International Information and Engineering Technology Association
Advancing the World of Information and Engineering

Ingénierie des Systèmes d'Information
Vol. 28, No. 1, February, 2023, pp. 239-245
Journal homepage: http://iieta.org/journals/isi

Design of Dynamic Network for Parallel Processing on a Distributed System

Aseel Thamer Ebrahem[1*], Basil Shukr Mahmmood[2]

[1] Department of Computer Engineering, Northern Technical University, Mosul 41002, Iraq
[2] Computer Engineering Department, Collage of Engineering, University of Mosul, Mosul 41002, Iraq

Corresponding Author Email: aseelthamer@ntu.edu.iq

https://doi.org/10.18280/isi.280128

Received: 20 December 2022
Accepted: 30 January 2023

Keywords:
parallel processing, network topologies, threads, dynamic selection

ABSTRACT

Modern computation systems involve multicomputer configurations. Multiple computers enable multiple threads to be executed simultaneously, with the ability to perform the same operations on different processors (computers) at the same time. This paper addresses the building of a software application to be implemented on 8*8 dynamic multistage network exchange depending on the client/server principles so that one can select one type of different topologies that is suitable to the type of application. The network can serve any number of nodes. Two different applications were examined, convolution operations and matrix multiplication. The goal of this paper is to explore the different ways, in which the multistage network topology can simulate supercomputer systems employing large-scale parallel processing. This paper proposes designing parallel systems on a distributed system, running different topologies such as linear systolic, mesh, and hypercube topologies of the parallel processor's networks, and also a dynamic selection of the appropriate network topology based on the nature of the solved problem. Simulation of parallel processing systems on distributed environments mainly done through Socket programming based on JAVA threads.

1. INTRODUCTION

Parallel processing is a method of computing that involves using two or more CPUs or nodes in order to do different parts of a bigger task. Parallel computing is the use of two or more processors (cores, computers) to tackle a single problem in parallel. It is a sort of computer architecture in which numerous processors simultaneously run or process an application or task. Parallel computing enables the execution of complex computations by distributing the workload among numerous processors, all of which are working on the computation at the same time. The majority of supercomputers work on the basis of parallel computing concepts. Parallel processing comes in a variety of flavors, including MMP, SIMD, MISD, SISD, and MIMD. Although SISD computers cannot operate in parallel, a cluster can be formed by linking several of them. A program's execution time can be reduced by distributing a task's multiple parts across several processors. Each processor runs a different section of the program concurrently [1], and when execution is finished, the results are blended and sent back to the main computer. There are various varieties of parallel processing; SIMD and MIMD are two of the most popular varieties. Single instruction multiple data, or SIMD is a form of parallel processing in which a computer contains two or more processors that each handle a different kind of data while yet conforming to the same set of instructions. Another popular method of parallel processing is MIMD, or multiple instructions multiple data, in which each computer has two or more processors, and each executes its own program and receives data from several data streams. In this form of architecture, the concepts of server and client computers will be used. The server's computer transmits commands to client machines for concurrent executions [2]. Distributed system software, also known as middleware, is used to implement the network topology or connections between nodes. Examples of this middleware include socket, remote procedure call, and remote method invocation. Middleware is described as a software layer that sits above the operating system and below the application program, offering a common programming paradigm for distributed systems [3-5]. As examples of using a distributed system to execute parallel programs, in this research we will see the results of executing some mathematical formulas, such as convolution, matrix multiplication. etc. using parallel methods.

The structure of this paper is as follows: A brief history of parallel processing is given in Section 2, followed by descriptions of distributed systems and cloud computing in Section 3, the design of the general network to implement topologies like a mesh, hypercube, and omega networks in Section 4, the results and discussions in Section 5, and the conclusions in Section 6.

2. PARALLEL PROCESSING

Parallel processing is increasingly being viewed as the sole practical approach for the quick resolution of computationally challenging and data-intensive issues. This creates the conditions for significant parallel software growth [6]. There are three types of parallel processing client-side object rendering, client-side and server-side operations [7].

However, under the parallel programming technique, programs are created explicitly to take use of parallelism. As a result, creating a parallel application requires splitting up the workload into tasks and assigning those tasks to workers (i.e., the computers where the tasks will be accomplished.) [8-10]. Additionally, there is a wealth of literature on designing parallel programs and the applicability of algorithms for parallel execution [10-16].

In order to accomplish the processing, processing units are connected via an interconnection network and the required software. For large-scale computer systems, interconnection networks enable efficient communication between components like processors and memory units [17].

Due to their scalability and adaptability, multistage interconnection networks (MINs) have been extensively researched for distributed systems, parallel, communication networks, optical networks, and contemporary embedded systems as they are dynamically configured and scalable. MINs are used to connect numerous processors or processor-memory systems. Dynamic interconnection networks, in particular, created parallel machines' efficient and flexible communication capacity. The previous few decades have seen a variety of MIN topologies proposed. Either multipath or unique path networks make up the majority of these topologies. MINs try to lower costs and reduce diameter, where diameter is the distance between any two nodes [18-23]. Most well-liked MIN architecture is called SEN (Shuffle Exchange Network). It is available in many different classes, such as cube, baseline, and mesh networks. The self-routing network (SEN) in Figure 1 is based on the shuffle and exchange routing functions.

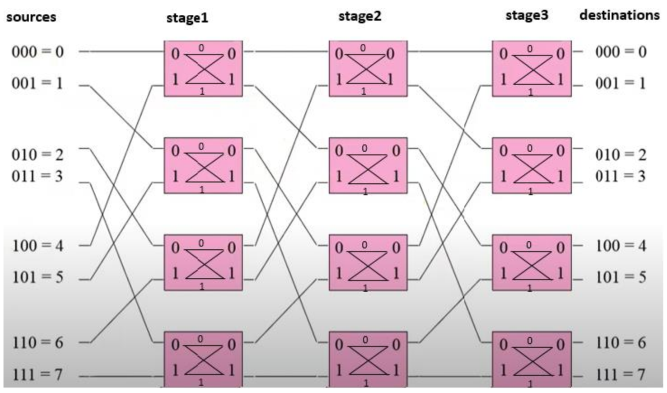
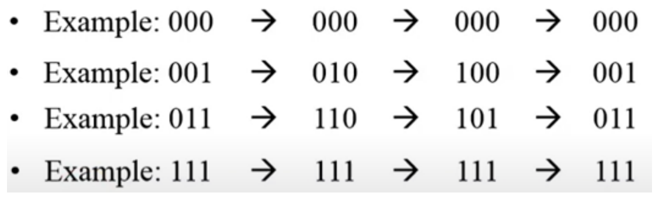


Figure 1. 8*8 SEN topology

The network has self-routing property. It works for each possible input. The logical rotation left of the bits used in the port IDs determines how the stages are connected to one another. Figure 2 shows this operation.



- Example: 000 → 000 → 000 → 000
- Example: 001 → 010 → 100 → 001
- Example: 011 → 110 → 101 → 011
- Example: 111 → 111 → 111 → 111

Figure 2. The logical rotation left of the bits used in the port Ids (If bit is “0” then send cell out upper port, if bit is “1” then send cell out lower port)

Bit0 indicates connecting to the higher output (upper port) for each entry in each switch, while bit1 indicates connecting to the lower output (lower port) as shown in Figure 1. This multi-stage architecture can be used in supercomputer systems that widely use parallel processing. Different network topologies such as hypercube, mesh, and linear topologies are implemented in multi-stage interconnection networks as shown in Figure 3, to implement the connection between any two nodes, two key classes from the java.net package used for creation of a TCP stream are ServerSocket class and Socket class [24].

1. ServerSocket class

The java.net.ServerSocket class is used by the server to create a socket at the server port and listing for the connection request from the clients, this class contain many methods, one of its method is accept method, which gets a connection request from the queue or if there is no connection request (the queue is empty), this method blocks until one connection request arrives [25]. The result of executing accept method is an instance of Socket used for communicating with the client.

2. Socket class

The java.net.Socket class is used by the client and server, the client uses the constructor of the socket class to create a socket specifying the hostname (IP address) and port number of the server, this constructor not only creates a socket, but also connects the socket to the remote server [26].

getInputStream() and getOutputStream() are used for returns the socket input stream and returns the socket output stream respectively.

Each node in each network topology is represented in a specific thread and can be a dynamic selection of topology according to the problem that wants to be solved. So, the proposed design solved the problem of blocking in MINs. The use of a distributed system with at least eight input nodes and eight output nodes to construct a basic SaaS (Software as a Service) cloud computing system. Problems that were resolved using parallel systems are now solved with this cloud.

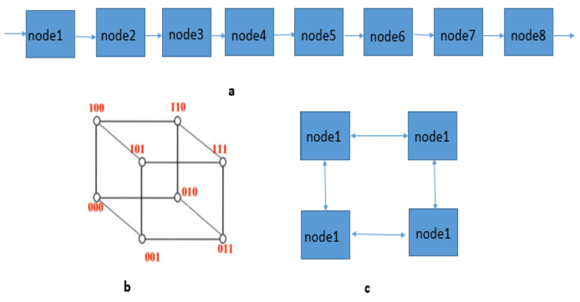


Figure 3. Networks that implement in the proposed design. (a) linear topology. (b) hypercube topology. (c) mesh topology

3. THE DISTRIBUTED SYSTEM AND CLOUD COMPUTING

A distributed system is made up of a number of networked devices (computers, hardware, or software) that communicate their actions through message passing [27]. Due to resource sharing, high availability, parallel processing, and other factors, heterogeneous, and distributed systems have gained popularity in modern computer applications with the advancement of computer network technology [28-32].

There are different types of distributed operating systems that concentrate on the overall system and the distribution of distributed system components over multiple machines. Systems that are client-server, three-tier, and N-tier, as well as middleware. Client-server architecture and peer-to-peer architecture are the two main system-level architectures that matter today. The server will react appropriately to any requests made by the client. The remote side is typically handled by a single server, however, using many servers ensures complete safety. Peer-to-peer networks, sometimes referred to as P2P networks rely on the premise that there is no centralized control in a distributed system. Once a node joins the network, it can act either as a server or a client at any given time.

A lot of computing devices are linked together over a real-time communication network (internet) in the networking technology known as "cloud computing". It refers to the delivery of computing resources over the internet. If we want to store some pictures, or solve some computational problem online instead of using our own computer, we are using cloud computing services. In another word, cloud computing allows individual users to use the cloud services which is managed by third parity at remote location [33, 34]. It is broadly divided into IaaS (infrastructure as a service), PaaS (Platform as a Service), and SaaS (Software as a Service). This technology is believed to be cost-saving and offers resource optimization to a larger extent. In this paper, a SaaS cloud computing system for parallel processing network topology was designed. The nodes in the distributed system are connected to each other according to the network topology to be created. In this work, a control program has been created through which it is controlled how the nodes are linked after determining the type of topology to be created for the sequence of nodes in the network.

4. DESIGN DYNAMIC NETWORK AND METHODOLOGY

Using socket programming in the Java eclipse and TCP datagrams, the Dynamic Network for Parallel Processing was created. The controller, clients, and servers' programs were displayed, put to the test, and evaluated.

Clients and servers use a socket, which serves as both an interface for the application and the network to connect and establish connections. In addition, it is among the greatest distributed computing techniques for enhancing system performance [35-39]. Java socket programming uses BSD (Berkeley Software Distribution) style sockets, which offer features for inter-process communication when interacting with TCP/IP services. It was designed as an Application Programming Interface (API) for inter-process virtual communication that used Internet connections [40-43]. Java offers two classes supporting network connections, Socket, and Server Socket.

In suggestion system, each node has its thread for each application. For example, in this design, batting and covering threads are used. By using interconnection, we can create a topology of a parallel processing network without blocking so that we can improve the performance of the omega network bypassing the blocking problem.

MPI (Message Passing Interface) is used to express tasks for parallel processing. A multistage network is used to distribute tasks to network nodes so they can be carried out. The multicast method's distribution latency cost is O(log(N)), and N refers to the total number of target nodes. for job processing. A number of stages n can be calculated as $n=\log_2 N$, each stage consisting of N/2 switching elements.

In the proposed methodology, the application was developed in Java utilizing socket programming Eclipse IDE (Integrated Development Environment) provided by TCP/IP datagram. The client-server is used to design a general interconnection network which is used to connect different types of topologies and can select the type of topology dynamically according to the application that wants to implement it. The connection between nodes is changed according to the type of topology selected from the GUI main frame. Swing package of java GUI (Graphical User Interface) is used, it was used to create a desktop application that can run on any platform and on any system because it is a component of the Java framework. The proposed system and the application both made use of classes, interfaces, exceptions, InetAddress, IOException, object input and output streams, and Util.Scanner. Figure 4 represents the implementation of the general network (SEN) for parallel processing on a distributed system. In the controller class which has GUI frame after selecting one of the topologies (systolic array, mesh, or hypercube), to transmit and receive data to other system nodes, the nodes open input and output streams on TCP Sockets that they build. After entering the nodes ID of the network that wants to be created, these IDs are converted to binary numbers and implement XOR operation between source and destination then, to determine the path of connection through switches in the network. This is implemented by using these equations [6]:

$$h=2k \qquad \text{if} \qquad 0 =< k =< (p/2)-1 \tag{1}$$

$$2k+1-p \qquad (p/2) =< k =< p-1 \tag{2}$$

P: no. of inputs or outputs nodes=8 as an example
K: input node ID
h: output node ID

This network is made up of log p stages, where p represents the number of inputs (processing nodes) as well as the number of outputs. Each stage of the omega network is made up of an interconnection pattern that connects p inputs and p outputs; if Eqns. (1) and (2) are valid, a link exists between input k and output h. These equations represent the operation of left-rotating the binary representation of k to get h. This pattern of connections is known as a perfect shuffle. It was used to build the connection of switches in Omega Network as software. For example, if the network has eight nodes (p=8), the first four inputs (k=0 to 3) connect to the first input of each switch in network according to the Eq. (1) and the second four inputs (k=4 to 7) connect to the second input of each switch in the network according to the Eq. (2). These equations are implemented on any number of nodes.

Thread in the program is used and represented as tj, j from 0 to 7, and each number represents node ID for servers.

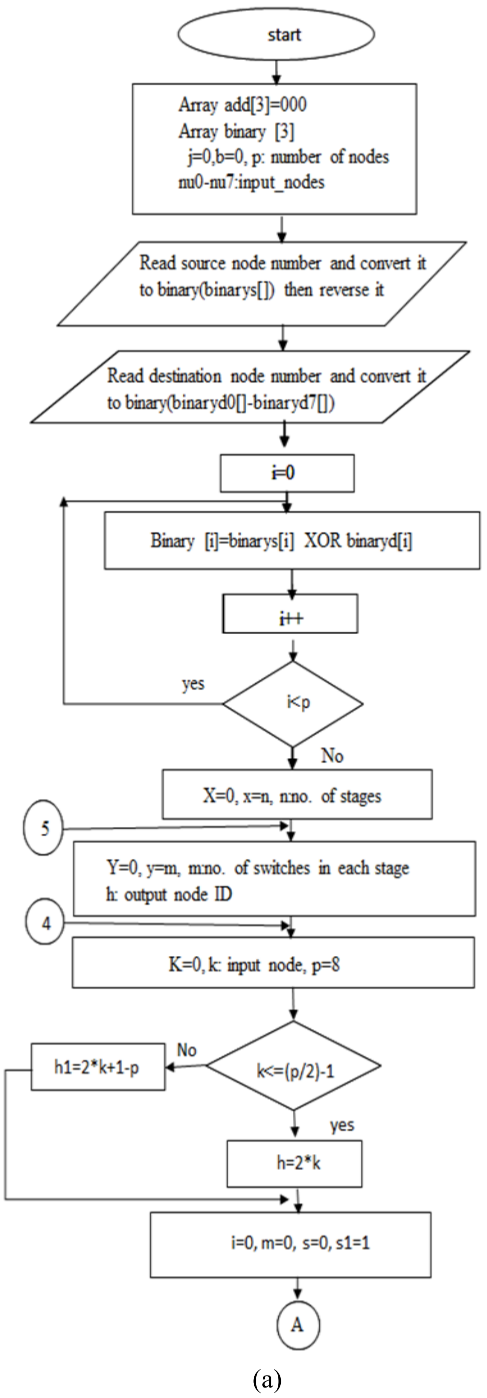
To deliver a high-quality real-time overview, a GUI application built on the Java swing framework is developed. One of the greatest and most reliable frameworks for developing GUI applications is the swing set. The GUI app builder in the Eclipse IDE was utilized for the design. We utilized the Java class JFrame, a container type that derives from the java.awt.Frame class, for the main frame. It performs similarly to the primary window's labels and buttons, and text fields are added to make a GUI. JFrame is in the javax.swing.* library and its constructor class function is used to build an object using the controller program:

JFrame frame=new JFrame("Omega Network 8x8 (Supercomputing)....TCP Socket programing ");

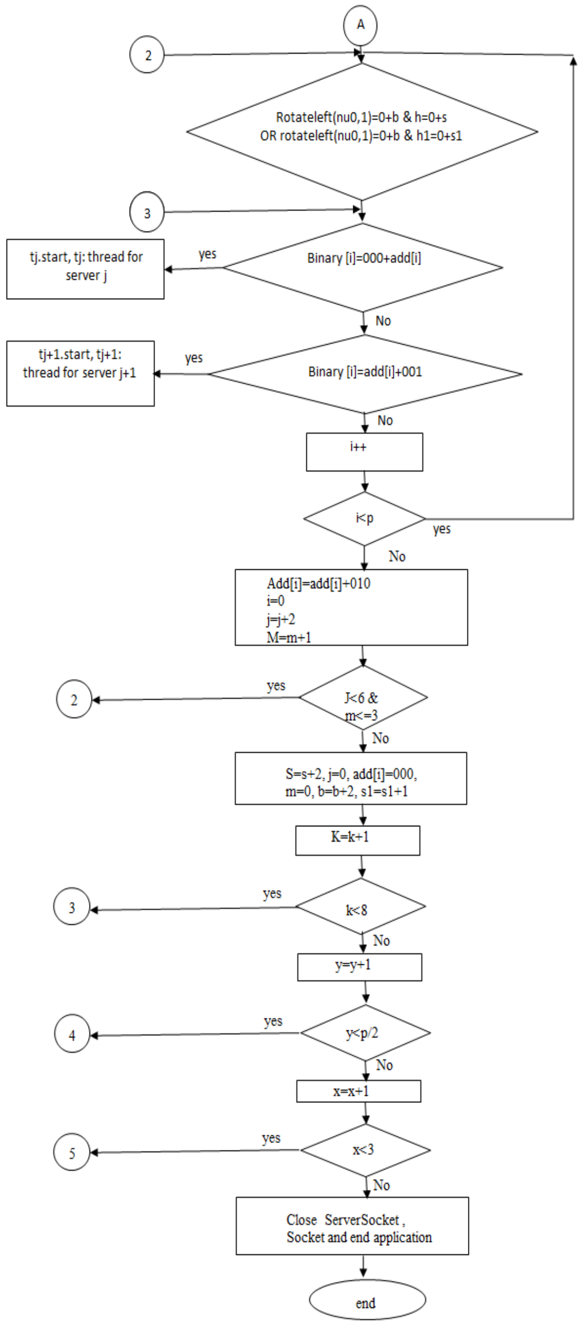
Figure 5 is an example that shows the main interface of the proposed system that has three buttons named MI Multiplication to design mesh topology of four nodes and implement matrix multiplication as an application, LT Convolution Operation to design linear topology of eight nodes, and implement convolution operation as an application, and HC Multiplication operation to design hypercube eight nodes and implement multiplication operation as an application. When clicking on each button new frame is displayed. This is implemented by using the following code for each button:

nameButton.addActionListener(new ActionListener());

Anytime you click the button, the Java ActionListener receives a notification. It is informed of an ActionEvent with a warning. The ActionListener interface is part of the java.awt.event package.


start
Array add[3]=000
Array binary [3]
j=0,b=0, p: number of nodes
nu0-nu7:input_nodes
Read source node number and convert it to binary(binarys[]) then reverse it
Read destination node number and convert it to binary(binaryd0[]-binaryd7[])
i=0
Binary [i]=binarys[i] XOR binaryd[i]
i++
yes
i<p
No
X=0, x=n, n:no. of stages
5
Y=0, y=m, m:no. of switches in each stage
h: output node ID
4
K=0, k: input node, p=8
No
h1=2*k+1-p
k<=(p/2)-1
yes
h=2*k
i=0, m=0, s=0, s1=1
A

(a)



(b)

Figure 4. Flowchart of the proposed design for the controller program

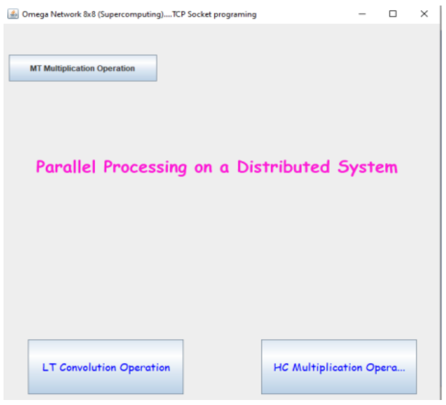

Omega Network 8x8 (Supercomputing)....TCP Socket programing
MT Multiplication Operation
Parallel Processing on a Distributed System
LT Convolution Operation
HC Multiplication Opera...

Figure 5. Flowchart of the proposed design for the controller program

The java.net package implemented two different types of abstraction for network-related programming classes.

(1) Low-level APIs: sockets for bi-directional communication, network interfaces, network identifiers like IP addresses, etc.

(2) Universal Resource Identifiers and Universal Resource Locators are two examples of high-level APIs. Connections.

With the following code, the client Java program creates a communications socket:

Socket s = new Socket (host, port);

The host is a String that contains the server's IP address. The socket constructor throws an exception if it can't locate the host. Two streams can be connected to the socket once it has established a connection with the server for reading (InputStream) and writing (OutputStream) operations. In the proposed system, we created eight clients and named tc0-tc7. All of them started concurrency by using (client name.start) code and waiting for servers to start depending on the network type and application to be implemented. As an example:

tc0.start

The port number needed to build a server socket is given as follows (this may also throw an Exception):

ServerSocket ss = new ServerSocket(port);

Socket s = ss.accept();

The accept() method will remain around until the first client connects to the server.

5. RESULTS AND DISCUSSION

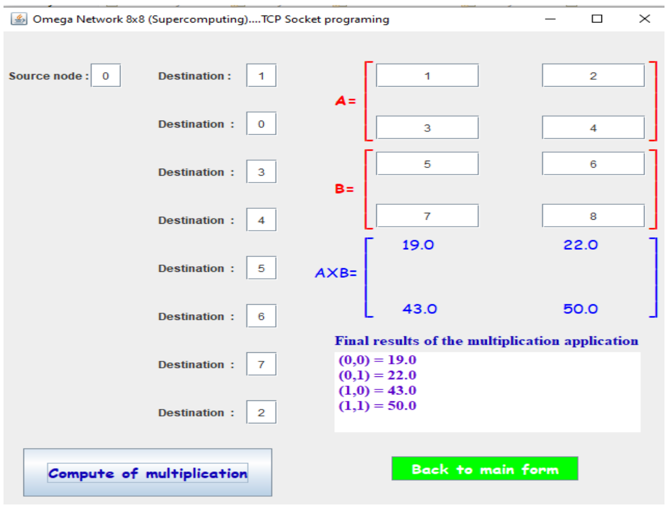
When clicking on the "HC Multiplication Operation "button, the following jpanal in Figure 6 is displayed which is used to create a hypercube topology with 8 nodes and implement matrix multiplication after inserting two matrices (A and B) It has one source that you can select any node that you want to connect to the network through insert any node ID from 0 to 7 (insert source node ID and destination nodes IDs where data to be sent to them). Initially, the source node has fed by the problem (matrix multiplication) that to be solved. The problem is then divided into parts and sent to the destination nodes (according to the sequence of nodes in the network topology) to be able to solve this problem in a parallel processing mode. The outcome is then returned to the controller class which sums the collected results and displays it on the main screen and console workspace of the java program. As an example, assuming multiplying two matrices of dimension 2*2, A and B.

$$A= [1\ 2\ 3\ 4] \qquad (3)$$

$$B= [5\ 6\ 7\ 8] \qquad (4)$$

The multiplication operation A*B is performed after spreading this operation to the destination nodes in parallel processing. The final result is displayed in a text area in Figure 6 and in Figure 7 which represents the console window of the java program.

When selecting "Back to main form" button in Figure 6, the main window is displayed and can implement another network topology with a different sequence. Figure 8 is displayed when selecting "LT convolution operation" button in the main frame which creates a systolic array network topology, after entering the node sequence (IDs) that are required to connect the desired topology. For example, two matrixes H= [2.3, 4.5, 6, 12.3] and X= [2, 4, 8.5, 11.3] can be controlled using four nodes. When selecting "compute convolution", the results are as shown in the text area of Figure 8.

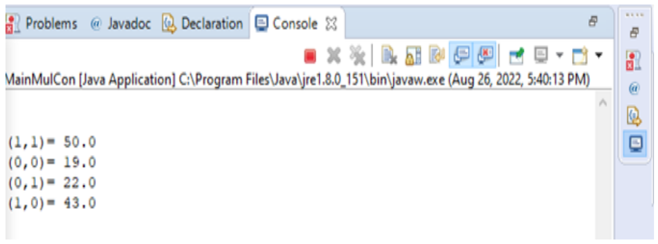

Omega Network 8x8 (Supercomputing)....TCP Socket programing
Compute of multiplication
Back to main form

Source node : 0
Destination : 1
Destination : 0
Destination : 3
Destination : 4
Destination : 5
Destination : 6
Destination : 7
Destination : 2

A= [1 2; 3 4]
B= [5 6; 7 8]
AXB= [19.0 22.0; 43.0 50.0]

Final results of the multiplication application
(0,0) = 19.0
(0,1) = 22.0
(1,0) = 43.0
(1,1) = 50.0

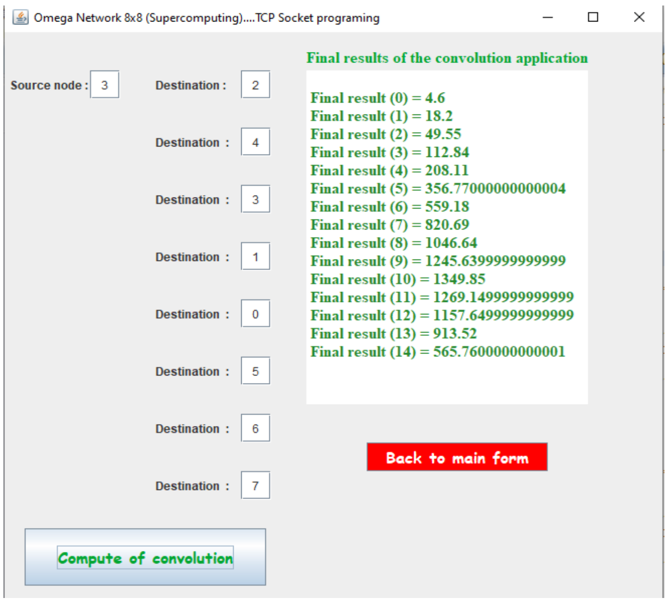
Figure 6. Matrix multiplication A*B



```
MainMulCon [Java Application] C:\Program Files\Java\jre1.8.0_151\bin\javaw.exe (Aug 26, 2022, 5:40:13 PM)

(1,1)= 50.0
(0,0)= 19.0
(0,1)= 22.0
(1,0)= 43.0
```

Figure 7. Results of multiplication on the console window

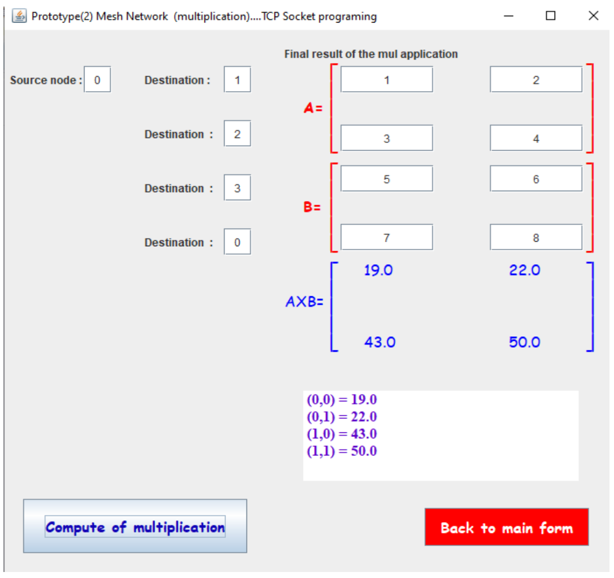


Source node : 3
Destination : 2
Destination : 4
Destination : 3
Destination : 1
Destination : 0
Destination : 5
Destination : 6
Destination : 7

Final results of the convolution application
Final result (0) = 4.6
Final result (1) = 18.2
Final result (2) = 49.55
Final result (3) = 112.84
Final result (4) = 208.11
Final result (5) = 356.77000000000004
Final result (6) = 559.18
Final result (7) = 820.69
Final result (8) = 1046.64
Final result (9) = 1245.6399999999999
Final result (10) = 1349.85
Final result (11) = 1269.1499999999999
Final result (12) = 1157.6499999999999
Final result (13) = 913.52
Final result (14) = 565.7600000000001

Figure 8. Linear topology used to implement convolution operation

Figure 9 is displayed when clicking on the "MT multiplication operation" button from the main frame which implements mesh topology with four nodes as an example. After selecting the sequence of nodes that formulate a mesh topology, data of the elements of matrixes (A and B) should be entered. Figure 10 shows the results of this type of multiplication on the console window of java.

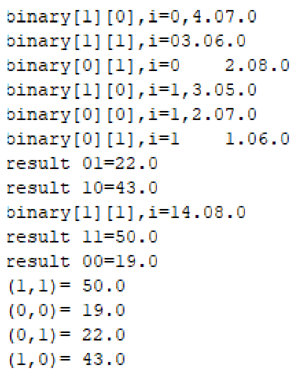


```
Prototype(2) Mesh Network (multiplication)....TCP Socket programing

Source node : 0    Destination : 1        Final result of the mul application
                   Destination : 2        A=  [ 1   2 ]
                   Destination : 3            [ 3   4 ]
                   Destination : 0        B=  [ 5   6 ]
                                              [ 7   8 ]
                                          AXB= [ 19.0   22.0 ]
                                               [ 43.0   50.0 ]

(0,0) = 19.0
(0,1) = 22.0
(1,0) = 43.0
(1,1) = 50.0
```

Figure 9. Mesh topology with 4 nodes



```
binary[1][0],i=0,4.07.0
binary[1][1],i=03.06.0
binary[0][1],i=0    2.08.0
binary[1][0],i=1,3.05.0
binary[0][0],i=1,2.07.0
binary[0][1],i=1    1.06.0
result 01=22.0
result 10=43.0
binary[1][1],i=14.08.0
result 11=50.0
result 00=19.0
(1,1)= 50.0
(0,0)= 19.0
(0,1)= 22.0
(1,0)= 43.0
```

Figure 10. Results of multiplication on the console window of java

6. CONCLUSIONS

This work produced a complete system with all its algorithms for parallel processing operations using distributed system approach. The necessary interconnection architecture of the parallel nodes is formed on a distributed system through the implementation of parallel processing. The designed system was tested on eight nodes depending on the principle client/server and used to solve many computational problems based on various topologies. Parallel programs that need a large number of processors can make benefit from the distributed system by implementing the required topology of processors to execute the program. So, this paper introduces some of these topologies. It can be inferred that an actual system has been addressed to provide the capability of conducting large amounts of processing in parallel with the ability to select the type of network topology, depending on the type of application. As well as the ability to use processing power across the cloud domain based on distributed system concepts. This research paper provides an overview as well as a thorough implementation of a general interconnection network that is suitable to implement different network topologies such as linear systolic topology, mesh, and hypercube topologies, and uses those topologies in parallel processing.

ACKNOWLEDGMENT

The authors would like to express their gratitude to the faculty and staff of the University of Mosul-College of Engineering, especially the departments of computer engineering, for their support to complete this work.

REFERENCES

[1] Xiao, F., Zhan, C., Lai, H., Tao, L., Qu, Z. (2017). New parallel processing strategies in complex event processing systems with data streams. International Journal of Distributed Sensor Networks, 13(8): 1550147717728626. https://doi.dox.org/10.1177/1550147717728626

[2] Chaubey, M.S., Oluwaseun, A. (2019). Parallel and distributed computing. IJRAR, 6(1).

[3] Le, M., Clyde, S., Kwon, Y.W. (2019). Enabling multi-hop remote method invocation in device-to-device networks. Human-centric Computing and Information Sciences, 9(1): 1-22. https://doi.org/10.1186/s13673-019-0182-9

[4] Kang, H., Jeong, K., Lee, K., Park, S., Kim, Y. (2016). Android RMI: a user-level remote method invocation mechanism between Android devices. The Journal of Supercomputing, 72: 2471-2487. https://doi.org/10.1007/s11227-015-1471-3

[5] Madhavi, D. (2016). Transparency in remote method invocation (RMI) for distributed systems: Middleware layer. International Journal of Emerging Technologies in Engineering Research (IJETER), 4(1): 44-49.

[6] Grama, A., Gupta, A., Karypis, G., Kumar, V. (2003). Introduction to Parallel Computing. Second Edition. Addison-Wesley.

[7] Frachtenberg. E., Schwiegelshohn, U. (2007). Job Scheduling Strategies for Parallel Processing. Springer Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-71035-6.

[8] Sottile, M.J., Mattson, T.G., Rasmussen, C.E. (2010). Introduction to Concurrency in Programming Languages. CRC Press. https://doi.org/10.1201/b17174

[9] Andrews, G.R. (1999). Foundations of Multithreaded, Parallel, and Distributed Programming. Addison Wesley.

[10] Mattson, T.G., Sanders, B.A., Massingill, B. (2005). Patterns for Parallel Programming. Addison-Wesley Professional.

[11] Kumar, V., Grama, A., Gupta, A., Karypis, G. (1994). Introduction to Parallel Computing: Design and Analysis of Algorithms. Benjamin/Cummings Publishing Company.

[12] Fox, G., Johnson, M., Lyzenga, G., Otto, S., Salmon, J., Walker, D. (1988). Solving Problems on Concurrent Processors, vol. 1. Prentice Hall, Englewood Cliffs. https://doi.org/10.1063/1.4822815

[13] Quinn, M.J. (1994). Parallel Computing: Theory and Practice. McGraw-Hill, New York, NY.

[14] Hansen, P.B. (1995). Studies in Computational Science: Parallel Programming Paradigms. Prentice-Hall, Englewood Cliffs, NJ.

[15] Chandy, K.M., Misra, J. (1988). Parallel Program Design: A Foundation. Addison-Wesley, Reading, MA.

[16] Doroshenko, A.Y., Ovdii, O.M., Yatsenko, O.A. (2017). Ontological and algebra-algorithmic tools for automated

design of parallel programs for cloud platforms. Cybernetics and Systems Analysis, 53: 323-332. https://doi.org/10.1007/s10559-017-9932-8

[17] Amodu, O.A., Othman, M., Yunus, N.A.M., Hanapi, Z.M. (2021). A primer on design aspects and recent advances in shuffle exchange multistage interconnection networks. Symmetry, 13(3): 378. https://doi.org/10.3390/sym13030378

[18] Chen, C.W., Lu, N.P., Chen, T.F., Chung, C.P. (2000). Fault-tolerant gamma interconnection networks by chaining. IEE Proceedings-Computers and Digital Techniques, 147(2): 75-81. https://doi.org/10.1049/ip-cdt:20000185

[19] Rastogi, R., Chauhan, D.S., Govil, M.C. (2012). Disjoint paths multi-stage interconnection networks stability problem. arXiv preprint arXiv:1202.0612. https://doi.org/10.48550/arXiv.1202.0612

[20] Chen, C.W., Lu, N.P., Chung, C.P. (2003). 3-Disjoint gamma interconnection networks. Journal of Systems and Software, 66(2): 129-134. https://doi.org/10.1016/S0164-1212(02)00070-5

[21] Borkar, M.A., Nitin (2011). 3D-CGIN: A 3 disjoint paths CGIN with alternate source. In: Abraham, A., Mauri, J.L., Buford, J.F., Suzuki, J., Thampi, S.M. (eds) Advances in Computing and Communications. ACC 2011. Communications in Computer and Information Science, vol 193. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-22726-4_4

[22] Rajkumar, S., Goyal, N.K. (2014). Design of 4-disjoint gamma interconnection network layouts and reliability analysis of gamma interconnection networks. The Journal of Supercomputing, 69: 468-491. https://doi.org/10.1007/s11227-014-1175-0

[23] Gunawan, I., Fard, N.S. (2012). Terminal reliability assessment of gamma and extra-stage gamma networks. International Journal of Quality & Reliability Management, 29(7): 820-831. https://doi.org/10.1108/02656711211258553

[24] Buyya, R., Selvi, S.T., Chu, X. (2009). Object-Oriented Programming with Java: Essentials and Applications. Tata McGraw-Hill.

[25] Reilly, D., Reilly, M. (2002). Java Network Programming and Distributed Computing. Addison-Wesley Professional.

[26] Garg, V.K. (2005). Concurrent and Distributed Computing in Java. John Wiley & Sons.

[27] Coulouris, G.F. (2012). Distributed Systems: Concepts and Design. Boston, Addison-Wesley.

[28] Shukur, H., Zeebaree, S., Zebari, R., Ahmed, O., Haji, L., Abdulqader, D. (2020). Cache coherence protocols in distributed systems. Journal of Applied Science and Technology Trends, 1(3): 92-97. https://doi.org/10.38094/jastt1329

[29] Haji, L.M., Zeebaree, S., Ahmed, O.M., Sallow, A.B., Jacksi, K., Zeabri, R.R. (2020). Dynamic resource allocation for distributed systems and cloud computing. TEST Engineering & Management, 83(May/June 2020): 22417-22426.

[30] Shukur, H., Zeebaree, S., Zebari, R., Zeebaree, D., Ahmed, O., Salih, A. (2020). Cloud computing virtualization of resources allocation for distributed systems. Journal of Applied Science and Technology Trends, 1(3): 98-105. https://doi.org/10.38094/jastt1331

[31] Dino, H.I., Zeebaree, S.R., Ahmad, O.M., Shukur, H.M., Zebari, R.R., Haji, L.M. (2020). Impact of load sharing on performance of distributed systems computations. International Journal of Multidisciplinary Research and Publications (IJMRAP), 3(1): 30-37.

[32] Zeebaree, S.R., Shukur, H.M., Haji, L.M., Zebari, R.R., Jacksi, K., Abas, S.M. (2020). Characteristics and analysis of hadoop distributed systems. Technology Reports of Kansai University, 62(4): 1555-1564.

[33] Rashid, A., Chaturvedi, A. (2019). Cloud computing characteristics and services: A brief review. International Journal of Computer Sciences and Engineering, 7(2): 421-426. https://doi.org/10.26438/ijcse/v7i2.421426

[34] Patidar, S., Rane, D., Jain, P. (2012). A survey paper on cloud computing. In 2012 Second International Conference on Advanced Computing & Communication Technologies, pp. 394-398. https://doi.org/10.1109/ACCT.2012.15

[35] Zebari, D., Haron, H., Zeebaree, S. (2017). Security issues in DNA based on data hiding: A review. International Journal of Applied Engineering Research, 12(24): 0973-4562.

[36] Zeebaree, S., Zebari, I. (2014). Multilevel client/server peer-to-peer video broadcasting system. International Journal of Scientific & Engineering Research, 5(8): 260-265.

[37] Saleem, S.I., Zeebaree, S., Zeebaree, D.Q., Abdulazeez, A.M. (2020). Building smart cities applications based on IoT technologies: A review. Technology Reports of Kansai University, 62(3): 1083-1092.

[38] Zebari, I.M., Zeebaree, S.R., Yasin, H.M. (2019). Real time video streaming from multi-source using client-server for video distribution. In 2019 4th Scientific International Conference Najaf (SICN), Al-Najef, Iraq, pp. 109-114. https://doi.org/10.1109/SICN47020.2019.9019347

[39] Selamat, S.A.A.Z.A. (2017). Electronic learning management system based on semantic web technology: A review. International Journal of Advances in Electronics and Computer Science, 4(3): 1-6.

[40] Sawant, A.A., Meshram, B. (2013). Network programming in Java using Socket. Google Scholar.

[41] Kalita, L. (2014). Socket programming. International Journal of Computer Science and Information Technologies, 5(3): 4802-4807.

[42] Maata, R.L.R., Cordova, R., Sudramurthy, B., Halibas, A. (2017). Design and implementation of client-server based application using socket programming in a distributed computing environment. In 2017 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC), pp. 1-4. https://doi.org/10.1109/ICCIC.2017.8524573

[43] Godla, S.R., Fikadu, G., Adema, A. (2022). Socket programming-based RMI application for Amazon web services in distributed cloud computing. In: Raj, J.S., Kamel, K., Lafata, P. (eds) Innovative Data Communication Technologies and Application. Lecture Notes on Data Engineering and Communications Technologies, vol 96. Springer, Singapore. https://doi.org/10.1007/978-981-16-7167-8_37