International Journal of Innovation, Creativity and Change. www.ijicc.net
Volume 13, Issue 7, 2020

Pipelined Parallel Processing Implementation based on Distributed Memory Systems

Adnan Mohsin Abdulazeez[a], Lana Latif Nahmatwlla[b], Diyar Qader Zeebaree[c], [a]Presidency of Duhok Polytechnic University, Duhok, Kurdistan Region, Iraq, [b]Computer Science Department University of Salahaddin, Erbil-Kurdistan Iraq, [c]Research Center of Duhok Polytechnic University, Duhok, Kurdistan Region, Iraq, Email: [a]adnan.mohsin@dpu.edu.krd, [c]dqszeebaree@dpu.edu.krd

Complex problems take a long time to solve, with low efficiency and performance. So, to overcome these barriers and problems, studies have headed towards a way to divide the problem into independent parts, to solve and remedy each of the parts separately in a way that each component can implement part of it, with simultaneous problems with others. Parallel processors are computer systems made up of several processing units connected over some interconnection networks and the programs needed to perform the processing. .Parallel processing and pipelining approaches are duals of each other and if there is pipelining of a calculation, it can further be executed in parallel. Both exploit concurrency accessibility in the computation using various methods. In this paper, we implemented pipelined parallel processing on distributed memory by taking advantage of both parallel processing and the pipeline approach. We proposed a parallel application design using client/servers' principles and divided the workload among multiple hosts at the servers' side. This paper is based on merging and sorting case studies and all the algorithms related to this case study are implemented by using Borland C++ Builder language. Experimental results showed much-improved performance and throughput.

Key words: *Parallel Processing, pipeline approach, Merge sorting.*

Introduction

Nowadays, internet communication becomes a major part of infrastructure. Based on the internet most of the applications of infrastructure systems can be operated (Abdulazeez, Zeebaree, & Sadeeq, 2018; D. A. Zebari, Haron, Zeebaree, & Zeebaree, 2018). This is due to the expansion in transmission of data through the Internet and its constrained transfer speed, and the time taken by the data to achieve the goal additionally expanded (D. A. Zebari, Haron, Zeebaree, & Zain, 2019). To speed-up the execution of a program, the program is divided into multiple fragments that can be executed simultaneously, each on its own processor. A program be executed across the processors might execute n times faster than it would using a single processor (S. R. Zebari & Yaseen, 2011).

In the present state of the parallel processing art, the individual processing nodes of a parallel processing network are designed to execute efficiently a single network protocol or perhaps, say, two protocols, but at different times. A protocol might be one that implements a message passing algorithm between the processing nodes, it might be a protocol for sharing a common memory between the individual nodes, or any of a number of other types of protocols. U.S. Pat. No. 4,247,892, issued to P. N. Lawrence, is illustrative of the state of the art of message passing parallel processing networks. Each processor of the network calculates the number of destination nodes a message from the processor must pass through en route to its destination node. Each processor then casts a vote for the most advantageous protocol, from its viewpoint, to use in the next message transmittal phase. A foreman computer tallies the votes and informs the other processors of the winning protocol. Only one type of protocol, however, may be performed in any transmitting phase of the network (DeBenedictis, 1988; Merigot & Petrosino, 2008; Sasikumar, Shikhare, & Prakash, 2014). Parallel Processing (PP) is certainly not a new concept. For decades, performance research has focused on reducing the time it takes to execute floating-point and other operations related to solving numerically intensive algorithms used in such fields as structural mechanics and fluid dynamics. There are three distinct areas of PP: server-side functions, server process client-side functions and client process object rendering (Buluç & Madduri, 2011; Frachtenberg & Schwiegelshohn, 2007; McCalpin, 1995; McDonald & White, 1994). Uses or applications for PP come from two different areas; on the one hand, there are high performance systems for speeding up computer intense calculations. These can be executed on traditional supercomputer systems or on large clusters of workstations. On the other hand, there are embedded control systems on sequential hardware, which requires Parallel Programming concepts to control concurrent external actuators or internal processes. PP is commonplace today under standard operating systems such as Linux and Windows, with parallel software design becoming more and more important (S. R. Zebari & Yaseen, 2011). Scientific applications usually require the processing of large amounts of data and need a long time to be solved. Parallel architectures are a probable result to tackle these issues. Parallel processing is linked to the incorporation of several computers running in parallel and interpreting computationally rigorous difficulties. The core objective of parallel processing is to avail users with execution which no single computer may furnish(Atiqullah & Rao, 2000; Chae & Abraham, 2001; Drozdowski, 2009; Haji, Zeebaree, Jacksi, & Zeebaree, 2018; Junji, 2004). For parallel programming, an activity is split among several threads or processes executing collaterally.

An activity might be processed n times faster if executed via n processors in comparison to using a single processor. Moreover, the same split activities can be executed concurrently on distinct nodes in a high-speed network. On the contrary, the structure of the code and the hardware specification may determine the feasibility of the quantity of parallelisation (Langenbach, Thesing, & Heckmann, 2002; Raghavendra et al., 2010; Walsh & Miller, 1998; S. R. Zebari & Yaseen, 2011).

It is important to first comprehend the notion of assembly lines, in an automated production plant, before comprehension of the idea of pipelining, where the former comprises of items linked from distinct parts and the result of one phase becomes the entry data to another phase. Initiation of pipelining in a processor P follows the following stages:

- Sub-divide the input operation into a continuous subtask. These subtasks will make phases of pipeline, also termed as segments.
- Every phase Si of the pipeline in accordance with the subtask will execute some activity on a definite set of operands.
- After completion of phase Si, the outcomes are communicated to the succeeding phase Si+1 for the execution that follows.
- The phase Si receives a new arrangement of input from the preceding phase Si-1.

In this way, parallelism in a pipelined processor is achievable such that m independent functions can take place concurrently in m units (Brifcani & Brifcani, 2010; Solanki & Singh, 2017)

Parallel computers may be generally grouped into two basic categories based on the topology of how they are linked: multi-processors with shared memory and multicomputers with distributed memory (AL-Zebari, Zeebaree, Jacksi, & Selamat, 2019; Kim, Kim, & Lee, 2010).

One of the main ideas of network computing is the client-server model. Client/server application entails two core entities: the client and the server. Normally, the server and the client are located on distinct computers (Zeebaree, Haron, Abdulazeez, & Zebari, 2019).The notion of the client/server programming is that the server is the giver of services. A client program makes a request service from the server through conveying a message. Then the server executes the tasks requested by the clients. Servers are most active as they wait for a client request. During the waiting moment, servers can execute other activities. Unlike the client, the server ought to run regularly as clients can make requests at any time. On the contrary, clients only need to run when there is need for a service (Abdulazeez et al., 2018; Cardellini, Colajanni, & Yu, 1999; Zamfir, 2010; R. R. Zebari, Zeebaree, & Jacksi, 2018).

The key problem in this work is how to make these systems more efficient and increase the throughput with high performance, and how to divide the heavy-load into sub-parts and distribute it over multiple server-hosts in order for the total processing-time to decrease. In this work, we addressed this problem by preparing a complete distributed memory system that has the capability of applying the principles of parallel processing with respect to the client/server principle on the cluster-networking using four computers; one acts as a client-host and others act as a server-host. On the server-side the application is partitioned into the number of threads which is equal to the number of server-hosts processing them on server-hosts in minimum time; the pipeline technique is applied with the parallel processing approach needed to increase the system throughput. The paper is organised as follows: the first section is an introduction to the main concepts. Section two discussed the theory of System Design, the Hardware Part of the System, Software Part of the System, Case study: Merge Sorting Algorithm. Section three presents a detailed description of the contents of the experimental results. Finally, section five presents the conclusions of this paper.

System Design

In this section we will explain the general structure of the proposed system. The system consists of hardware and software parts. The system has been built depending on the client/server structure with two processing-sides (client and servers) sides.

Hardware Part of the System

The proposed system is organised with two hardware-sides which are the client-side and server-side, and the network to connect both sides designed according to star topology. The client-side consists of only one computer which contains the main program that controls the sending message to other side. The client-side usually represents the user interface portion of an application and contains the original data of the case study that must be sent to the other side and receives the results that were calculated by the servers-side.

The server-side consists of three identical computers connected with each other by a switch (in this work we used CPU i7 2.40 GHz). Each server must contain a program that has the ability to receive data, making the required processing and then sending them to the client-side.

The Software Part of the System

The software part of the system is also organised with the client software-side and server software-side and each side has its own Graphic User Interface (GUI) that forms the main algorithm that is related to this work as shown:

Step-1: Initialising Client-host and Server-hosts.
Step-2: Checking the status of the connection among hosts. (Determine the server-hosts IP address and connected the client-side to the server-side).
Step-3: At the client-side, determine the size of the array and create unsorted files.
Step-4: Send the files to the servers-side.
Step-6: The servers-side receives the unsorted files from the client-side and converts them to structure.
Step-7: At the server-side partition of an unsorted array is done according to the number of server-hosts.
Step-8: Each server-host implements a merge sorting algorithm on its own part of the data to convert it from unsorted status to sorted status in parallel and the output of each server-host is the input to the next server-host.
Step-9: Last, the server- host sends the sorted file to the client-side
Step-10: The client-side receives the sorted file from the server-side and saves all these results.

Case Study: Merge Sorting Algorithm

Sorting is one of the most common operations nowadays used in most areas of scientific research. All the sorting algorithms are problem-specific which means every sorting algorithm executes better on particular kinds of issues. In this work Merge Sort method is studied because it is a suitable case study for an application displaying the effects of pipelined parallel processing and a very effective sorting technique when data size is considerably large. Also, it is an efficient divide-and-conquer sorting algorithm and frequently used to introduce the divide-and-conquer approach itself.

When partitions of an unsorted array are completed by the first server, it then sends partitions for the remainder of servers, for each server implements merge sort algorithm on its own sub unsorted array of data in parallel to sort it. These algorithms typically follow a divide-and-conquer approach by breaking the problem into several subproblems (at the midpoint when the number of elements is even, otherwise the right half has one extra) that are similar to the original problem but smaller in size. If the array has one item it is sorted, but if the array has more than one item, then the array is split until each element of the array to be sorted is an array of length 1; the subproblems are solved recursively, and then these solutions are combined to create and reach the size of the current array. Output of each server-host is the input to next server-host.as shown in figure (1)

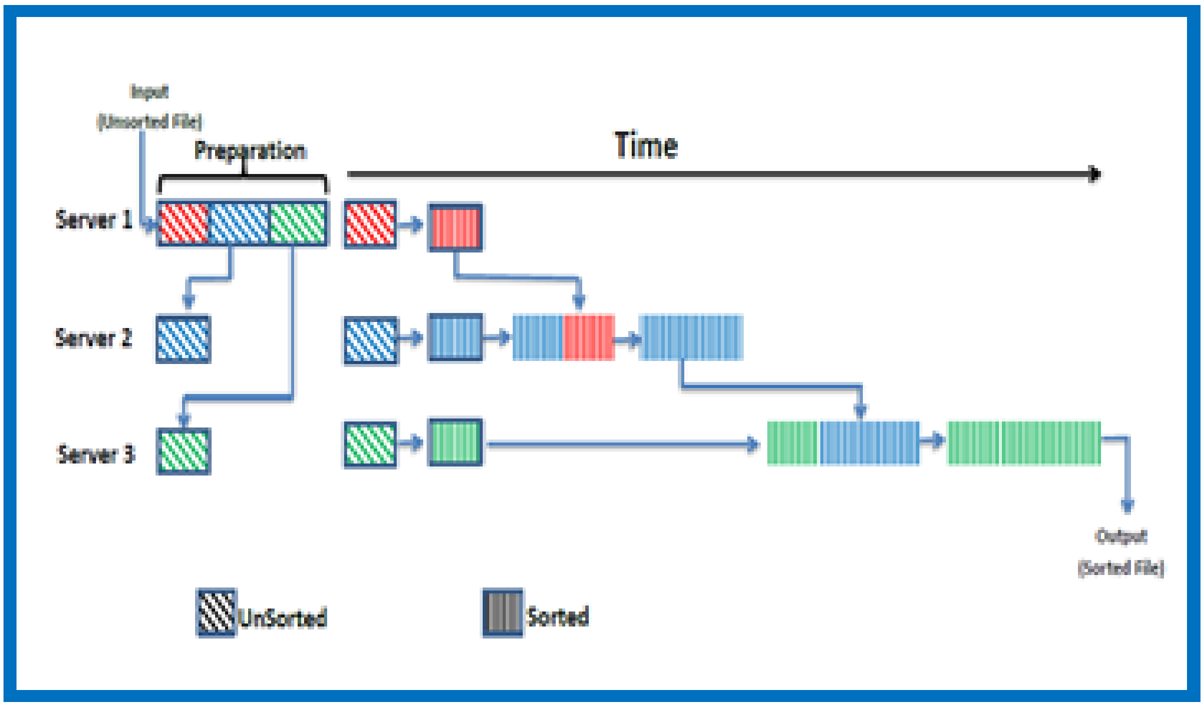
The pseudo code for merge sort looks like:

```
mergesort(L = a1, a2, . . ., an: array of numbers)
if (n = 1) then return L
else
m = ⌊n/2⌋
L1 = a1, a2, . . ., am
L2 = am+1, am+2, . . ., an
return merge(mergesort(L1),mergesort(L2))
```

And the pseudo code for the merge function might look like:

```
merge (L1,L2: sorted arrays of numbers)
          A= empty list
          i = counter of L1 elements
          j = counter of L2 elements
          k= counter of A elements
          i =j =k= 1
While there are still elements in both L1 and L2 that have
not been copied to A do
          begin
               if (L1[i]< L2[j])
             then
                begin
                    A [k] = L1[i]
                    i++
                    k++
                   end
              else
                begin
             A[k] =L2 [j]
             j++
             k++
                end
           end
Place remaining (sorted) elements in L1 or L2 in A
       return A
```

Figure 1. Diagram of Implementation Merge Sorting Algorithm in Pipelined parallel processing



Results and Discussion

The effect of the pipelined parallel technique appears in greater load-tasks and the values are illustrated in table 1)and plotted in Figures 2 and 3 that represent total sorting time and stages time.

Table 1: Times (Sec.) of Pipelined Parallel Technique

No. elements	Times (Sec.) of Pipelined Parallel Technique			
	Stage 1	Stage 2	Stage 3	Total time
10,000,000	1.058	0.108	0.140	1.306
25,000,000	2.801	0.273	0.374	3.448
50,000,000	5.853	0.624	0.764	7.241
60,000,000	7.145	0.655	0.920	8.720
70,000,000	8.361	0.780	1.077	10.218

The value of stage one returned by server number one, the value of the second stage returned by server number two and the value of the third stage returned by server number three to the client-host when they completed their tasks., where the number of elements is 10000000, 25000000, 50000000, 60000000 and 7000000.

Figure 2 shows that first stage in the Pipelined Parallel Technical consumed more time than other stages because the first stage performs the sorting operation on its part of the unsorted file such that the total time of this operation is calculated by the full time that is consumed by the stage one in order to be completed and reached to the second stage (server two).

Another reason that makes the first stage consume more time than other stages in some cases, is when dividing the original unsorted array to three parts (sub unsorted array) the data may be not partitioned equally; this means some partitions are bigger than others. Bigger partitions are usually performed by the first server. Also the curves of these figures illustrate that stage two consumes less time than other stages because in the pipelined parallel technique all server-hosts implement the merge sort step on their own part in parallel; so the consumed time by the second server, when receiving part one from first server and merging it with its own sorted part, also consumes time by the third server when receiving the merged part from second server.

Figure 2. Stages time for 50 million elements

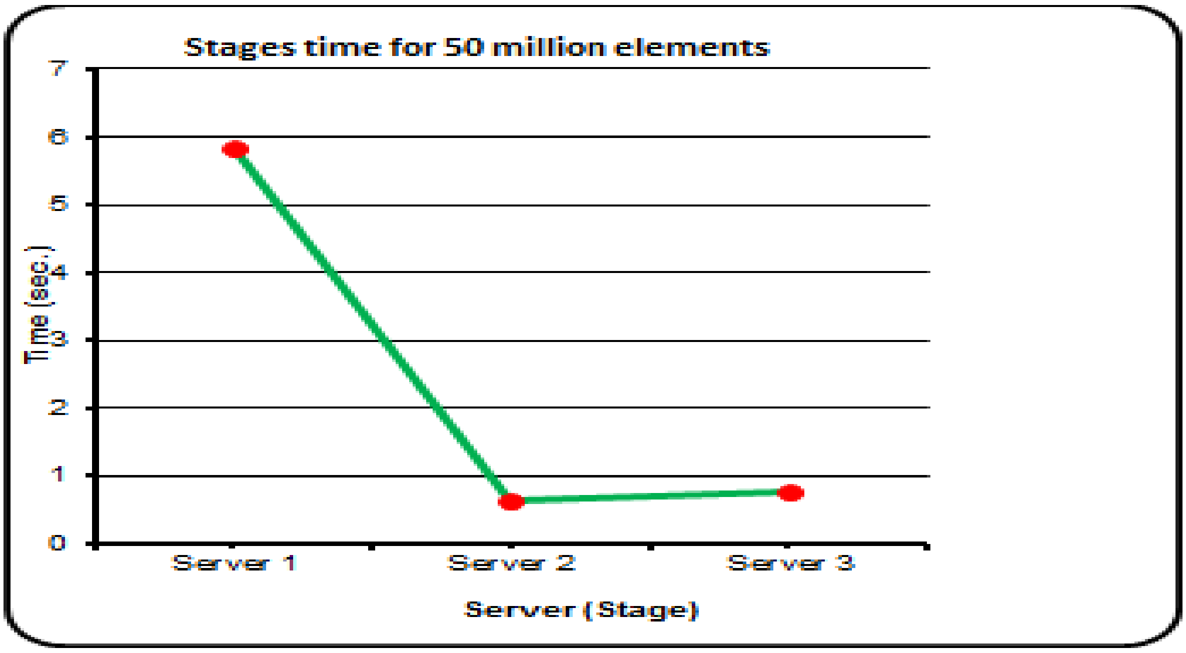
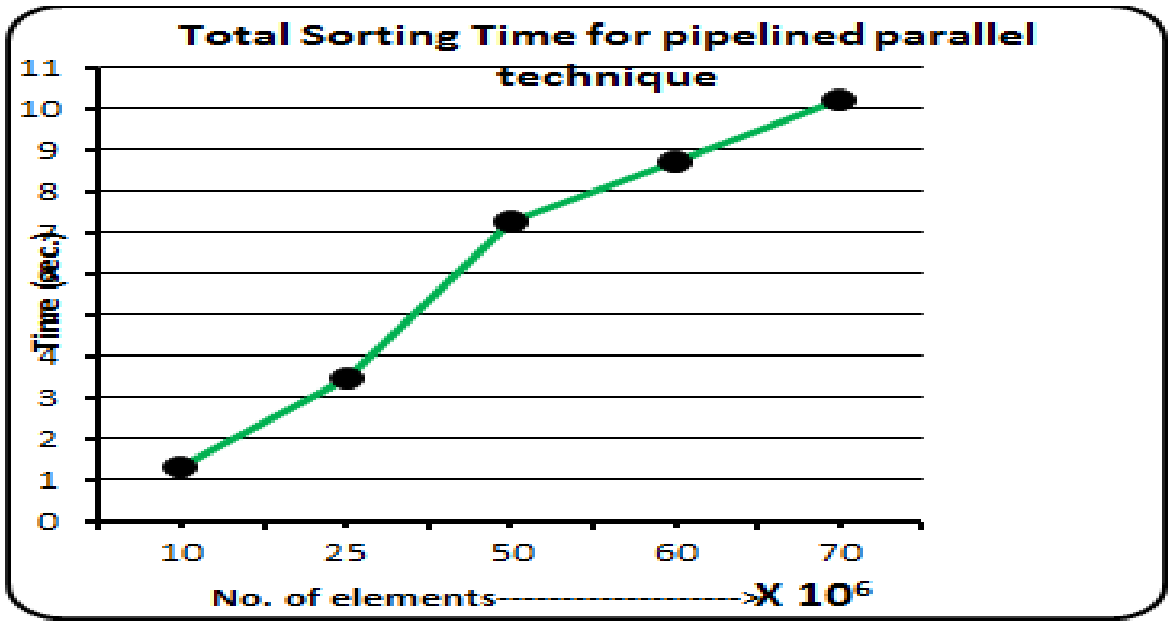


Figure 3 shows a positive relationship between total sorting time and load for all operation modes. It means total sorting time will be increased by increasing the load; this is because the number of elements and number of operations for each order are increased. For instance when the 10,000,000 element is tested, the total sorting time for the pipelined parallel technique to perform these operations takes 1.306 seconds, while when increasing the load to the 70,000,000 element total sorting time increased to 10.218 seconds.

Figure 3. Total Sorting Time for the pipelined parallel technique



Conclusion

In this paper we have presented pipelined parallel processing implementation based on distributed memory and is discussed by depending on the client/server system. This system consists of four nodes, one of them is a client and the others are servers that have identical properties. The system performance was improved by implementing and designing parallel applications on the system and dividing the workload among multiple server-hosts at the server side. Experimental results showed the effects of parallel processing that appeared clearly on the total sorting time by increasing the load. Also, the results showed the time will be reduced by taking advantage of both the parallel processing and pipeline approach.

REFERENCES

Abdulazeez, A. M., Zeebaree, S. R., & Sadeeq, M. A. (2018). Design and Implementation of Electronic Student Affairs System. Academic Journal of Nawroz University, 7(3), 66-73.

AL-Zebari, A., Zeebaree, S. R., Jacksi, K., & Selamat, A. (2019). ELMS–DPU Ontology Visualization with Protégé VOWL and Web VOWL. Journal of Advanced Research in Dynamic and Control Systems, 11, 478-485.

Atiqullah, M. M., & Rao, S. S. (2000). Simulated annealing and parallel processing: an implementation for constrained global design optimization. Engineering Optimization+ A35, 32(5), 659-685.

Brifcani, A. M. A., & Brifcani, W. M. A. (2010). Stego-based-crypto technique for high security applications. International Journal of Computer Theory and Engineering, 2(6), 835.

Buluç, A., & Madduri, K. (2011). Parallel breadth-first search on distributed memory systems. Paper presented at the Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis.

Cardellini, V., Colajanni, M., & Yu, P. S. (1999). Dynamic load balancing on web-server systems. IEEE Internet computing, 3(3), 28-39.

Chae, M. J., & Abraham, D. M. (2001). Neuro-fuzzy approaches for sanitary sewer pipeline condition assessment. Journal of Computing in Civil engineering, 15(1), 4-14.

DeBenedictis, E. P. (1988). Parallel processing network and method. In: Google Patents.

Drozdowski, M. (2009). Classic scheduling theory. In Scheduling for Parallel Processing (pp. 55-86): Springer.

Frachtenberg, E., & Schwiegelshohn, U. (2007). New challenges of parallel job scheduling. Paper presented at the Workshop on Job Scheduling Strategies for Parallel Processing.

Haji, L. M., Zeebaree, S. R., Jacksi, K., & Zeebaree, D. Q. (2018). A State of Art Survey for OS Performance Improvement. Science Journal of University of Zakho, 6(3), 118-123.

Junji, N. (2004). Parallel computing techniques. Humboldt-universität berlin. Center for Applied Statistics and Economics (CASE), 27.

Kim, M., Kim, D., & Lee, H. (2010). Embedding algorithms for star, bubble-sort, rotator-faber-moore, and pancake graphs. Paper presented at the International Conference on Algorithms and Architectures for Parallel Processing.

Langenbach, M., Thesing, S., & Heckmann, R. (2002). Pipeline modeling for timing analysis. Paper presented at the International Static Analysis Symposium.

McCalpin, J. D. (1995). Memory bandwidth and machine balance in current high performance computers. IEEE computer society technical committee on computer architecture (TCCA) newsletter, 2(19–25).

McDonald, R. J., & White, N. M. (1994). Parallel information processing in the water maze: evidence for independent memory systems involving dorsal striatum and hippocampus. Behavioral and neural biology, 61(3), 260-270.

Merigot, A., & Petrosino, A. (2008). Parallel processing for image and video processing: Issues and challenges. Parallel Computing, 34(12), 694-699.

Raghavendra, P., Behki, A. K., Hariprasad, K., Mohan, M., Jain, P., Bhat, S. S., . . . Prabhu, V. (2010). A study of performance scalability by parallelizing loop iterations on multi-core SMPs. Paper presented at the International Conference on Algorithms and Architectures for Parallel Processing.

Sasikumar, M., Shikhare, D., & Prakash, R. P. (2014). Introduction to parallel processing: PHI Learning Pvt. Ltd.

Solanki, C. S., & Singh, H. K. (2017). Principle of Texturization for Enhanced Light Trapping. In Anti-reflection and Light Trapping in c-Si Solar Cells (pp. 65-82): Springer.

Walsh, R. J., & Miller, B. (1998). Parallel processing system for virtual processor implementation of machine-language instructions. In: Google Patents.

Zamfir, C. (2010). DiaSys-A Dialogue System based on Conditional Knowledge. Annals of the University of Craiova-Mathematics and Computer Science Series, 37(3), 78-91.

Zebari, D. A., Haron, H., Zeebaree, D. Q., & Zain, A. M. (2019). A Simultaneous Approach for Compression and Encryption Techniques Using Deoxyribonucleic Acid. Paper presented at the 2019 13th International Conference on Software, Knowledge, Information Management and Applications (SKIMA).

Zebari, D. A., Haron, H., Zeebaree, S. R., & Zeebaree, D. Q. (2018). Multi-Level of DNA Encryption Technique Based on DNA Arithmetic and Biological Operations. Paper presented at the 2018 International Conference on Advanced Science and Engineering (ICOASE).

Zebari, R. R., Zeebaree, S. R., & Jacksi, K. (2018). Impact Analysis of HTTP and SYN Flood DDoS Attacks on Apache 2 and IIS 10.0 Web Servers. Paper presented at the 2018 International Conference on Advanced Science and Engineering (ICOASE).

Zebari, S. R., & Yaseen, N. O. (2011). Effects of Parallel Processing Implementation on Balanced Load-Division Depending on Distributed Memory Systems. Journal of university of Anbar for Pure science, 5(3), 50-56.

Zeebaree, D. Q., Haron, H., Abdulazeez, A. M., & Zebari, D. A. (2019). Machine learning and Region Growing for Breast Cancer Segmentation. Paper presented at the 2019 International Conference on Advanced Science and Engineering (ICOASE).