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

Effect of Multi-Core Processors on CPU-Usage with Heavy-Load Problem Solving

Adnan Mohsin Abdulazeez[a], Nishtiman Rashid Ali[b], Diyar Qader Zeebaree[c], [a]Presidency of Duhok Polytechnic University, Duhok, Kurdistan Region, Iraq, [b]Computer Science Department, Duhok Polytechnic University, Duhok, Kurdistan Region, Iraq, [c]Research Center of Duhok Polytechnic University, Duhok, Kurdistan Region, Iraq, Email: [a]adnan.mohsin@dpu.edu.krd, [b]dqszeebaree@dpu.edu.krd

Traditional approaches to enhance the execution of a processor utilised more transistors on chips and a rising clock rate. Conversely, this result has attained its limitation. As a result of excess heat and utilisation of more power, chip developers have signalled a shift in microprocessor design from single core to multi-core processors. Multi-core is an integrated circuit chip that utilises more than one core place within a single processor. This approach is used to split the computational activity of a threaded application and disperse it over multiple execution cores so that the computer can benefit from a better performance of the system. In this paper we propose the effect of a multi-core processor on CPU-usage; we address this problem by designing a parallel application by using client/server principles. We divided the workload among multiple cores at the server side for testing performance in terms of execution time and CPU-usage. This paper is based on the Matrix multiplication case study; all algorithms related to this case study are implemented using Borland C++ Builder language. Experimental results show that our algorithms are computationally fast, which is reducing the execution time and maximising throughput; also it is the finest algorithm to full utilisation of the available processing cores.

Key words: *Parallel processing, Multi-core processors, Matrix Multiplication.*

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; Mahdi, Al-Mayouf, Ghazi, Wahab, & Idris, 2018; Zebari, Haron, Zeebaree, & Zeebaree, 2018), due to the expansion in transmission of data through the Internet and its constrained transfer speed, and time taken by the data to achieve the goal additionally expanded (Zebari, Haron, Zeebaree, & Zain, 2019). Performance of sequential computers increases incredibly fast; it is insufficient for a large number of challenging applications. Applications requiring much more performance are numerical simulations in industry and research as well as commercial applications such as query processing, data mining, and multi-media applications. Current hardware architectures contributing to high execution do not only exploit parallelism on a very fine grain level within a single processor, but apply a medium to many processors concurrently to a single computation (Mohr, 2006). Recent developments in the computing industry have signalled a shift in microprocessor design towards the use of multiple processing cores in single chip. This allows a single device to execute multiple instructions simultaneously. The process by which multiple instructions are executed in parallel on multiple CPUs is referred to as parallel processing (Grant & Afsahi, 2007). Parallel processing refers to speeding 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 to be executed across n processor might execute n times faster than it would using a single processor (Haji, Zeebaree, Jacksi, & Zeebaree, 2018; Young, 2009), and accomplish the activity better than it would on a single core processor. Multi- core processors have the power to accomplish several activities at a single moment (Nanehkaran & Ahmadi, 2013). Manufacturers typically merge the cores into one integrated circuit die (known as a chip multiprocessor or CMP). Designers may pair cores in a multi-core gadget tightly or loosely. For instance, cores may or may not share caches, and they may apply message passing or shared memory inter-core communication techniques. The usual network topologies to link cores comprise of bus, ring, two-dimensional mesh, and crossbar (Rashid, Sharif, & Zeebaree, 2018; D. Q. Zeebaree, Haron, Abdulazeez, & Zebari, 2019).

Client/server is a distributed computing model where the server processes requests from the client applications. Clients and servers typically run on separate computers linked via a computer network. A client application is a process or program that conveys information to a server via the network. Those messages' requisitions entail activity execution by the server. The client controls restricted resources like the display, keyboard, local disks and other peripherals. The server process or program listens for client requisitions that are relayed through the network. Servers admit those requisitions and discharge results (Brifcani & Al-Bamerny, 2010; A.-Z. S. R. Zeebaree, Adel, Jacksi, & Selamat). Server processes usually run

on strong PCs, workstations or on mainframe computers (Chowdhury, 2010; Hemmendinger, Ralston, Reilly, & Maffeis, 1998; D. Q. Zeebaree, Haron, Abdulazeez, & Zeebaree, 2017).

The key problem in this work is how to split the workload among multiple cores and how to distribute it over multiple execution cores. If the workload is not well optimised or designed, then it gives additional burden to the execution task. In the other word, if some cores are just standing by, the multi-core will become meaningless. But if every core takes a part of a problem, it is able to work efficiently.

To increase the system performance in this paper, the problem addressed by design is a system using two computers; the first one acts as a client and the second one act as server; the type of the server must be a multi-core processor. In the server-side partitioning the application into the amount of threads such that the number of threads created is similar to the number of cores. Then each thread is allocated to each core; the field of parallel processing is concerned with architectural and algorithmic methods for enhancing the performance of digital computers through various forms of concurrency (Haji et al., 2018).

Multi-core is a term associated with parallel processing and one of the most important trends of increasing the speed of the processor to get a boost in performance. A multi-core processor is a single computing element, with more than one independent actual central processing units (called "cores") (Hassan, Abdulazeez, & TİRYAKİ, 2018; Rashid et al., 2018). A core is the section of the processor which achieves reading and implementation of the commands. Single core processors can fulfil a single requirement at a time. On the contrary, as the name insinuates, Multi-core processors comprise more than a single core. An ordinary instance would be a dual core processor. What favours a multi-core processor above a single core one is that the multi-core processor can either utilise both its cores to fulfil a single activity or it can traverse threads which splits activities linking both its cores, so that it discharges double the duration it would take for testing performance in terms of processing time and CPU-usage. This is crucial because it determines how effectively those computing resources can be utilised.

System Design

This section explains the general structure of the proposed system. The system consists of two main parts: the first part is related to hardware of the work and the second is about the software parts. The system has been built depending on the client/server structure with two sides (client and server).

Hardware Part of the System

The proposed system is organised with two hardware-sides (client and server). Each side consists of only one computer and is connected both sides by the network. The client-side has 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; it is responsible of validating data entered by the user and transmitting requests to the server side. Original data of the case study that must be sent to the other side and calculated results by the server-side, are stored on client-host. Also, the server-side has only one computer; the type of the server must be multi-core processors (in this work we use CPU i7 with 8-cores). The server host contains a program that has the ability of receiving data, making the required processing, calculating the results and then sending them to the client side.

The Software Part of the System

The software part of the system is also organised with two software-sides (client software-side and server software-side); each side has its own Graphic User Interface form (GUI-form). This form is designed by Borland C++ builder tools. The main algorithm that is related with this work is:

Begin

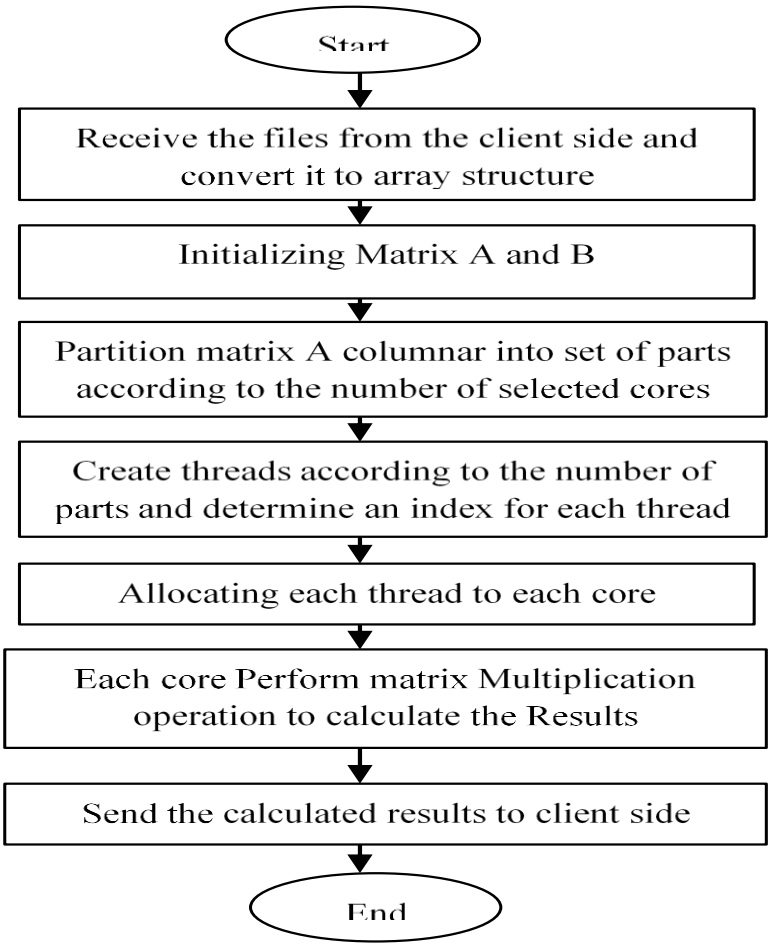
Step-1: initialising server-side and client-side.
Step-2: Checking the status of the connection. (Determine the server IP address and connected client-side to server-side).
Step-3: At the client side determine the size of the array and create two files which are represented by matrix A and B.
Step-4: At the client- side, choose the number of cores to be used by the server-side and send the job to the server-side.
Step-5: The server side receives the files from the client side and converts them to the array structure.
Step-6: At the server-side partition matrix A columnar according to the number of selected cores.
Step-7: Perform matrix multiplication operation on all partitions in matrix A with shared matrix (matrix B) to calculate results.
Step-8: Send the calculated result to client side.
Step-9: Client side receives the output results from the server side and saves all these results.

End.

Parallel Matrix multiplication

The server side receives two files from the client side and converts them to matrix structures. In this work to reduce memory space, the server-side partitions matrix A columnar according the number of selection cores, but matrix B does not partition; it is shared between all parts of the partitioned matrix. For each partition in the partitioned matrix, determine an index and create threads such that the numbers of created threads are equal to the number of selected cores at the client side then allocate each thread to each core to perform the operations. After the partition operation is complete, the server runs the appropriate subroutines according to the requirement of the client-side, in this case perform the matrix multiplication operation on matrix A (partitioned matrix) and matrix B (shard matrix); this is to calculate the results such as CPU-usage and CPU time for each core without user interaction, and then the server-side sends the calculated results to the client-side. The function of the partitioning and multiplication are illustrated in the flowchart shown in Figure 1.

Figure 1. parallel matrix multiplication

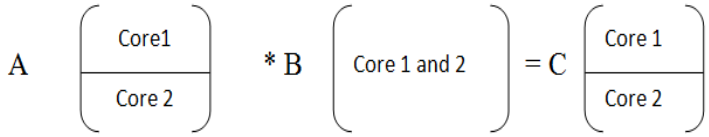


Case Study: Matrix

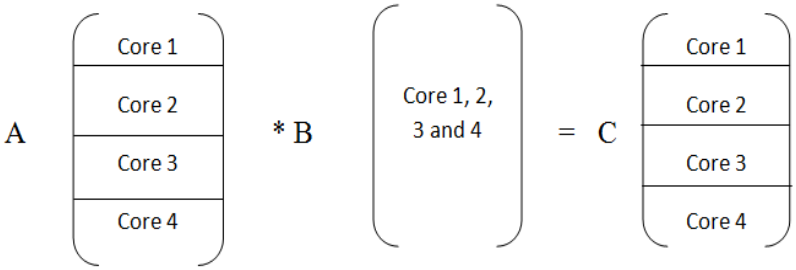
Multiplication

To determine the importance of the parallel processing technique, researchers usually depend on one or more case studies to explain the algorithm and produce the results with less time of execution compared with single processing techniques. Matrix multiplication is a significant application commonly used in almost all areas of scientific research and the famous type of case studies related to parallel processing. In this work we use a computer the that isa multi-core processor (i7 with 8-cores) as a server and performs matrix multiplication operation as part of matrix algebra according to the following situation:

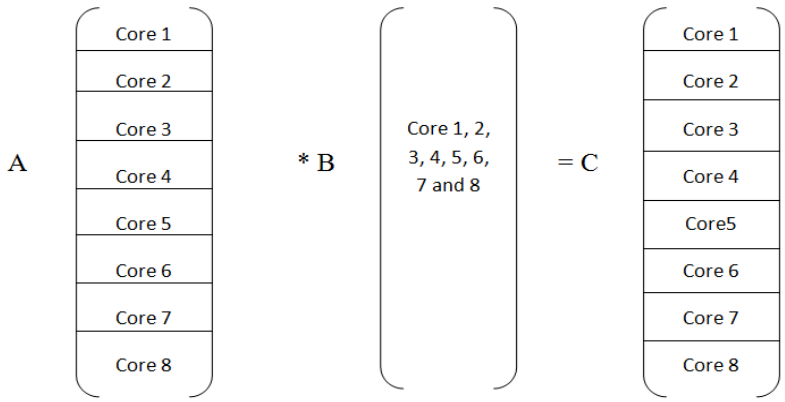
First Case: Apply multiplication operation for all complete matrices on two cores, after dividing matrix A into two parts.



Second Case: Apply multiplication operation for all complete matrices on four cores, after dividing matrix A into four parts.



Third Case: Apply multiplication operation for all complete matrices on eight cores, after dividing matrix A into eight parts.



Results and Discussion

To obtain results, two computers are used in the system; one acts as a client and the others act as a server. The designed algorithm: deal with two original matrices which is the equal size. The sizes of the matrix which are used in this work are: 500, 4000, 8000, 10000 and 20000. In this work the obtained results are divided into three main groups: the first one is related with the maximum CPU time; this time is a latest returning result by the server side which is represents the longest value of consumed CPU-time for all selected cores at the server side as acceptable values to be depended. This value is illustrated in Table 1 and plotted as Figures 1 to 3 and Figures 7 to 9.

The second group is related with the average value of CPU-time for all cores at the server side which is represented by the average of related times for all selected cores as an acceptable value to be depended; this values are illustrated in Table 2 and plotted in Figures 4 to 6 and Figures 10 to 12.

The third groups are related with the overall CPU usage; this value represents the total CPU-usage that is used by all selected cores at the server side, and these values are illustrated in Table 3 and plotted in Figures 13 to 15.

Table 1: Maximum CPU time

Matrix size	2-cores	4-cores	8-cores
500	0.484	0.266	0.109
4000	958.891	337.079	195.407
8000	6500.965	3501.204	2249.391
10000	13377.979	7227.563	4825.813
20000	170365.209	89453.329	54144.172

Table 2: Average CPU time

Matrix Size	2-cores	4-cores	8-cores
500	0.484	0.266	0.109
4000	958.859	335.996	192.674
8000	6500.865	3478.339	2231.421
10000	13376.732	7185.739	4769.535
20000	167363.18	89065.328	51911.762

Table 3: Overall CPU-usage

Matrix Size	2-cores	4-cores	8-cores
500	24	48	96
4000	24	48	96
8000	24	48	96
10000	24	48	96
20000	24	48	96

To evaluate the system performance compare the maximum CPU-time, average CPU-time and Overall CPU-usage using CPU with selecting 2-cores, 4-cores and 8-cores. Figures 1 to 3 illustrated the relationship between of maximum CPU-time and the number of orders (500, 4000, 8000, 10000 and 20000) using CPU with selecting two-cores, four-cores and eight-cores respectively. In all these figures it can be seen that the maximum CPU time will be increasing the load on the same number of selected cores; this is because the number of elements and operations for each order are increasing. This means that there is a positive relationship between CPU time and the load.

At the beginning; the effect of the workload on the processing time do not appear well, but its effects are clearly observed with increasing the load (this condition appeared well at order 8000 and ascending).

Figure 2. Maximum CPU using CPU with selecting 2-cores for matrix algebra (500,4000,8000,10000 and 20000)

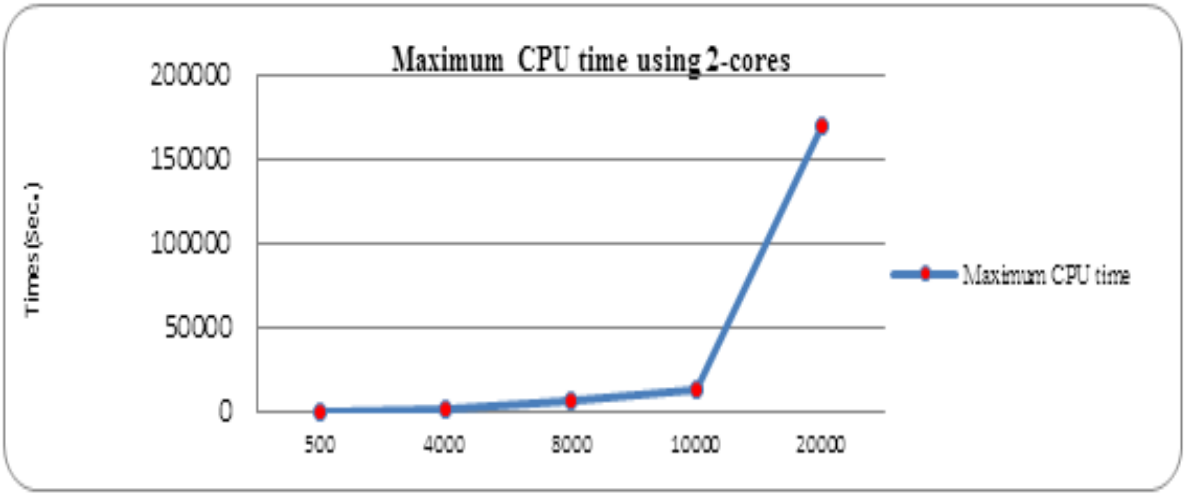


Figure 3. Maximum CPU using CPU with selecting 4-cores for matrix algebra (500,4000,8000,10000 and 20000)

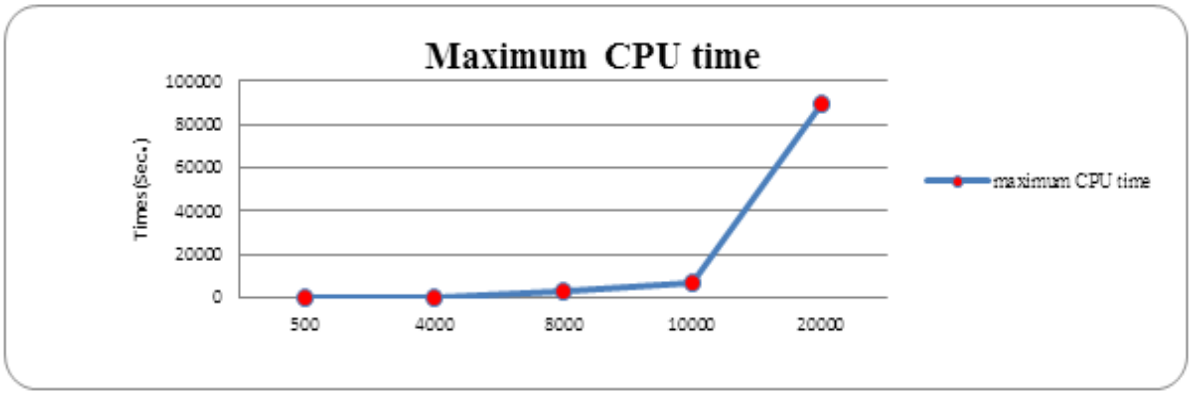
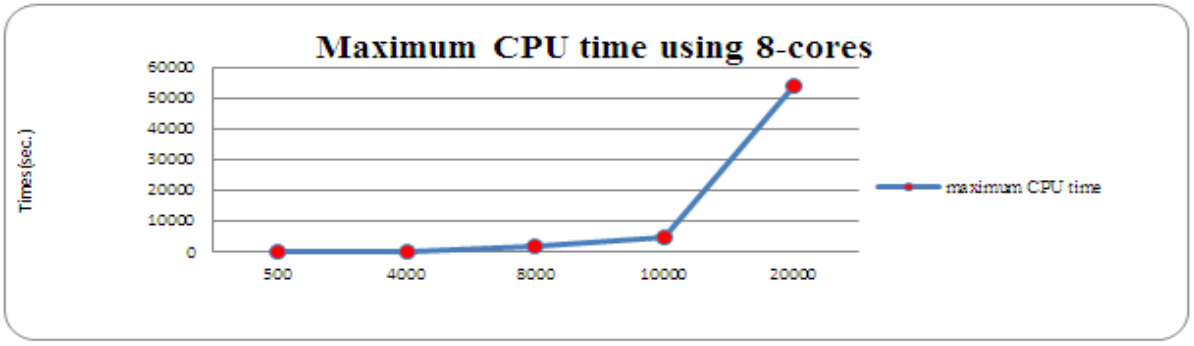


Figure 4. Maximum CPU using CPU with selecting 8-cores for matrix algebra (500,4000,8000,10000 and 20000)



The average result of CPU time is seen in Figures 4 to 6, as they effected as the situation of maximum CPU time.

Figure 5. Average CPU using CPU with selecting 2-cores for matrix algebra (500, 4000, 8000, 10000 and 20000)

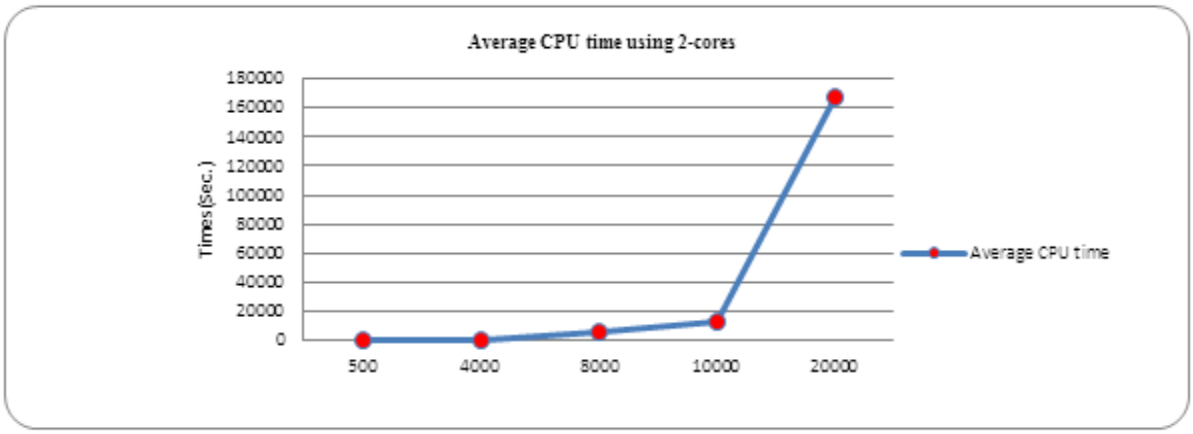


Figure 6. Average CPU using CPU with selecting 4-cores for matrix algebra (500, 4000, 8000, 10000 and 20000)

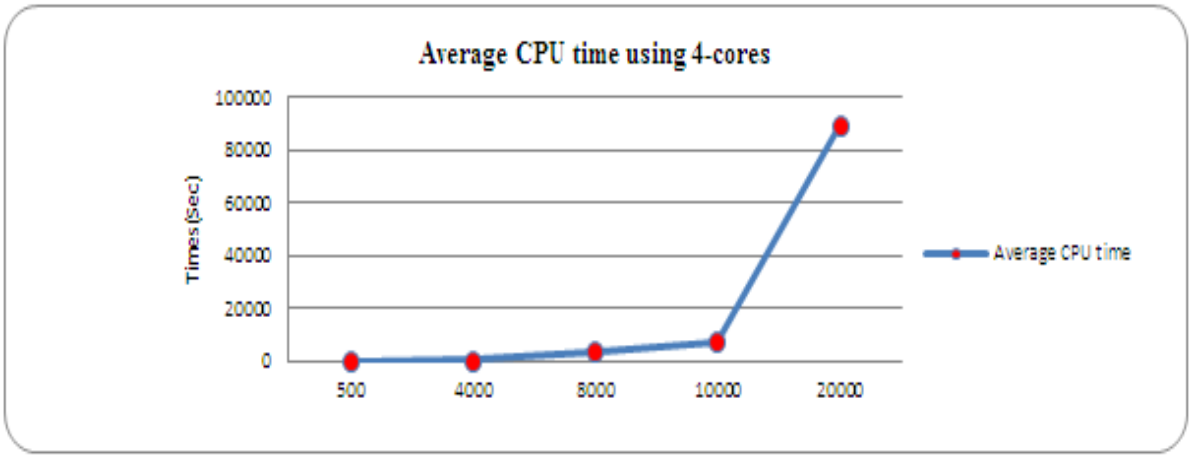
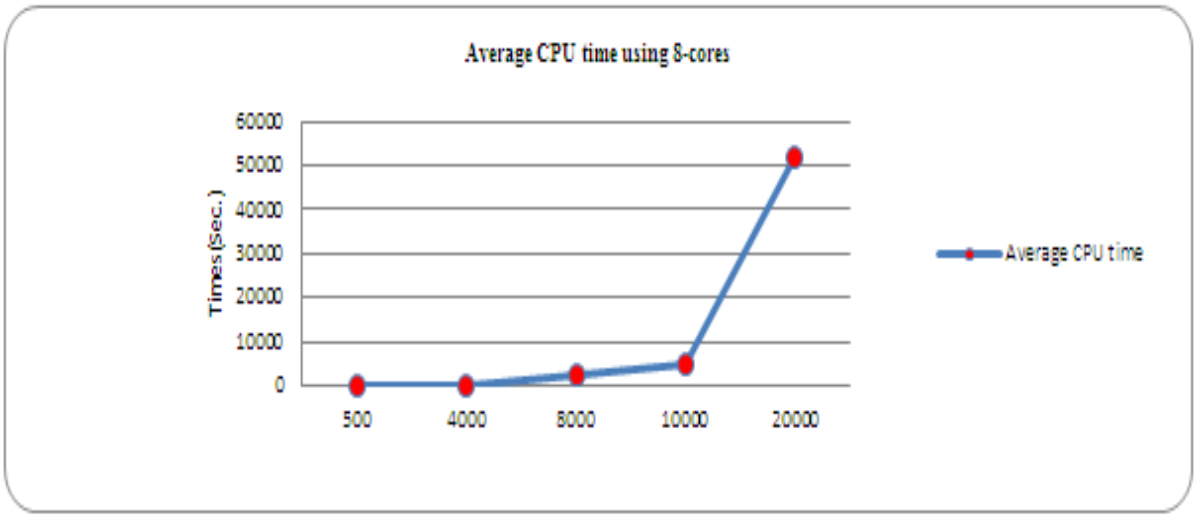


Figure 7. Average CPU using CPU with selecting 8-cores for matrix algebra (500, 4000, 8000, 10000 and 20000)



Figures 8 to 10 illustrate the relationship between the maximum CPU time and the number of cores for orders ((500 and 4000), (8000 and 10000) and 20000) respectively. In all these figures it is seen that the time will be decreased by increasing the number of selection cores such as in Figure (8) when tested the order 500 and 4000 on the CPU with selecting 2-core; the maximum CPU time is equal to 0.484 and 958.891. Second, while when the same orders tested on the processor with selecting 4-cores, the maximum CPU time will be decreased to 0.266 and 337.079 seconds and with selecting 8-cores the maximum CPU time will be decreased to 0.109 and 195.407 seconds respectively. This means that there is a negative relationship between the maximum CPU time and number of selection cores.

In the same way for orders 8000, 10000 and 20000, the maximum CPU time reducated by increasing the number of selection cores as shown in Figures 9 and 10. These effects for a small load do not appear well, but the effects of parallel processing technique appeared well as the load is growing; this is clear for order 20000; the maximum CPU-Time is reduced from 170365.209 sec. using 2-cores to 54144.172 sec. using 8-cores; the reduced time is 32 hours.

Figure 8. Maximum CPU time using CPU with selecting 2-core,4-core and 8-cores for order 500 and 4000

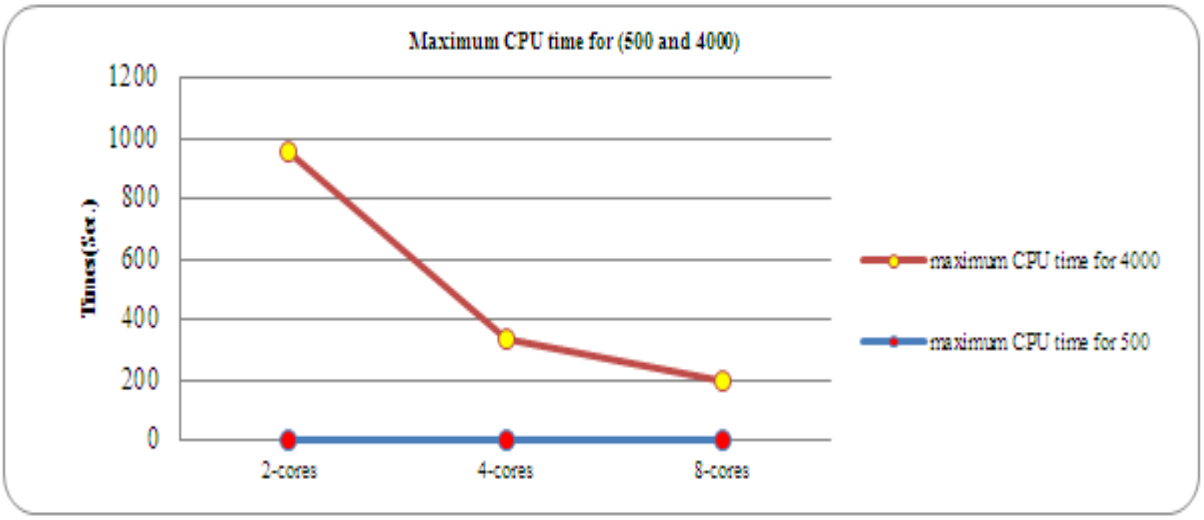


Figure 9. Maximum CPU time using CPU with selecting 2-cores, 4-cores and 8-cores for order 8000 and 100000

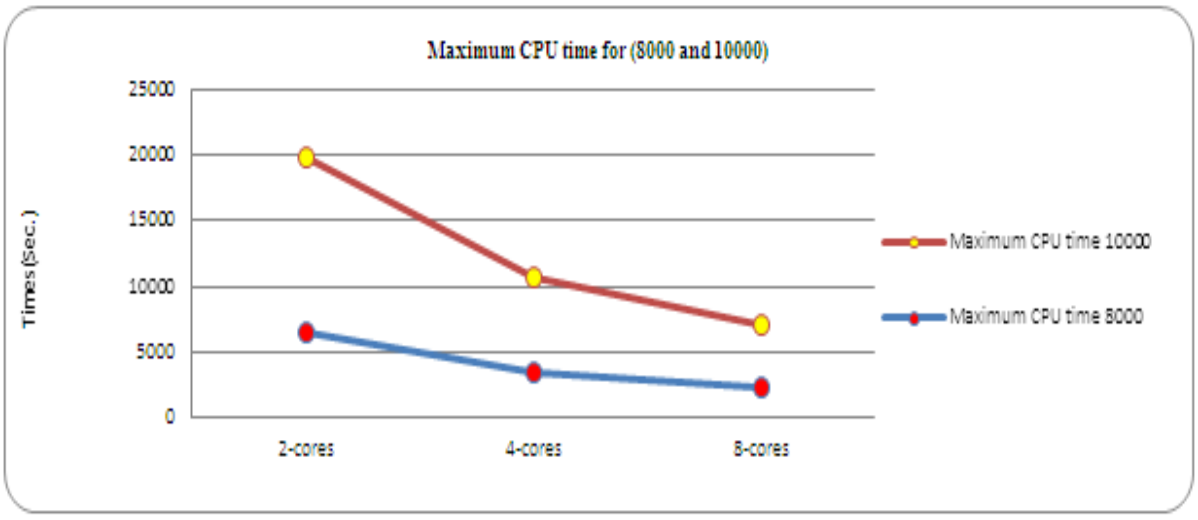
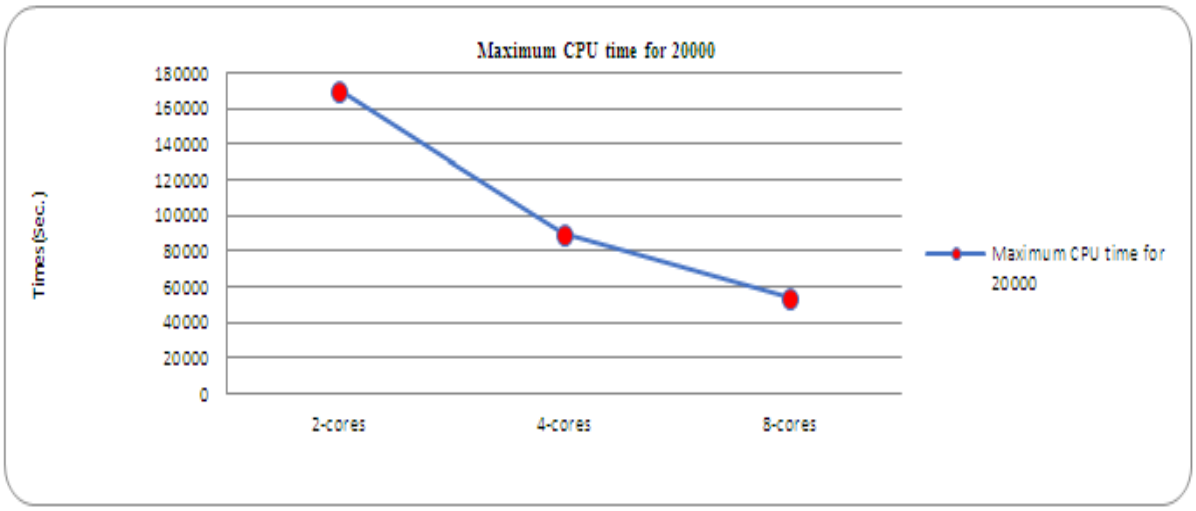


Figure 10. Maximum CPU time using CPU with selecting 2-cores, 4-cores and 8-cores for order 20000



The results of the average CPU time are illustrated in Figures 11 to 13, affected as the case of maximum CPU time.

Figure 11. Average CPU time using CPU with selecting 2-cores, 4-cores and 8-cores for order 500,4000

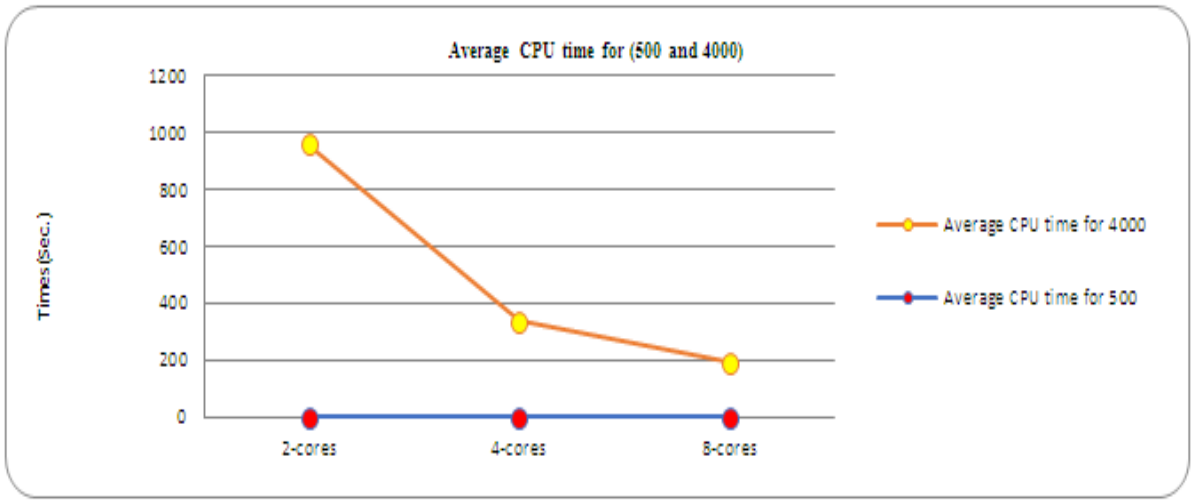


Figure 12. Average CPU time using CPU with selecting 2-cores, 4-cores and 8-cores for order 8000 and 10000

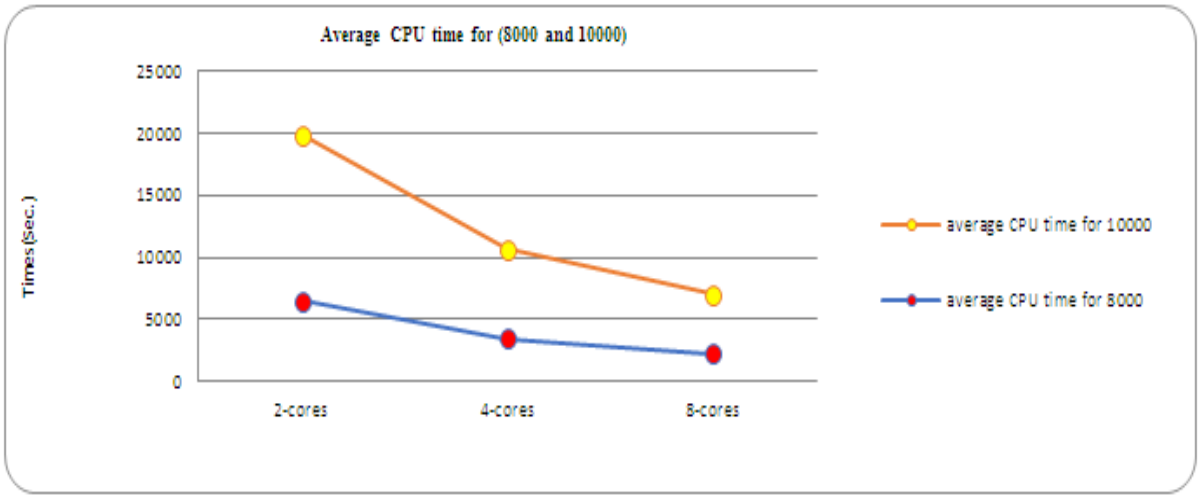
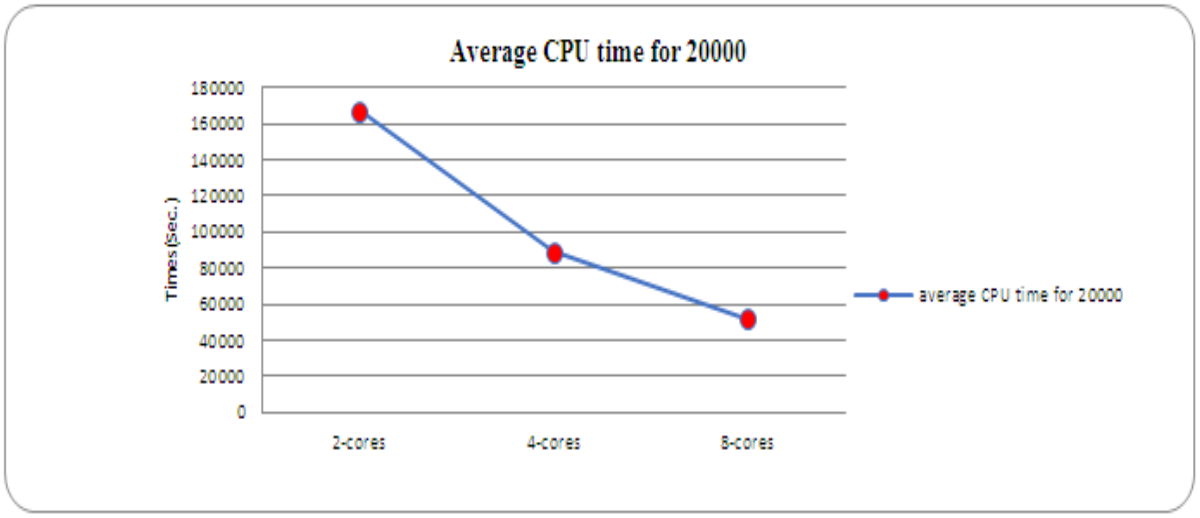


Figure 13. Average CPU time using CPU with selecting 2-cores, 4-cores and 8-cores for order 20000



Figures 15 and 16 represent the overall CPU-usage value for all selected cores of the system. The overall CPU usage for two cores is equal to 24% because each core participates by equal ratio which is 12%; the ratio of each core is equal because of the equality division of the CPU-usage of the processor. But with increasing the number of selection core to 4-core and to 8-core the overall CPU-usage will be increasing to 48 and 96, which means the value of overall CPU usage increased with the increasing number of selection cores (there is a positive relationship).

Figure 14. Overall CPU usage for orders 500 and 4000 using CPU with selecting 2, 4 and 8 cores

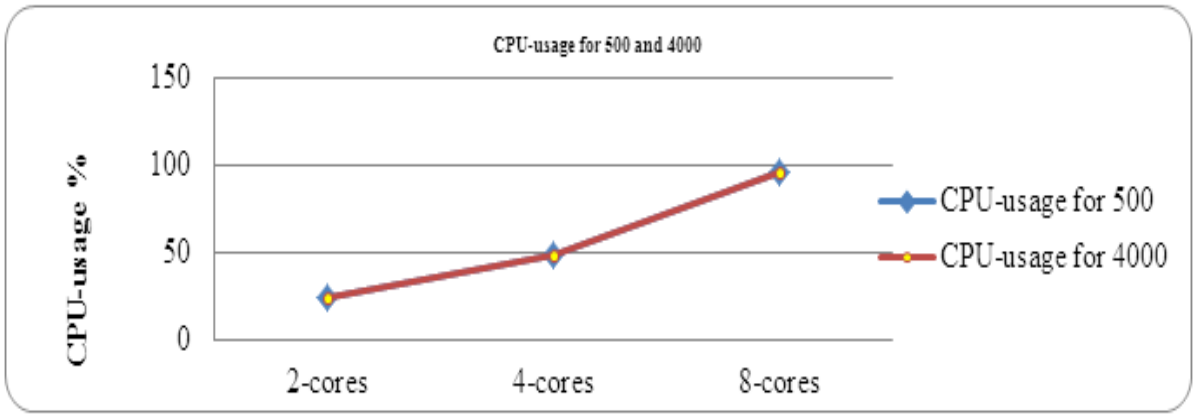


Figure 15. Overall CPU usage for orders 8000 and 10000 using CPU with selecting 2, 4 and 8 cores

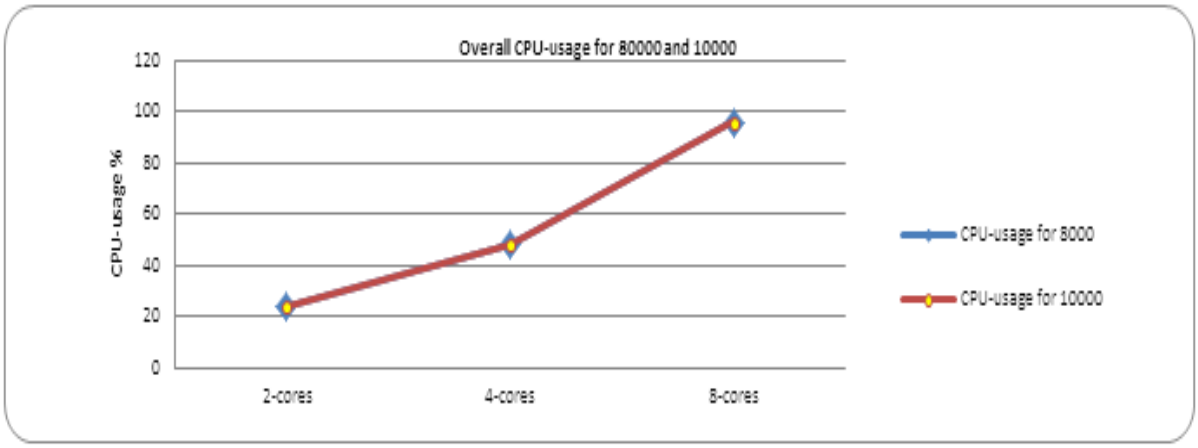
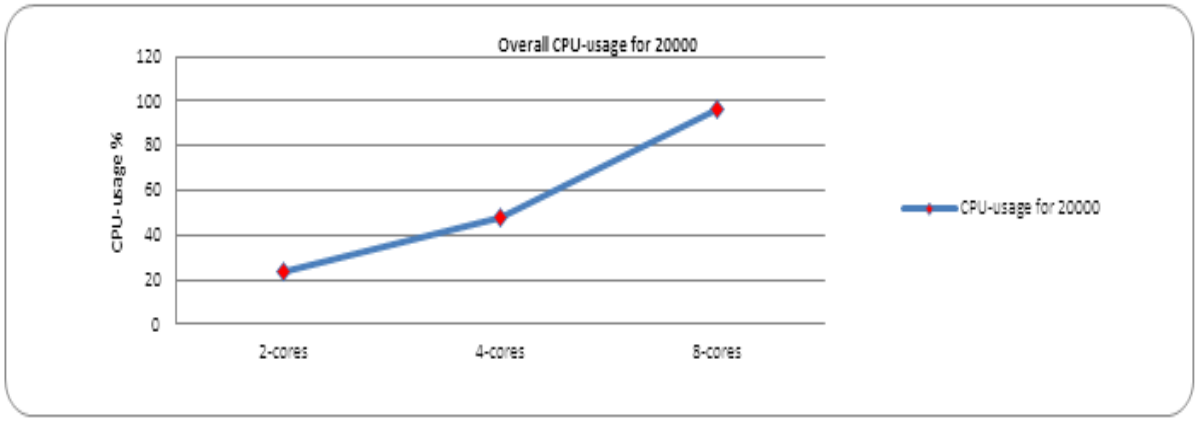


Figure 16. Overall CPU usage for orders (20000) using CPU with selecting (2, 4 and 8) cores



Conclusion

In this paper the effect of the multi-core processor system addressed, depends on client/servers' principles with a network consisting of two nodes; one of them is a client and the other is a server. The type of the server was a multi-core processor. Improved system performance was implemented and designed parallel application used the client/server principle; the workload was divided among multiple cores at the server side such that the amount of threads created is equal to the number of cores in the system. Algorithm in this work is capable of distributing the workload among all selected cores and keeps the utilisation of all processor cores on average. In this work the results showed that the effects of parallel processing appear clearly on the processing time with increasing the load. But its effects did not appear well with a small load application. Also, the results showed the time would be reduced by increasing the number of selection cores.

REFERENCE

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.

Brifcani, A. M. A., & Al-Bamerny, J. N. (2010). *Image compression analysis using multistage vector quantization based on discrete wavelet transform.* Paper presented at the 2010 International Conference on Methods and Models in Computer Science (ICM2CS-2010).

Chowdhury, R. (2010). Parallel Computing with OpenMP to solve matrix Multiplication. *Uconn biogrid reu Summer*.

Grant, R. E., & Afsahi, A. (2007). Analysis and improvement of performance and power consumption of chip multi-threading SMP architectures.

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.

Hassan, O. M. S., Abdulazeez, A. M., & TİRYAKİ, V. M. (2018). *Gait-based human gender classification using lifting 5/3 wavelet and principal component analysis.* Paper presented at the 2018 International Conference on Advanced Science and Engineering (ICOASE).

Hemmendinger, D., Ralston, A., Reilly, D., & Maffeis, S. (1998). Client/server term definition. *1998 International Thomson Computer Publishing*.

Mahdi, O. A., Al-Mayouf, Y. R. B., Ghazi, A. B., Wahab, A. W. A., & Idris, M. (2018). An Energy-Aware and Load-balancing Routing Scheme for Wireless Sensor Networks. *Indonesian Journal of Electrical Engineering and Computer Science, 12*(3), 1312-1319.

Mohr, B. (2006). Introduction to parallel computing. *Multiscale Simulation Methods in Molecular Sciences Lecture Notes*, 535.

Nanehkaran, Y. A., & Ahmadi, S. B. B. (2013). The challenges of multi-core processor. *International Journal of Advancements in Research & Technology, 2*(6), 36-39.

Rashid, Z. N., Sharif, K. H., & Zeebaree, S. (2018). Client/Servers Clustering Effects on CPU Execution-Time, CPU Usage and CPU Idle Depending on Activities of Parallel-Processing-Technique Operations ". *INTERNATIONAL JOURNAL OF SCIENTIFIC & TECHNOLOGY RESEARCH, 7*(8), 106-111.

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).

Zeebaree, A.-Z. S. R., Adel, A., Jacksi, K., & Selamat, A. Designing an ontology of E-learning system for duhok polytechnic university using protégé OWL tool. *J. Adv. Res. Dyn. Control Syst., vol, 11*, 24-37.

Zeebaree, D. Q., Haron, H., Abdulazeez, A. M., & Zebari, D. A. (2019). *Trainable Model Based on New Uniform LBP Feature to Identify the Risk of the Breast Cancer.* Paper presented at the 2019 International Conference on Advanced Science and Engineering (ICOASE).

Zeebaree, D. Q., Haron, H., Abdulazeez, A. M., & Zeebaree, S. R. (2017). Combination of K-means clustering with Genetic Algorithm: A review. *International Journal of Applied Engineering Research, 12*(24), 14238-14245.