ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 6, Issue 2, August 2016

Parallel Processing Implementations on Distributed System

Basil Shukr Mahmood, Manar Husamaldeen Hammadi

Abstract*— This paper discusses the implementation of linear systolic array (of SIMD processor) topology on a distributed system consisting of four nodes (computers). Nodes were connected together by a linear systolic array topology. Socket programing is used to implement this topology. Two types of applications are selected, convolution operation and parallel sorting. A comparison between the execution time of some applications (parallel sorting and convolution problems) on a distributed system and the execution time of the same applications that on a centralized single computer. The results show that a speed-up can be achieved specially when big data is considered, and that was obvious for the bubble sort problem. On the contrary, enhancement in the execution time for linear array convolution couldn't be achieved for small number of nodes. However, the main aim of this work is to implement a linear systolic topology on a distributed system, so that programs on SIMD machine can now be executed on a distributed system.

***Index Terms*— Distributed system, SIMD, Middleware, RPC, RMI, Socket, Linear systolic array topology.**

I. INTRODUCTION

From 1945 until around 1985, computers were large and very expensive [1]. Minicomputer costs tens of thousands of dollars, mainframe costs millions of dollars. Starting around the 1985 two important developments had been appeared. The first was the microprocessors, and the second was the invention of computer networks [1]. The result of these developments led to the emergence of the distributed system. The distributed system is a collection of networked nodes (computers) that communicate and coordinate its action by using message passing [2]. The distributed system has three main types: distributed computing system, distributed information system and distributed embedded system [1]. We deal with a distributed computing system (which uses the distributed system to solve some computational problems). Each node (which represents SIMD machine) in the distributed system are connected to another by using logical network topology such as linear systolic array topology. The network topology or connection between nodes is implemented by using distributed system software (Middleware) (such as Remote Procedure Call (RPC), Remote Method Invocation (RMI), and Socket. Middleware defined as a layer of software below the application program but above the operating system that provides a common programming abstraction across a distributed system [3].

This paper is organized as follows: Section II provides a short background about the communication in distributed system; Section III describes the synchronization in distributed system; Section IV presents the design of the convolution and sorting operations on a linear array topology; Section V presents the results and discussion; and Section VI provides conclusion and future work.

II. COMMUNICATIONS IN DISTRIBUTED SYSTEM

Communications in distributed system actually occurs between processes (which is an instance of a program running in a computer [11]). These processes can communicate with each other by using distributed system software such as RPC, RMI, Socket.

A. RPC and RMI

RPC: The Remote Procedure Call (RPC) is a protocol that one program in the client side can use it to request a service from a program running in the server side (another computer in a network) without having to understand the details of the network, in another word, in RPC a procedure in the server side (remote machines) can be called as if they are procedures in the local address space [4]. RMI: It is a Java mechanism of RPC or object-oriented equivalent of RPC, allows an object running in one Java virtual machine (JVM) to invoke methods inside an object running in another Java virtual machine [5].

B. Socket

Socket defined as an interface between the application layer and network layer. Sockets originate from BSD UNIX but are also present in most operating systems such as windows and Linux [2]. Inter-process communication consists of transmitting a message between socket in one process and socket in another process [2]. If one process wants to receive message using socket, its socket must be bound to a local port number and the IP address of the computer on which it runs.

a) Function call for using socket

Sockets are created and used with a set of function calls sometimes called the socket API, the system calls of socket API are divided into three categories [6]:

i. Managing connection:

- Socket (): This function creates an endpoint for communication and returns a file descriptor that refers to that endpoint.
- Bind (): This function binds the socket with IP address and port number.
- Listen (): This function allow the server to listen on the socket for the incoming connection requests. Connect (): This function is used by TCP client to initiate a connection with a TCP server.
- Accept (): The accept function is called by the server process to accept the connection. If no client is trying to connect the call will block.

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 6, Issue 2, August 2016

ii. Sending and receiving data:
- Send ()/recv(): These functions are used to send/receive data over stream sockets or CONNECTED datagram sockets.
- Sendto ()/recfrom():These functions are used to send/receive data over UNCONNECTED datagram sockets.

iii. Managing endpoint characteristics:
- Close (): This function is used to close the connection between client and server.

The client process uses the followings system calls: socket(), connect(), send()/sendto(), recv()/recvfrom(), close. While the server process uses the followings system calls: socket(), bind(), listen(), accept(), send()/sendto(), recv()/recvfrom(), close().

b) Types of socket

The two most common types of socket are:

i. SOCK_STREAM (TCP socket): Stream sockets allow processes to communicate using TCP protocol. A stream socket provides reliable, sequenced, bidirectional, and unduplicated flow of data with no record boundaries [7]. To establish a TCP socket on the client side we must do the followings: create socket using the socket () function, connect the socket to the address (IP and port) of the server using the connect () function, Send and receive data using read () and write () functions, close the connection using close () function [8]. To establish a TCP socket on the server side we must do the followings: create a socket with the socket() function, bind the socket to an address using the bind() function, listen for connections with the listen() function, accept connection with the accept() function, send and receive data by using send() and receive() functions, close the connection by using close() function[8].

ii. SOCK_DGRAM (UDP socket): Datagram sockets allow processes to communicate using UDP protocol. A UDP socket supports bidirectional flow of messages. Processes that use a datagram socket may receive duplicate messages and may receive messages in a different order from the sending sequence [7]. To establish a UDP socket communication on the client side, we must do the followings: create socket using the socket() function, send and receive data by using sendto() and recvfrom() functions [8]. To establish a UDP socket communication on the server side we must do the followings: create socket with the socket() function, bind the socket to an address using the bind() function, send and receive data by using sendto() and recvfrom() functions[8].

c) Typical client and server socket programs

The steps for creating a typical client socket program are: create socket, determine the address and port number of the server, initiate the connection to the server (in case of TCP), write/read data to/from socket, close the socket [8]. The steps for creating a typical socket server program are: create socket, associate local address and port with the socket, indicate how many clients-in-waiting to permit, accept an incoming connection from client, send/receive data to/from socket, close the socket, repeat with the next connection request [8].

III. DISTRIBUTED SYSTEM SYNCHRONIZATION

Synchronization in the distributed system defined as matching of more than one process in time. Each node in the distributed system maintains its own time from its local clock, and this time may be not the same for other nodes, this causes several problems [2]. The solution for this problems is to synchronize the physical clock of each node with the other nodes by using one of the common clock synchronization methods such as Christian algorithm, Berkeley algorithm or NTP (Network Time Protocol) [2]. In this research, a program called Network Time Synchronization which depends on the NTP server is used. This server enables clients across the internet to be accurately synchronized to a Universal Coordinated Time (UCT) in order to synchronize each node in the system [9].

IV. LINER SYSTOLIC ARRAY TOPOLOGY

In the linear array topology, the processing elements (PEs) are arranged in one dimension and the interconnection between each PEs are the nearest neighbor only as shown in Fig (1).

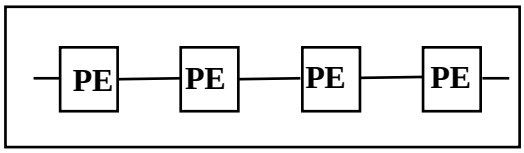

Fig. (1) linear systolic array topology [12]

Different types of applications can be implemented on a linear array topology such as convolution, Discrete Fourier Transform (DFT), sorting, …etc. In this research two types of applications were considered, the first is the convolution and the second is the sorting problem.

A. Convolution on a linear array topology

There are many ways to implement convolution process on a linear array topology, in this research it is assumed the architecture of the required PE is as shown in Fig (2). As shown in Fig (2), hi represents the weight or one of the impulse response tabs. Xj: represents the input signal. S: represents the output signal.

To implement convolution, each node (PEi) must do the followings:

i. Receive hi from the controller node

ii. Receive Xj and S from its neighbor node, and send Xj to the next neighbor node.

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 6, Issue 2, August 2016

iii. Multiply Xj by hi and store the result in its own queue, where the tail of the queue equal to i.
iv. Calculate the new S, S=S + first element in the queue. Then send S to the next neighbor node

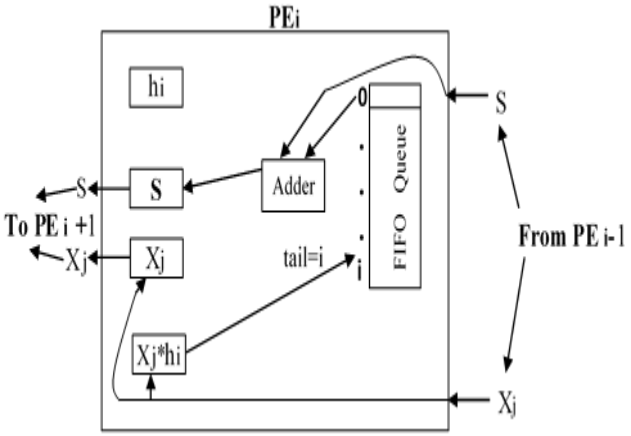

Fig. (2) Structure of each node in the convolution application.

B. Parallel Sorting on a linear array topology

In this research we implement a type of parallel sorting called odd-even merge sort that can sort data of size (D/P), (P: is the number of PEs, D: is any number divisible by p) [10]. Each PE receives data of size (D/P) and performs a local sorting on this data [10]. This method of parallel sorting depends on odd-even exchange concepts, which means, PEs operate in two alternating phases odd and even. In the even phase, PEs with even ID number exchange data with their neighbors, while in the odd phase PEs with odd ID number exchange data with their neighbors [10]. After (D/P) of Odd-Even phases, the data becomes sorted [10]. The following steps occurs to implements this type of sorting:

i. The controller node receives data of size D, divide them into blocks each of size (D/P), then sends each block to the appropriate PE.
ii. Each PE performs a local sorting on its (D/P) block.
iii. In the even phase, PE with even ID number, exchanges its own (D/P) block with the next neighbor PE, as shown in Fig (3).
iv. In the odd phase, PE with odd ID number, exchanges its own (D/P) block with the next neighbor PE, as shown in Fig (3).
v. Each PE merge its (D/P) block with the (D/P) block received from its neighbor PE, and maintains only the appropriate half of the merged block.

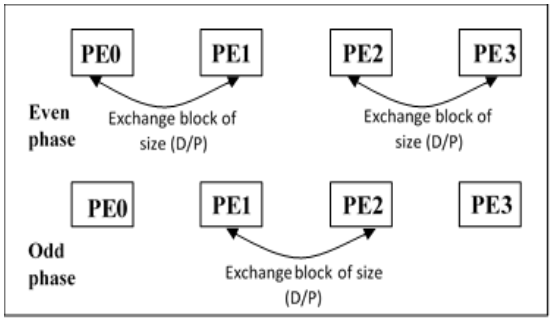

Fig. (3) Odd-Even exchange of data

V. RESULTS AND DISCUSSION

In this research we implemented two types of linear array applications: convolution and sorting over only four nodes by using socket programing. Each node (computer) has a CPU of type core i5 and operating system of type Windows 10. The programing language used is Java and the NetBeans IDE 8.0.2 software is used as a development environment tool.

The execution times of sorting and convolution problems that are implanted on a linear array are compared with the execution times of the same problems (sorting, convolution) that are assumed to be in centralized system (single computer). In parallel processing, it is known that the execution time decreases as the number of processors increases and, in another words, the results that was obtained (as shown in Figs (4,5)) can be improved dramatically as the number of nodes increase. The type of data chosen is random double (size of each element is 8bytes), and the size of data ranged from 100 to 2000000 elements.

Fig (4) represents the execution times of sorting problem implemented on a linear array and on a single computer. The sorting on a linear systolic array needs less execution time than sorting on a single computer. As the size of data increases, the execution time on a single computer increases significantly as compared to the linear array sorting. When the size of data becomes 200000, the execution time becomes almost 25 times of the execution time of linear array sorting. Fig (5) represents the execution times of the convolution problem implemented on linear systolic array and on a single computer. The convolution on a single computer needs less execution time than the execution time of the linear array convolution, this is because the number of nodes (four) used in the design is very small.

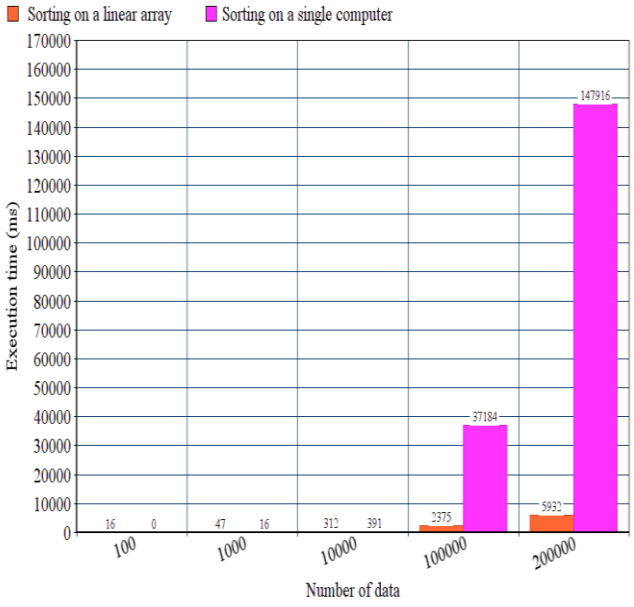

Fig. (4)

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 6, Issue 2, August 2016

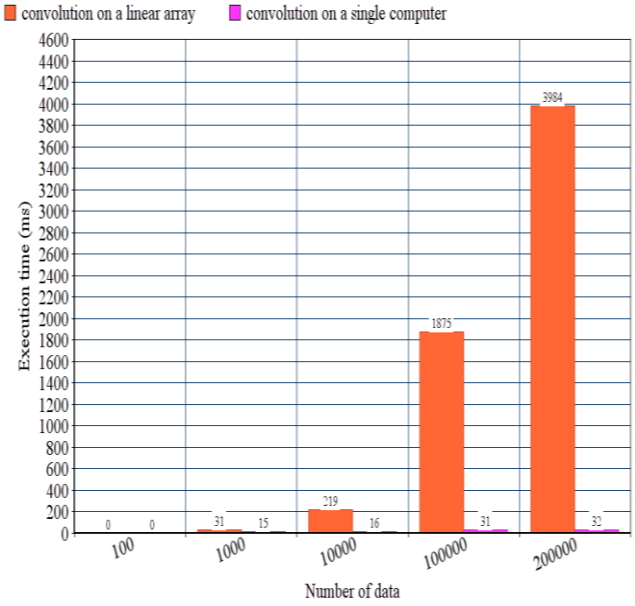

Fig. (5)

VI. CONCLUSION AND FUTURE WORK

Linear systolic topology is implemented on a distributed system so that operations like convolution and sorting are executed just like executing them on a parallel computer system. Their execution times are compared with the execution time of them (convolution and sorting) when they are assumed to be executed on a centralized system (single computer). It is noted that sorting on a linear systolic array needs less execution time than sorting on a single computer in spite of the small number of the used nodes(computers), while the linear systolic array convolution cannot offer speed-up for this number of nodes. However, the main conclusion is that parallel processing can be simply implemented on a distributed system when the required interconnection topology of the parallel machine is simulated on a distributed system.

When the number of used nodes increases, the performance of the system improves (the execution time decreases). In this research, the applications (convolution and sorting) are programmed in such a way, so that, if more nodes are added to the system, the same program applies for those extra nodes without the need of any change. The topology can be implemented in such a way that it can be isolated from the applications; this allows any problem that can be solved on the linear array systolic topology can be simply programmed separately. Also, other types of topologies like mesh, cubic, hyper cubic ...etc. are recommended to be simulated on a distributed system as a future work.

REFERENCES

[1] TANENBAUM, A. S., & STEEN, M. V. (2007).” Distributed systems: principles and paradigms”. Upper Saddle River, NJ, Pearson Prentice Hall.

[2] COULOURIS, G. F. (2012).” Distributed systems: concepts and design”. Boston, Addison-Wesley.

[3] Mangal Sain, HoonJae, Lee,Wan-Young Chung. (15-18 Feb. 2009)’Middleware for ubiquitous Healthcare Information system’, Advanced Communication Technology, 2009. ICACT 2009. 11th International Conference on, 03, pp. 2325 - 2328.

[4] W.R. Stevens. (1999).” UNIX Network Programming, Interprocess Communication”. 2 editions. Volume 2. Upper Saddle River, NJ, Pearson Prentice Hall.

[5] https://docs.oracle.com/javase/tutorial/rmi/.

[6] Brian Hall. (2016). Beej's,” Guide to Network Programming Using Internet Sockets”. n/a.

[7] Vijay K. Garg.(2004) ,”Concurrent and Distributed Computing in Java”. John Wiley & Sons.

[8] Michael J. Donahoo, Kenneth L. Calvert. (2002). “TCP/IP Sockets in C: Practical Guide for Programmers”. 1 edition. Morgan Kaufmann.

[9] D.Adithya Chandra Varma, Praveen Kumar Reddy.M Prof.Gopinath. (Sep-Oct 2013) ‘Performance Comparison of Physical Clock Synchronization Algorithms’, Int. Journal of Engineering Research and Applications (3, 5), pp. 1355-1364.

[10] Henri Casanova, Arnaud Legrand, Yves Robert. (2008)’Parallel Algorithms’. Chapman & Hall/CRC.

[11] ABRAHAM SILBERSCHATZ, PETER BAER GALVIN, GREG GAGNE. (2005). Operating System Concepts. 7 editions. John Wiley & Sons.

[12] Behrooz Parhami. (1999). Introduction to Parallel Processing: Algorithms and Architectures. KLUWER ACADEMIC.