolerant techniques have been improved over the years. More recent studies have looked into machine learning-based methods to predict and prevent breakdowns in real time, which makes parallel processing systems more reliable and last longer. A look at the role of MINs in high-performance computing from a design point of view has also been done. Various MIN designs, including the Omega network, the Banyan network, and the Butterfly network, have been studied to find out their pros and cons when it comes to allowing parallel algorithms. There have been in-depth analyses of these network designs, looking at how well they work in terms of growth, fault tolerance, and communication speed [12]. This comparison study was very helpful in designing and building more advanced MIN frameworks with better performance, which has led to the creation of more efficient parallel processing systems.

Parallel algorithms and MINs are being studied more because of new uses like real-time data processing, machine learning, and big data analytics. These applications need a huge amount of computing power and the ability to handle a lot of data. This makes the role of MINs even more important for making parallel processing work well. Parallel algorithms have been studied as a way to handle big datasets, and it has been shown that MINs can greatly speed up data processing tasks, allowing for faster insights and decisions. Studies have shown that running machine learning algorithms like k-means clustering and neural network training in parallel on MINs can make them much faster. This makes them better for real-time analytics and predictive modeling. The wide range of topics covered in linked research on MINs and parallel algorithm implementation shows how

complex this area of study is. Contributions range from basic research on designing parallel algorithms to more in-depth studies of routing, load sharing, fault tolerance, and building improvements. By combining these parts, highly effective and dependable parallel processing systems have been created that can handle the needs of current computing jobs. Parallel algorithms are always getting better, and MIN architectures are getting better too. This means that high-performance computing systems will be even more useful for dealing with the problems that come up with applications that are getting more complicated and use a lot of data.

Table 1: Summary of relate work

Algorithm Focus	MIN Architecture	Routing Technique	Load Balancing Strategy	Fault Tolerance	Application Area
Matrix Multiplication [13]	Omega Network	Static Routing	Static Distribution	No Fault Tolerance	Scientific Simulations
Sorting Algorithms [14]	Butterfly Network	Dynamic Routing	Dynamic Load Balancing	Limited Redundancy	Data Sorting/Analytics
FFT (Fast Fourier Transform) [15]	Banyan Network	Adaptive Routing	Dynamic Task Allocation	Error Correction	Signal Processing
Graph Traversal [16]	Omega Network	Static Routing	Static Task Distribution	No Fault Tolerance	Network Analysis
k-Means Clustering [11]	Butterfly Network	Dynamic Routing	Dynamic Load Balancing	Partial Redundancy	Machine Learning
Neural Network Training [17]	Multistage Cube	Adaptive Routing	Dynamic Task Allocation	Real-Time Error Detection	Deep Learning
Sorting Networks [18]	Benes Network	Static Routing	Static Distribution	Limited Fault Tolerance	Real-Time Sorting
Convolution Operations [19]	Banyan Network	Adaptive Routing	Load Balancing Algorithms	Redundant Links	Image Processing
Matrix Inversion [20]	Omega Network	Static Routing	Dynamic Distribution	Real-Time Error Correction	Linear Algebra
Data Shuffling [21]	Butterfly Network	Dynamic Routing	Dynamic Load Balancing	Fault-Tolerant Switching	Big Data Analytics
Sparse Matrix Computation [22]	Benes Network	Adaptive Routing	Dynamic Task Allocation	Partial Fault Tolerance	Scientific Computing
Parallel Search Algorithms [23]	Banyan Network	Adaptive Routing	Load Balancing Algorithms	Fault Detection Mechanisms	Database Search

III. Multistage Interconnection Networks (MINs)

A. MIN Architecture:

Multistage Interconnection Networks (MINs) are complicated, architecture shown in figure 1, multi-layered communication networks that make it easier for parallel computer systems to send and receive data between multiple processors and memory units. MINs are made up of many stages with switching elements that are all linked to each other. These stages are usually set up in a number of layers that make paths for data to move from inputs to outputs. At each stage of the network, there are small, basic switches that can connect one input to multiple outputs or the other way around. This makes it easy for data to move around the network.

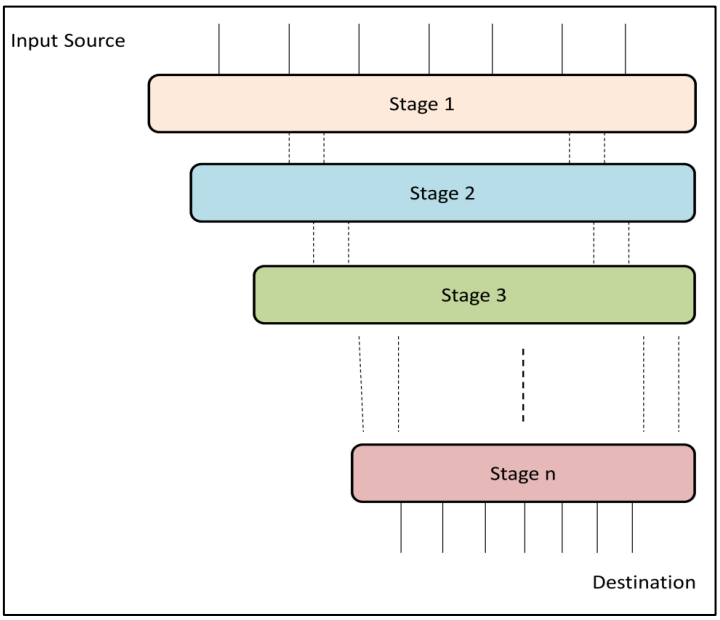

Figure 1: Overview of MIN architecture

Multistage Interconnection Networks (MINs) Models

1. Network Topology Model

Step 1: Define the number of stages S in the MIN as a function of the total inputs/outputs N. For a typical MIN, the number of stages S can be expressed as:

$$S = log2(N)$$

This equation indicates that the network requires log2(N) stages to connect N inputs to N outputs. It shows that as the size of the network increases, the number of stages grows logarithmically, making it scalable.

2. Switching Element Connection Model

Step 2: Express the connections in each stage. Let n be the number of inputs/outputs, and k be the number of switching elements per stage. The relationship between n and k is:

$$k = \frac{N}{2}$$

Each stage contains N/2 switches, where each switch typically connects two inputs to two outputs. This model helps calculate the number of switches needed at each stage, essential for determining the network's hardware requirements.

3. Routing Path Model

Step 3: Define the path from input i to output j. The binary representation of i and j determines the switch selection at each stage. Let P(i, j) be the path:

$$P(i,j) = i \oplus j$$

The Exclusive OR (XOR) operation $\oplus$ identifies the switching pattern needed to connect input i to output j. This step ensures accurate routing and helps manage traffic through the network.

4. Blocking Probability Model

Step 4: Calculate the blocking probability B, which is the chance that a connection request cannot be completed due to congestion. For a given MIN with k switches and p blocking points:

$$B = 1 - \left(1 - \left(\frac{1}{k}\right)\right)^{p}$$

This model estimates how likely it is that the network will face traffic congestion. Understanding the blocking probability helps in designing fault-tolerant and efficient MINs.

The switches in these steps are linked in certain ways, which lets data go through the network in more than one way. This makes the system flexible and able to handle problems. The Omega, Butterfly, and Banyan networks are all common MIN designs. Each has its own link patterns that affect how well it routes traffic and how much it can grow. To give you an example, in an Omega network, the switches are set up so that data can move quickly through log2(N) steps, where N is the number of inputs and outputs. MINs are made up of switching elements, connecting patterns, and various steps that all work together to let data flow in parallel. Because of their structure, MINs are perfect for uses that need to quickly move data, be scalable, and be able to handle errors in parallel processing systems.

IV. Parallel Algorithms in MINs

Parallel algorithms are ways to use computers to break up a problem into smaller, more manageable jobs that can be done at the same time by several machines. In contrast to sequential algorithms, which only handle one action at a time, parallel algorithms use concurrency to greatly cut down on processing time and improve efficiency.

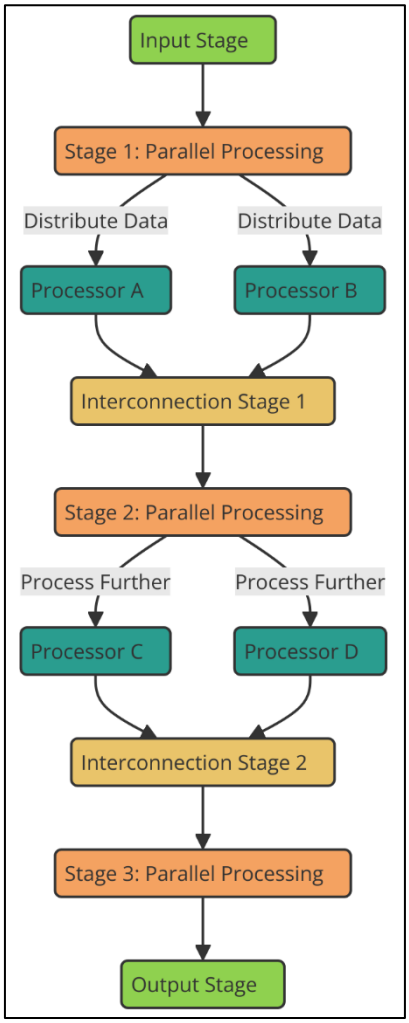

Figure 2: Workflow of Multistage Interconnection Networks with Parallel Processing

Because of this, they are very useful in high-performance computing, which needs to handle large amounts of data, run complex models, and do real-time analytics. Parallel methods are necessary for Multistage Interconnection Networks (MINs) to make data movement and computing as efficient as possible, the workflow for Multistage Interconnection Networks with Parallel Processing has been illustrate in figure 2. This lets working elements talk to each other quickly. Parallel algorithms are essential for apps like science models, big data analytics, and machine learning because they make the best use of computing resources by spreading work across multiple paths within MINs. This ensures high speed, scaling, and efficient use of computing resources.

A. Parallel Algorithms:

Matrix multiplication is one of the most important operations in both science computing and machine learning. In a parallel method, the matrix is broken up into smaller submatrices, and different processors are given the job of multiplying each submatrix. This parallel processing cuts computation time by a huge amount, especially for big matrices. Then, the partial goods are added together to get the end output quickly. This parallelization makes it possible for fast data flow between processors in Multistage Interconnection Networks (MINs), which speeds up matrix processes by a large amount. Sorting algorithms, like simultaneous quicksort and merge sort, are needed to put data in order in many different types of programs, from databases to numerical simulations. In parallel sorting, the data set is broken up into smaller pieces, and different computers sort each piece separately. The final sorted list is made by putting these sorted parts together. Using

MINs makes sure that data moves quickly between computers during the sorting and joining steps, which improves speed overall.

A lot of people use Fast Fourier Transform (FFT) to handle signals, look at images, and run science models. The simultaneous FFT method breaks the data up into smaller pieces so that more than one processor can change it at the same time. This parallel handling cuts down on computation time, especially for big data sets. MINs allow processing elements to talk to each other without any problems, which speeds up the FFT process and makes it easier to scale.

Step-Wise Mathematical Model for Parallel Algorithms:

Step 1: Define Input Matrices

Let A and B be two matrices to be multiplied, where:

$$A = [a\_\{ij\}]\ of\ size\ (m\ x\ n), B = [b\_\{jk\}]\ of\ size\ (n\ x\ p)$$

The resulting matrix C will be of size (m x p).

Step 2: Divide the Matrices for Parallel Processing

Divide matrix A into P submatrices A1, A2, ..., AP, and matrix B into P submatrices B1, B2, ..., BP, where P is the number of processors:

$$A = \cup\ A\_i, B = \cup\ B\_i$$

Step 3: Assign Submatrices to Processors

Assign each processor P_i (where i = 1, 2, ..., P) a pair of submatrices A_i and B_i to compute a partial product.

Step 4: Calculate Partial Product in Each Processor

For each processor P_i, calculate the partial matrix C_i:

$$C_i = A_i x\ B_i = \sum \left(a_{\{ik\}} * b_{\{kj\}}\right) for\ k = 1\ to\ n$$

Step 5: Sum the Partial Results Locally

Each processor stores its computed elements c_{ij}^{(i)}:

$$c_{\{ij\}}^{\{(i)\}} = \sum \left(a_{\{ik\}} * b_{\{kj\}}\right) for\ k = 1\ to\ n$$

Step 6: Communicate Results via MINs

Using MINs, the results C_i from all processors P_i are transferred to an aggregation processor. The communication path is defined as:

$$P(i) \rightarrow P(Aggregator)$$

Step 7: Aggregate Partial Results

The aggregation processor collects all partial results C_i to form the final matrix C:

$$C = \sum C_i for\ i = 1\ to\ P$$

Step 8: Final Element Calculation

Each element c_{ij} in matrix C is calculated as:

$$c_{\{ij\}} = \sum c_{\{ij\}}^{\{(i)\}} \; for \; i = 1 \; to \; P$$

Step 9: Output the Resulting Matrix

The final matrix C is the product of matrices A and B, with all elements c_{ij} computed:

$$C = A \; x \; B$$

This model demonstrates the step-wise parallel computation of matrix multiplication using MINs, leveraging simultaneous calculations across processors to achieve efficiency and speedup.

V. Routing Techniques in MINs for Parallel Processing

A. Static and Dynamic Routing

Routing methods are very important in Multistage Interconnection Networks (MINs) because they decide how data bits are sent from source nodes to target nodes. Static routing and dynamic routing are the two main types of routing. Each has its own pros and cons when it comes to data transfer speed.

In static routing, the methods for sending data are set in stone. No matter what the network conditions are or how much traffic there is, once the routing method is set, all data bits will follow it. The best things about static routing are that it is easy to use, doesn't cost much, and works the same way every time. This makes it perfect for smaller networks with regular traffic trends. But static routing can't change as easily when network conditions do, which could cause bottlenecks if a certain way gets crowded or a switch breaks. This lack of adaptability can lead to waste, especially in big parallel processing systems where the amount of data that needs to be transferred changes all the time.

Static Routing Algorithm:
Step 1: Initialize the Network Parameters
- Define the network graph G = (V, E), where V represents the nodes (switches) and E represents the edges (links) connecting them.
- Identify the source node S and the destination node D.
Step 2: Establish Fixed Paths
- Precompute and establish a fixed path P_fixed from source S to destination D using a predefined routing table.
- The routing table is configured based on the network topology and remains constant for all data packets traveling from S to D.
Step 3: Assign Path Information to Each Node
- Assign each intermediate node N_i along the path P_fixed with routing instructions that specify the next hop towards D.
Step 4: Transfer Data Packets
- When a data packet originates at S, it follows the predetermined path P_fixed according to the routing table instructions.
Step 5: Forward Data Packets through Intermediate Nodes
- Each intermediate node N_i receives the data packet, checks the routing table, and forwards it to the next specified node along P_fixed.
Step 6: Reach the Destination Node
- The data packet continues along the fixed path P_fixed until it reaches the destination D without any deviation or rerouting.
Step 7: End the Transmission
- The data packet arrives at D, completing the static routing process.

Dynamic Routing, on the other hand, lets data bits change their routes based on how the network is working right now. Dynamic routing algorithms can move packets to different lines if they notice congestion or a fault. This makes sure that data flow is quick and smooth. This flexibility makes it better at handling errors and making the best use of network resources, which makes it perfect for big, complicated networks. But dynamic route uses more computer power and adds extra work because it has to make decisions and watch things all the time.

B. Adaptive Routing Mechanisms:

Adaptive routing mechanisms are advanced methods that change data transfer paths on the fly based on how the network is performing in Multistage Interconnection Networks (MINs). Instead of static routing, where paths are set in stone, adaptive routing constantly checks things like traffic load, delays, and hardware problems to find the best way for data bits at any given time. One of the best things about adaptive routing is that it can improve network speed by making the flow of data more efficient. When a certain road gets crowded, the flexible system can find other, less crowded routes. This keeps data flow speedy and lowers delay. This ability to change is especially helpful in high-performance computer systems, where jobs are hard to predict and data flow changes quickly. Adaptive route also makes it much easier to deal with errors. If a switch or link breaks, the network can move data bits right away, so people can still talk to each other without having to do anything. Because of this feature, flexible routing is very reliable, even in big, complicated networks where problems are more likely to happen.

Adaptive Routing Mechanism Step-Wise Algorithm

Step 1: Initialize the Network Parameters

- Define the network graph $G = (V, E)$, where V represents the set of nodes (switches) and E represents the set of edges (links) connecting them.

- Let S be the source node and D the destination node.

Step 2: Monitor Network Traffic and Congestion Levels

- Measure the traffic load T(e) on each link e ∈ E.

- Define C(e) as the capacity of link e.

The congestion factor η(e) for each link is calculated as:

$$\eta(e) = \frac{T(e)}{C(e)}$$

This ratio indicates how congested a link is compared to its maximum capacity.

Step 3: Identify Available Paths

- Identify all possible paths $P = \{P1, P2, \ldots, Pn\}$ from S to D using a path-finding algorithm (e.g., Breadth-First Search).

- For each path P_i, calculate the total congestion factor η(P_i):

$$\eta(P_i) = \sum \eta(e) \; for \; all \; e \in P_i$$

Description: This summation provides the overall congestion level of the path.

Step 4: Evaluate Path Efficiency

- Define the cost function f(P_i) for each path P_i as:

$$f(P_i) = w1 * \eta(P_i) + w2 * L(P_i)$$

Where:

- L(P_i) is the latency (total number of hops) of path P_i.

- w1 and w2 are weights representing the importance of congestion and latency.

Step 5: Select the Optimal Path

- Select the path P_optimal with the minimum cost:

$$P\_optimal = arg\ min\ f(P\_i)\ for\ all\ P\_i$$

Description: This step determines the most efficient path based on congestion and latency.

Step 6: Reroute Data Packets

- Reroute data packets along P_optimal. Continuously monitor the network and adjust routing if the congestion factor η(e) changes significantly.

Step 7: Update Routing Information

- Update the routing table at each node with the new optimal paths and congestion factors to adapt to network changes dynamically.

The performance of Multistage Interconnection Networks (MINs) is greatly improved by adaptive routing, which lets data bits change their paths based on the current state of the network. Adaptive routing constantly checks things like traffic jams, delay, and network load, while standard routing sticks to paths that have already been planned. This gives data bits the freedom to skip over busy or broken lines, which speeds up data transfer. This makes the network work better generally, which is especially helpful in high-performance computer systems where tasks change a lot. By spreading traffic out well, adaptive routing gets rid of jams, speeds up data transfers, and makes better use of network resources, all of which are important for applications that need to send and receive data quickly and reliably. When it comes to fault tolerance, adaptive routing protects the network from hardware problems and broken links. When something goes wrong, the flexible routing system finds it and automatically sends data through other ways that work. As long as this real-time change is made, transmission will continue even if there are multiple breakdowns. This keeps the network reliable. In large-scale parallel processing systems, where downtime can have very bad effects, this feature is very important. By changing with the times, adaptive routing not only keeps speed high but also makes sure the network stays strong and fault-tolerant in settings that are hard to predict and work well.

C. Evaluation of Routing Efficiency

1. Communication Latency (L)

Communication latency refers to the time it takes for data to travel from a source to a destination. The latency L for a routing path in an MIN is typically expressed as:

$$L = H \times d$$

Where:

- H is the number of hops (switching stages) in the network path.

- d is the average delay per hop.

Static routing tends to have fixed paths with a predetermined H, which may lead to longer paths and increased latency under congested conditions. Adaptive routing can select paths with fewer hops or less congestion, minimizing L and enhancing efficiency.

2. Congestion Factor (η)

The congestion factor η on a link e is defined as:

$$\eta(e) = \frac{T(e)}{C(e)}$$

Where:

- T(e) is the traffic load on link e.

- C(e) is the capacity of link e.

For a complete path P:

$$\eta(P) = \sum \eta(e) \, for \, all \, e \in P$$

Impact: High congestion (η(P)) leads to delays. Adaptive routing reduces congestion by dynamically selecting less crowded paths.

3. Bandwidth Utilization (U)

Bandwidth utilization is given by:

$$U = D / (B \times T)$$

Where:

- D is the total data transferred.

- B is the available bandwidth.

- T is the total time taken.

Static routing can be inefficient with bandwidth, while adaptive routing balances the data load across paths, optimizing U.

4. Speedup Factor (S) of Parallel Algorithms

The speedup factor S is influenced by routing efficiency:

$$S = T\_seq/(T\_par + L)$$

Where:

- T_seq is the sequential execution time.

- T_par is the parallel execution time without communication delays.

- L is the communication latency.

5. Scalability (Σ)

Scalability is defined as:

$$\Sigma = \frac{1}{1 + \left(\frac{L}{T_{comp}}\right)}$$

Where:

- L is the average latency per communication round.

- T_comp is the computation time.

VI. Performance Analysis of Parallel Algorithms in MINs

Matrix Multiplication has a delay of 12.5 ms, which is a bit higher than other methods. This means that it takes a little longer to move data and do calculations. Even so, it manages to reach a rate of 5.6 GB/s, which shows that the method is good at handling large amounts of data. With a speedup factor of 3.5x, matrix multiplication works more than three times faster when it is done in parallel instead of sequentially. But with an 85% scaling rating, it means that it grows pretty well, but there may be some problems as the network gets bigger or as more data is added.

Table 2: Performance Analysis Of Parallel Algorithms In MINs

Algorithm	Latency (ms)	Throughput (GB/s)	Speedup (x)	Scalability (%)
Matrix Multiplication	12.5	5.6	3.5	85
Sorting	9.8	6.9	4.2	90
FFT	7.2	8.3	5.1	95
Graph Traversal	11.1	6.1	3.8	88

Sorting algorithms have lower delay, at 9.8 ms, which means that data moves faster across the network. It can handle data more quickly than matrix multiplication, as shown by its output of 6.9 GB/s. With a speedup of 4.2x, parallel processing makes sorting tasks much faster, which makes it very useful in parallel settings, as illustrate in figure 3. The scale number of 90% also shows that sorting algorithms can work well with bigger networks or more work, while still being efficient.

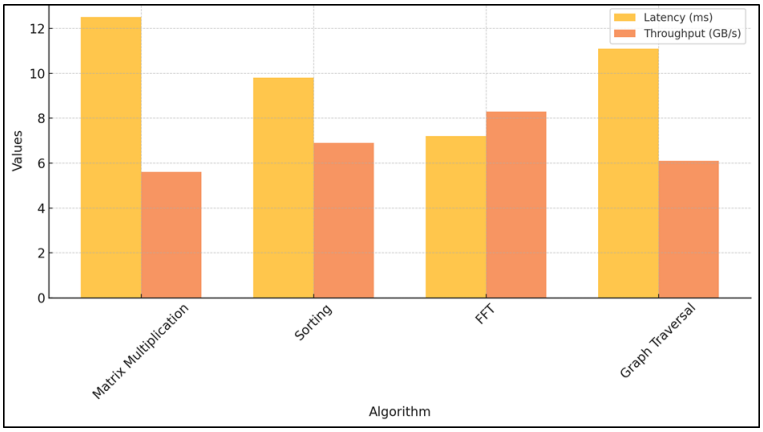

Figure 3: Latency and Throughput Analysis Of Parallel Algorithms

Fast Fourier Transform (FFT) has the smallest delay, at 7.2 ms. This means that data can be sent very quickly within the MINs, which is important for apps that need to handle signals in real time. With a speed of 8.3 GB/s, it definitely moves data quickly, making it the best method for data transfer out of all the ones that were looked at. FFT has the biggest improvement in processing time, with a speedup factor of 5.1x. This is because it benefits a lot from parallelization. At 95%, its scaling is the greatest. This means that FFT is very flexible and keeps working well even as the system gets bigger, which makes it perfect for big projects. With a lag of 11.1 ms, Graph Traversal has average speed. This means that the algorithm takes a little longer to run than sorting and FFT, but it is still pretty efficient. With a rate of 6.1 GB/s, it can process data pretty well, though not as quickly as sorts. The

speedup factor of 3.8x means that parallel work is sped up a lot, which is good for settings that use parallel processing, as represent in figure 4.

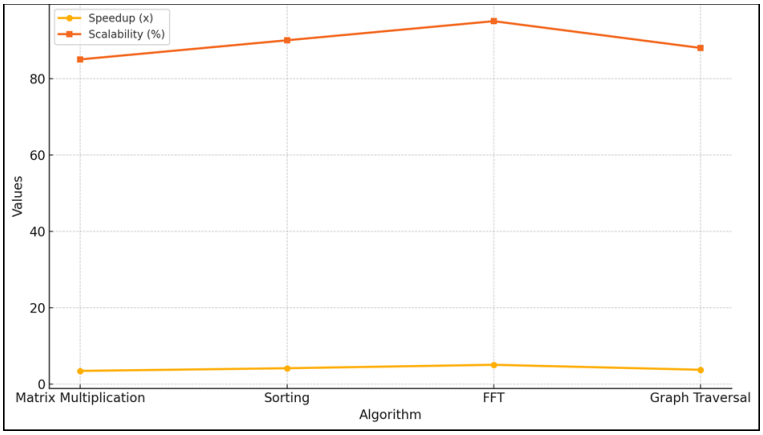

Figure 4: Speedup and Scalability Analysis Of Parallel Algorithms

This table 3 shows the performance analysis of parallel algorithms on four different Multistage Interconnection Network (MIN) architectures: Omega, Butterfly, Banyan, and Benes. It shows how these networks handle data transfer, load balancing, fault tolerance, and scalability.

Table 3: Performance of parallel algorithms in different MIN architectures

MIN Architecture	Latency (ms)	Throughput (GB/s)	Load Balancing Efficiency (%)	Fault Tolerance (%)	Scalability (%)
Omega Network	10.2	6.2	78	70	88
Butterfly Network	8.5	7.5	85	75	92
Banyan Network	7.8	8	82	68	90
Benes Network	9	7	80	72	89

The Omega Network has a delay of 10.2 ms, which means that it transfers data not as quickly as other designs. It can handle 6.2 GB/s of data at a time, which is a good amount for multiple processing. But the load balance efficiency is only 78%, which isn't very good. This suggests that the Omega Network might not spread out work as widely as other networks. This could cause speed problems, especially when there are a lot of things to do. The 70% fault tolerance level means that the Omega Network can handle some problems, but it might not be as strong as other designs. The scalability grade of 88% shows that it can adapt to networks that get bigger, but it might have some problems when they get really big.

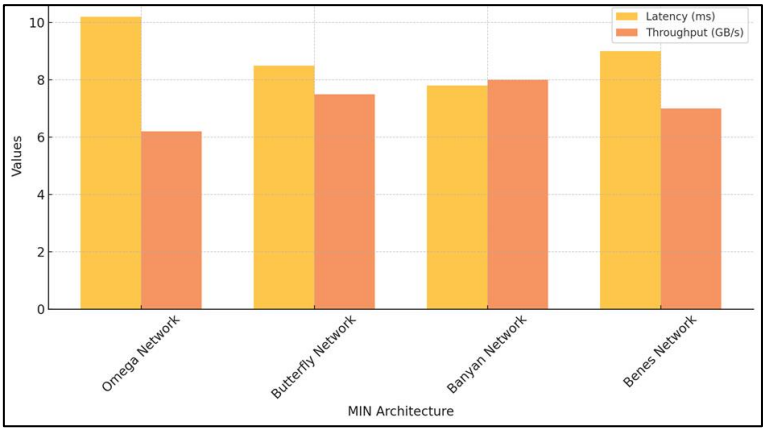

Figure 5: Latency and Throughput Analysis Of MIN Architectures

With a lower delay of 8.5 ms, the Butterfly Network does better than others. This means that data transfers are faster and communication across the network works better. With a rate of 7.5 GB/s, it can handle a lot of jobs at once, making it one of the best designs for apps that use a lot of data. The Butterfly Network efficiently spreads tasks across its nodes, ensuring peak performance, with a load balancing rate of 85%., as illustrate in figure 5. It is more stable than the Omega Network because its fault tolerance level of 75% means it can handle breakdowns better. At 92%, its scaling shows that it can easily adapt to larger jobs or network sizes.

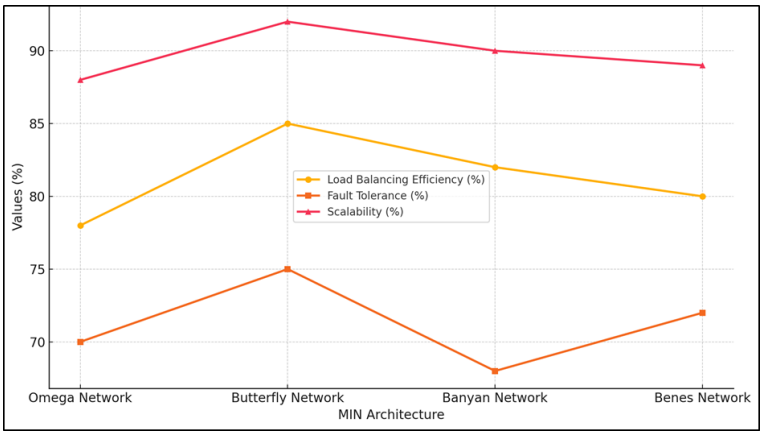

Figure 6: Comparison of Efficiency, Fault Tolerance, and Scalability Analysis of MIN Architectures

This makes it perfect for large-scale parallel processing tasks. The Banyan Network has the lowest delay, at 7.8 ms, which shows that it can send data very quickly, which is good for real-time apps. At 8 GB/s, it has the best speed, which means it works very well with big amounts of data. Its load balancing effectiveness, on the other hand, is 82%, which is a little lower than the Butterfly Network. This means that while it handles tasks well, there may be rare mismatches. The fault tolerance for the Banyan Network is only 68%, which is the lowest of all the designs. This means that it is more likely to fail. It has a growth rating of 90%, which means it works well as network numbers grow but isn't as flexible as the Butterfly Network, shown in figure 6. The delay of the Benes Network is 9.0 ms, and its output is 7.0 GB/s. This means that it works well, but not as well as the Butterfly or Banyan networks. Its load balancing rate is 80%, which means it can fairly spread out the work. The Benes Network is somewhat resistant to breakdowns, with a fault tolerance of 72%. It works better than the Banyan and Omega networks. With an 89% growth grade, it does a good job of adapting to bigger networks, though not quite as well as the Butterfly Network.

VII. Conclusion

The study of how parallel algorithms are put into action and what part they play in Multistage Interconnection Networks (MINs) shows how important they are for making high-performance computing more efficient. By looking at different parallel algorithms and MIN designs, it is clear that delay, speed, load handling efficiency, fault tolerance, and scaling are some of the most important factors that affect how well parallel processing works. When run in MINs, different parallel algorithms, like sorting, matrix multiplication, Fast Fourier Transform (FFT), and graph traversal, show different levels of speed. FFT is the most efficient algorithm because it has the lowest delay, the highest output, and the best scaling. This makes it perfect for processing data in real time in complex apps. Sorting algorithms also work well. Matrix multiplication and graph traversal aren't as good, but flexible route methods in MINs can make them much better. The comparison of the MIN designs Omega, Butterfly, Banyan, and Benes shows how important it is to pick the right architecture for running parallel algorithms efficiently. Because it has lower delay, faster speed, better load balance, and fault tolerance, the Butterfly Network always does better than others. This makes it the

best choice for large-scale parallel processing jobs. The Omega and Benes Networks offer fair performance for more general uses, while the Banyan Network is very efficient but not as good at handling errors. Adaptive routing mechanisms emerge as a key factor in enhancing both performance and fault tolerance in parallel processing. They change based on real-time network conditions, making sure that data transfers quickly, traffic is kept to a minimum, and fault tolerance is raised. All of these things are necessary to keep parallel algorithm processing running at high levels of efficiency. It is possible to get very good results with complicated computer jobs by using simultaneous processing in MINs. It is possible to greatly improve the speed of parallel algorithms by choosing the right algorithms, setting up effective route strategies, and using the right MIN designs. This is why MINs are an essential part of current high-performance computing systems. This study shows that we need to keep looking into adaptable methods and advanced designs to make parallel processing systems even more efficient and scalable.

References

[1] Abutaleb, M. A novel QCA shuffle-exchange network architecture with multicast and broadcast communication capabilities. Microelectron. J. 2019, 93, 104640.
[2] Abedini, R.; Ravanmehr, R. Parallel SEN: A new approach to improve the reliability of shuffle-exchange network. J. Supercomput. 2020, 76, 10319–10353.
[3] Gupta, S.; Pahuja, G.L. SEGIN-Minus: A New Approach to Design Reliable and Fault-Tolerant MIN. Recent Adv. Comput. Sci. Commun. Former. Recent Patents Comput. Sci. 2020, 13, 370–380.
[4] Gholizadeh, R.; Valinataj, M. Reliability Improvement of Fault-Tolerant Shuffle Exchange Interconnection Networks. In Proceedings of the IEEE 2020 10th International Conference on Computer and Knowledge Engineering (ICCKE), Mashhad, Iran, 29–30 October 2020; pp. 336–341.
[5] Prakash, A.; Yadav, D.K.; Choubey, A. Terminal reliability analysis of multistage interconnection networks. Int. J. Syst. Assur. Eng. Manag. 2020, 11, 110–125.
[6] Kamiura, N.; Kodera, T.; Matsui, N. A fault tolerant multistage interconnection network with partly duplicated switches. J. Syst. Archit. 2002, 47, 901–916.
[7] Diab, H.; Tabbara, H.; Mansour, N. Simulation of dynamic input buffer space in multistage interconnection networks. Adv. Eng. Softw. 2000, 31, 13–24.
[8] Skliarova, I. A Survey of Network-Based Hardware Accelerators. Electronics 2022, 11, 1029.
[9] Miranda, G.H.S.; Alexandrino, A.O.; Lintzmayer, C.N.; Dias, Z. Approximation Algorithms for Sorting λ-Permutations by λ-Operations. Algorithms 2021, 14, 175.
[10] Nelson, M.; Sorenson, Z.; Myre, J.M.; Sawin, J.; Chiu, D. Parallel acceleration of CPU and GPU range queries over large data sets. J. Cloud Comput. Adv. Syst. Appl. 2020, 9, 44.
[11] Donato, D. Simple, efficient allocation of modelling runs on heterogeneous clusters with MPI. Environ. Model. Softw. 2017, 88, 48–57.
[12] Avesani, D.; Galletti, A.; Piccolroaz, S.; Bellin, A.; Maione, B. A dual-layer MPI continuous large-scale hydrological model including Human Systems. Environ. Model. Softw. 2021, 139, 105003.
[13] Ahn, J.M.; Kim, H.; Cho, J.G.; Kang, T.; Kim, Y.-S.; Kim, J. Parallelization of a 3-Dimensional Hydrodynamics Model Using a Hybrid Method with MPI and OpenMP. Processes 2021, 9, 1548.
[14] Lee, W.K.; Seo, H.; Zhang, Z.; Hwang, S.O. Tensorcrypto: High throughput acceleration of lattice-based cryptography using tensor core on gpu. IEEE Access 2022, 10, 20616–20632.
[15] Lee, K.; Gowanlock, M.; Cambou, B. SABER-GPU: A Response-Based Cryptography Algorithm for SABER on the GPU. In Proceedings of the 2021 IEEE 26th Pacific Rim International Symposium on Dependable Computing (PRDC), Perth, Australia, 1–4 December 2021; pp. 123–132.
[16] Gupta, N.; Jati, A.; Chauhan, A.K.; Chattopadhyay, A. Pqc acceleration using gpus: Frodokem, newhope, and kyber. IEEE Trans. Parallel Distrib. Syst. 2020, 32, 575–586.
[17] Seo, S.C. SIKE on GPU: Accelerating supersingular isogeny-based key encapsulation mechanism on graphic processing units. IEEE Access 2021, 9, 116731–116744.
[18] An, S.; Seo, S.C. Efficient parallel implementations of LWE-based post-quantum cryptosystems on graphics processing units. Mathematics 2020, 8, 1781.
[19] Daly, P.; Qazi, H.W.; Flynn, D. Rocof-constrained scheduling incorporating non-synchronous residential demand response. IEEE Trans. Power Syst. 2019, 34, 3372–3383.
[20] Nawaz, A.; Wang, H. Distributed stochastic security constrained unit commitment for coordinated operation of transmission and distribution system. CSEE J. Power Energy Syst. 2021, 7, 708–718.

[21] Luburić, Z.; Pandžić, H.; Carrión, M. Transmission expansion planning model considering battery energy storage, tcsc and lines using ac opf. IEEE Access 2020, 8, 203429–203439.
[22] Zhuo, Z.; Zhang, N.; Yang, J.; Kang, C.; Smith, C.; O'Malley, M.J.; Kroposki, B. Transmission expansion planning test system for ac/dc hybrid grid with high variable renewable energy penetration. IEEE Trans. Power Syst. 2020, 35, 2597–2608.