IMPROVED APPROACH FOR UNBALANCED LOAD-DIVISION OPERATIONS IMPLEMENTATION ON HYBRID PARALLEL PROCESSING SYSTEMS

Subhi **R**afeeq **M**ohammed **Z**ebari[1] and **A**shur **S**argon **Y**owakib[2]
[1] Dept. IT, Akre Technical Inst., Duhok Polytechnic University, Duhok, Iraq.
[2] Dept. Basic Science, Faculty of Agricultural and Forestry, University of Duhok, Duhok, Iraq.

(Accepted for publication: June 9, 2011)

ABSTRACT

The modern computer-systems designed according to multiprocessor configurations. Multiple processors enable multiple threads to be executed simultaneously with the ability of executing the threads of the same process to be run on different processors at the same time. This paper addresses the building of a software application to be implemented on hybrid memory systems depending on client/server principles, the network can contain any number of nodes; one of them is a client and the others are servers.

An improved approach was produced for problem subdivision based on an unbalanced load division case study (Matrix multiplication). Many previous drawbacks overcame, such as matrix-size limitation, effect of multi-core with distributed systems and forcing the processes and threads among multi-core system processors. Thus, the communication-direction from client-side toward the servers-side and vice-versa became more powerful by binding the activities of both Massage-Passing-Interface (MPI) with those of Open Multi-Processing (OpenMP). The proposed algorithms are executed by Quasar Toolkit (QT) creator application using C++ and QT library. The application-software is implemented to get high speed with as possible as minimum time and detect the effects of this system on the CPU Execution time and CPU Usage, the results are very acceptable and the processing time is decreased by 5.4492 times comparing with those without using hybrid parallel processing.

KEYWORDS: *Parallel Processing, Parallel Programming, Client/Server, Clustering, MPI, OpenMP, CPU Execution Time, CPU Usage.*

1. INTRODUCTION

The last decade has witnessed a dramatic increase in computing, networking, and storage technologies. Dynamically adaptive techniques are being widely used to address the intrinsic heterogeneity and high dynamism of the phenomena modeled by these applications. The increasing complexity, dynamism, and heterogeneity of these applications coupled with similarly complex and heterogeneous parallel and distributed computing systems have led to the development and deployment of advanced computational infrastructures that provide programming, execution, and runtime management support for such large-scale adaptive implementations [*Manish 2010*].

Today scientists who wish to write efficient parallel software for high performance systems have to face a highly hierarchical system design, even (or especially) on “commodity” clusters. Parallel programming models on hybrid platforms are: Pure MPI, Hybrid master only, Hybrid with overlap, Pure OpenMP on clusters, and Mapping with fully hybrid MPI+OpenMP [*Georg 2009*].

In message passing paradigm, several separate processes used to complete the overall computation. Many concurrent processes are created, and all of the data involved in the calculation is distributed among them using different ways. There is no shared data; when a process needs data held by another one, the second process must send it to the first process. An MPI message passing protocol describes the internal methods and policies an MPI implementation employs to accomplish message delivery. There are two common message passing protocols, eager and rendezvous. Eager protocol is an asynchronous protocol that allows a send operation to complete without acknowledgement from a matching receives. Rendezvous protocol is a synchronous protocol which requires an acknowledgement from a matching receive in order to complete the send operation. Since MPI enables the programmer to control both of data distribution and process synchronization, problem decomposition and inter process communication represent two challenges in writing MPI parallel programs. Unless they are coded carefully, program performance will be negatively affected [*Alaa* 2011].

When running an OpenMP program on a Non-Uniform Memory Access (NUMA) node, data is normally distributed using “first touch”, the data will reside in the memory of the socket where the data is first used. The primary method of parallelization in OpenMP is the parallelization of loops. Loop iterations are scheduled for execution among threads according to the scheduling method specified in the OpenMP program, static, dynamic or guided (a “chunk-size” can be specified for each of these options). When an OpenMP program is executed on more than one socket, using dynamic or guided schedule will lead to the data distribution performance problem due to non-local data accesses. When an OpenMP program is executed on a single socket instead of on multiple sockets, then the data distribution performance problem goes away and one is free to use any of the scheduling options without a performance penalty due to non-local data accesses [*Glenn* 2010].

Hybrid (MPI/OpenMP) programming is a great way to take advantage from clusters of Symmetric Multi-Processing (SMP) architectures; using MPI across nodes and OpenMP within nodes this will provide good usage of shared memory system resource (memory, latency, and bandwidth), reduce the communication overhead by eliminating MPI communication within node, OpenMP adds fine granularity (larger message sizes) and allows flexibility of dynamic load balancing, lower memory latency and data movement within node, automatic coherency at node, some problems have two-level parallelism naturally, some problems could only use restricted number of MPI tasks, and could have better scalability than both pure MPI and pure OpenMP [*Byoung* 2009].

In order to address the proposed algorithms depended in this paper, it is recommended to produce a review to the related works, which are:

Wesley M. Eddy and Mark Allman 2000, produced an experiment to show that it is possible to use several computers in parallel to solve the problems that take long periods of time to complete on a single machine, and that by using more computers the total calculation time can be drastically reduced. *Gregory O. Khanlarov, etal. 2000*, a new algorithm addressed with two levels parallelization for direct simulation to solve unsteady problems of molecular gas-dynamics on shared and hybrid memory multiprocessor computers.

ROBERT GRANAT, etal. 2009, presented a novel variant of the parallel QR algorithm for solving dense non symmetric eigenvalue problems on hybrid distributed high performance computing (HPC) systems. *Numan O. Yaseen 2010*, addressed distributed memory system depends on client/servers principles. He improved an approach for problem subdivision and design flexible algorithms to communicate efficiently between client-side and servers-side in the way to overcome the problems of hardware networking components and message passing problems. *Farah H. Asaad 2011*, built an application algorithm for implementing the principles of parallel processing using shared memory system to reduce the execution time gradually by increasing number of cores. *Zryan N. Rashid 2012*, addressed distributed memory system depending on client/servers principles. His work addressed an improved approach for problem subdivision and design flexible algorithms to communicate efficiently between client-side and servers-side in the way to overcome the problems of hardware networking components and message passing problems.

However, from the above survey it is clear that there are two main trends of solving the complex problems (either depending on distributed-memory systems or on shared-memory systems). There are few works trend toward assembling these two approaches to get the benefits of them and produce more powerful systems having high ability to treat with heavy loads like the first three survey works, in general these works deal with approaches far from that depended here. This paper trends to assemble the two systems on one that is called hybrid-memory system with clear algorithms and a famous application (Matrix Algebra case-study) is applied here to illustrate the advantages of this approach from the two other types. All of the above related works are important to this work, but the last three works are more near to it and especially the last one of them. An important problem was overcame here which is the treating with as big as possible of matrix-order in the way to get algorithms more flexibility to handle heavy loads, in this paper the matrix-order increased up to (45,000), and the matrices depended here are of square type that produce the ability of manipulate (2,025,000,000 elements) for each matrix. Also, because of that today's computers are of multi-core type, so, the speed of processing will increase rapidly by using many computers with many cores in each one, hence all of these CPUs will participate with the processing in parallel and consequently providing more increased speed of processing.

2. HYBRID PARALLEL PROCESSING

A lot of research has been invested into the implementation of distributed virtual shared memory software which allows near-shared-memory programming on distributed memory parallel machines, notably clusters. Since 2006 Intel offers the “Cluster OpenMP” compiler add-on, enabling the use of OpenMP (with minor restrictions) across the nodes of a cluster. Therefore, OpenMP has literally become a possible programming model for those machines. It is, to some extent, a hybrid model, being identical to plain OpenMP inside a shared-memory node but employing a sophisticated protocol that keeps “shared” memory pages coherent between nodes at explicit or automatic OpenMP flush points. With Cluster OpenMP, frequent page synchronization or erratic access patterns to shared data must be avoided by all means. If this is not possible, communication can potentially become much more expensive than with plain MPI [*Georg 2009*].

Client/Server network uses a network operating system designed to manage the entire network from a centralized point, which is the server. Clients make requests of the server and the server responds with the information or access to a resource. Client/Server networks have some definite advantages over peer-to-peer networks. It is easier to find files and resources because they are stored on the server. Also have much tighter security. All usernames and passwords are stored in the same database (on the server), and individual users can’t use the server as a workstation. The server holds the database of user accounts, passwords, and access rights [*May* 2009]. In cluster sampling, cluster is a group of population elements, constitutes the sampling unit, instead of a single element of the population. The main reason for cluster sampling is “cost efficiency” (economy and feasibility) [*Saifuddin* 2009]. Clustering analysis has been the most popular approach in point data analysis in data-rich environments. It has been actively used in extracting useful information from geospatial point dataset. It answers where and what objects are aggregated. However, it lacks the ability to provide answers for why clusters are there [*Yang* 2011].

3. PROPOSED ALGORITHMS AND STRUCTURE OF HYBRID-MEMORY PARALLEL-PROCESSING SYSTEM

In this paper, the structure of the depended hybrid system consists of two parts (Hardware and Software). In general, any hybrid parallel-processing system is constructed on two main sides (Client and Servers). In this paper; there is no need to more than one computer at the Client-Side, and two computers are depended to be at servers-side, but these computers have different properties in all their characteristics in order to overcome the problem of adding more computers, so any number of computers can be added to the system at the servers-side.

3.1 Hardware part

Hybrid system can be seen as a group of cooperating devices; it may be the one that is responsible for coordinating the whole system and ensuring it works as intended. Figure (1) illustrates the hardware part which has been selected and adopted according to the following steps:

1. The hardware part constructed of client-side and servers-side, the network that contains both sides designed according to star topology.
2. In such work the properties of computers are important; either these properties will be deferent from one computer to another, or they will be the same, which means having identical computers. In fact, in real life providing similar computer (with identical properties) cannot be provided always. So, it has been in this work relied on computers with different specifications in terms of hardware.
3. Client-side has only one host, which controls the sending of message-passing operations to other side.
4. Client-host contains the main program that can treat with all servers-hosts individually, subgroups, or all of them.
5. The secondary storage of the client-host contains the original data of related case-study that must be sent to servers-side, and the receiving results that calculated by the servers-side.
6. Servers-side consists of (2) hosts connected in a way to get a cluster of (1*8 + 1*2) processors.
7. Each server-host contains a program that has the ability to receive data, make the required processing, calculate the results, and send them to the client-side.

8. Servers-side can store (the received data and the determined results) on their secondary storages, or sending them directly to client-side.

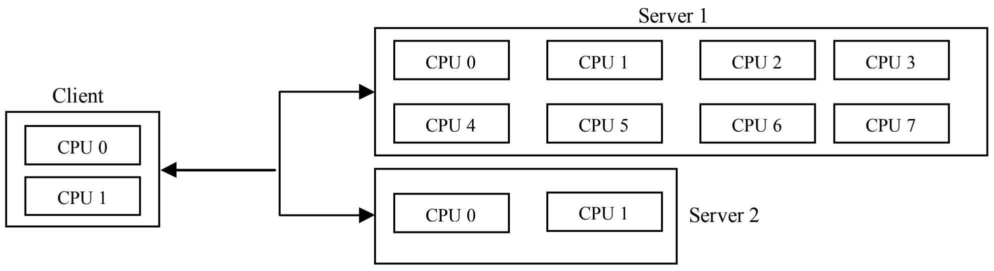


Figure (1): General Hardware-Structure of Depended Hybrid-Memory System

3.2 Software Part

The efficient use of a hybrid computer for the solution of a wide variety of problems requires that the computer hardware be complemented by a comprehensive software package. Such software ought to assist not only in the preparation of programs, but also in their execution as well. It is important that a widely-understood high-level procedure-oriented language be made available to users. Expansion of a purely digital programming language into a hybrid programming language may be accomplished either by rewriting the processor or by attaching a suitable subroutine library.

<u>Client-side software</u>:

Figure (2) represents the general structure of Client-side-software, which is responsible for the following tasks:

1. Detecting number of connected server-hosts at servers-side.
2. Detecting number of connected server-LPs at servers-side.
3. Deciding how many server-hosts will receive the messages from the client-side.
4. Deciding how many server-LPs within each server will receive the messages from the client-side.
5. Deciding which LPs will receive the messages from the client-side.
6. Sending control-messages to server-LPs.
7. Sending related data (as message-text or as data-files) to server-hosts.
8. Monitoring all related server-LPs in case if they send any results or any query-messages.
9. Responding the query-messages received from other servers-side.
10. Receiving the calculated results by server-LPs and accumulating them to get the final results.
11. Making sure that all sending or receiving messages and data are stored on the Client-side secondary storages.

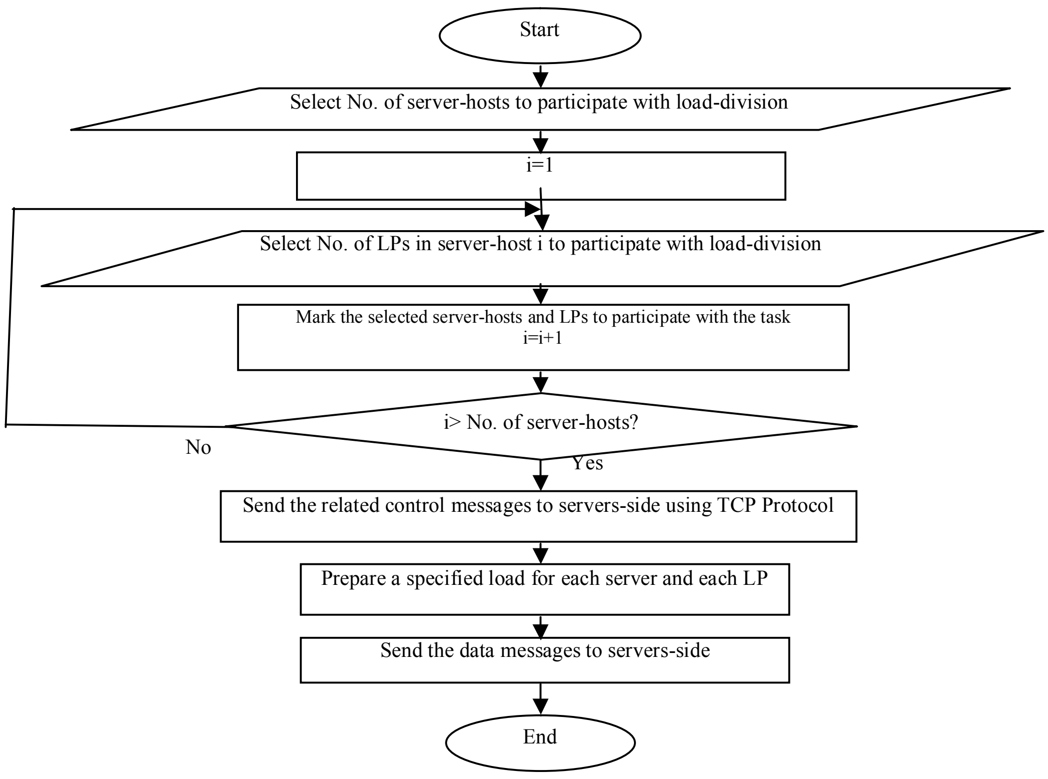

Start
Select No. of server-hosts to participate with load-division
i=1
Select No. of LPs in server-host i to participate with load-division
Mark the selected server-hosts and LPs to participate with the task
i=i+1
i> No. of server-hosts?
No
Yes
Send the related control messages to servers-side using TCP Protocol
Prepare a specified load for each server and each LP
Send the data messages to servers-side
End

Figure (2): Flowchart of General structure of Client-side-Software

Servers-side Software:

Servers-side-software, as shown in Figure (3) represents the programs that service the commands issued from the main program (client program). The software at each server-host is responsible for the following tasks:

1. Detecting the connection status of the client-host.
2. Deciding to work according to the number of server-hosts sent by the client, taking into consideration that it may be out of work for certain numbers of server-hosts, for example; if number of server-hosts is 1, then only server-hosts (1 or 2) will work.
3. Receiving the control-messages from client-host and guide the execution of the server-program to apply the client-requirements.
4. Receiving the related data (as message-text or as data-files) from client-host.
5. Monitoring client-host in case if it sends any immediate command, message, or data.
6. Run the appropriate-subroutines according to the requirements of client-host and calculate the correct results, knowing that each server-host will treat with that part of data that selected for it by the client. And internally within each server-host; the data will be divided among its LPs according to the Client requirements.
7. Sending the calculated results to client-host, knowing that these results will be arranged in a form to be managed by the client-host in a suitable manner.

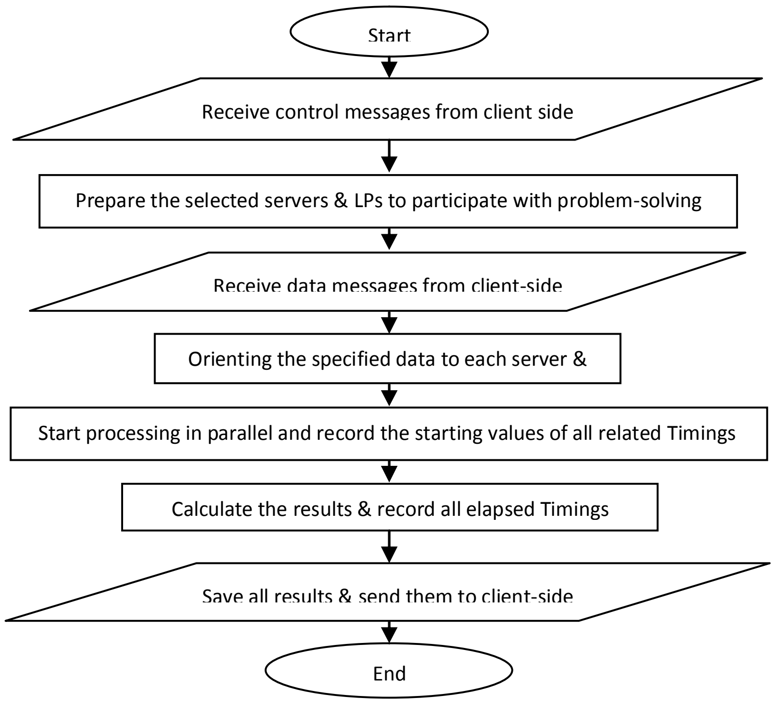

Start
Receive control messages from client side
Prepare the selected servers & LPs to participate with problem-solving
Receive data messages from client-side
Orienting the specified data to each server &
Start processing in parallel and record the starting values of all related Timings
Calculate the results & record all elapsed Timings
Save all results & send them to client-side
End

Figure (3): Flowchart of General Structure of Servers-Side-Software

Each server-program contains all subroutines of the same case study. This gives the server-program the ability to treat with any selected part of data and chose the appropriate subroutine to calculate the required results.

3.3 Messages Transferred Between Client-side and Servers-side

There are two types of messages related to hybrid systems parallel processing approaches which are (control- messages and data-messages).

Data Messages

Data messages; issued by client-side and/or servers-side. These messages carry specific data, which help running processes at server-processors if the messages are issued by client-host. Also, may be representing specific results if the messages issued by server-hosts as illustrated in Figure (4).

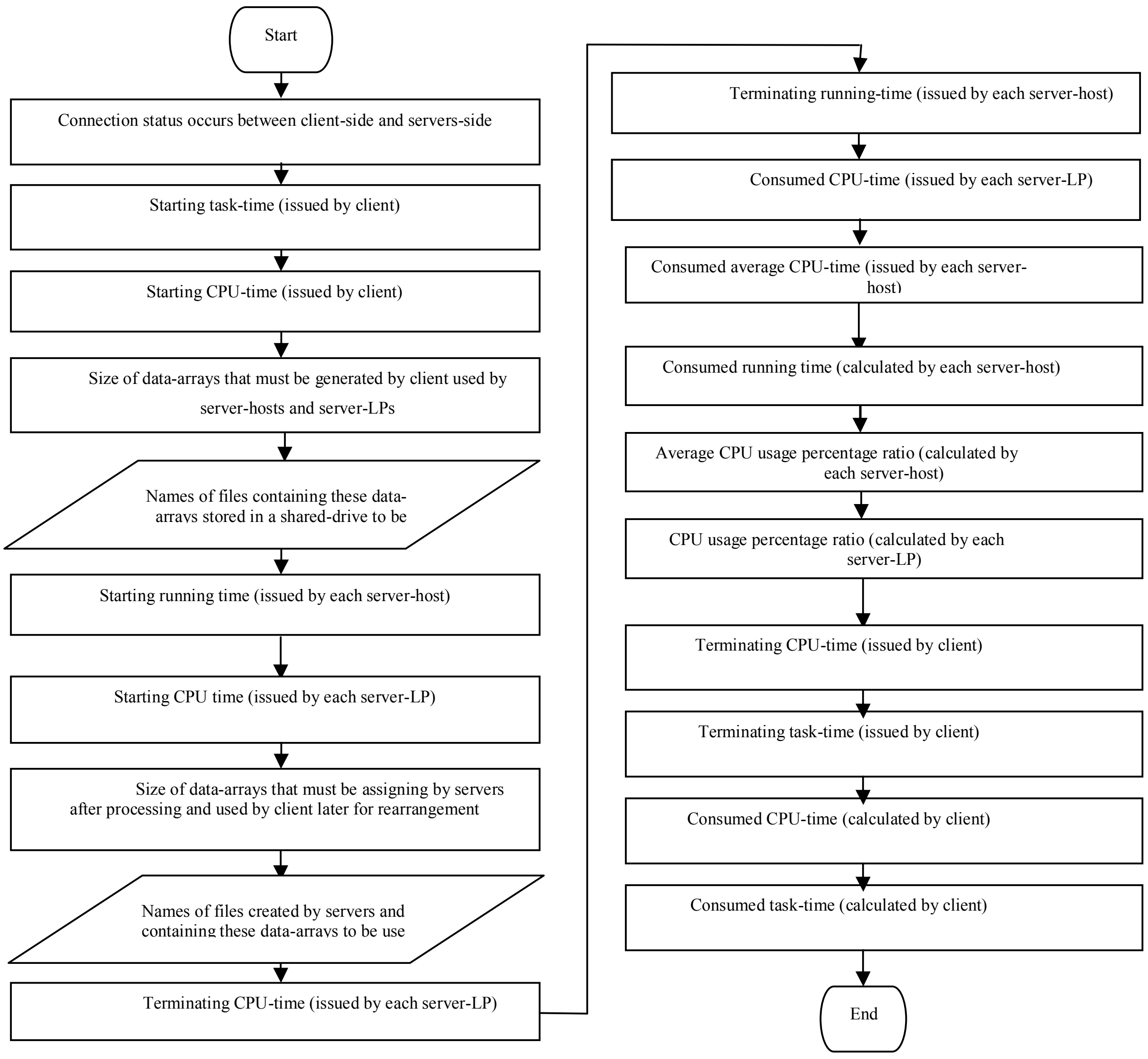

Start
Connection status occurs between client-side and servers-side
Starting task-time (issued by client)
Starting CPU-time (issued by client)
Size of data-arrays that must be generated by client used by server-hosts and server-LPs
Names of files containing these data-arrays stored in a shared-drive to be
Starting running time (issued by each server-host)
Starting CPU time (issued by each server-LP)
Size of data-arrays that must be assigning by servers after processing and used by client later for rearrangement
Names of files created by servers and containing these data-arrays to be use
Terminating CPU-time (issued by each server-LP)
Terminating running-time (issued by each server-host)
Consumed CPU-time (issued by each server-LP)
Consumed average CPU-time (issued by each server-host)
Consumed running time (calculated by each server-host)
Average CPU usage percentage ratio (calculated by each server-host)
CPU usage percentage ratio (calculated by each server-LP)
Terminating CPU-time (issued by client)
Terminating task-time (issued by client)
Consumed CPU-time (calculated by client)
Consumed task-time (calculated by client)
End

Figure (4): Flowchart of Data-Messages

This work uses the following data messages:

1. Starting task time (issued by client).
2. Starting CPU time (issued by client).
3. Size of data-arrays that must be generated by client used by server-hosts and server-LPs.
4. Names of files containing these data-arrays stored in a shared-drive to be used by both client-side and servers-side.
5. Starting running time (issued by each server-host).
6. CPU time (issued by each server-LP).
7. Size of data-arrays that must be assigning by servers after processing and used by client later for rearrangement.
8. Names of files created by servers and containing these data-arrays to be used by client-side later for rearrangement.
9. Terminating CPU time (issued by each server-LP).
10. Terminating running time (issued by each server-host).
11. Consumed average CPU time (issued by each server-host).
12. Consumed CPU time (issued by each server-LP).
13. Consumed running time (calculated by each server-host).
14. Average CPU usage percentage ratio (calculated by each server-host).
15. CPU usage percentage ratio (calculated by each server-LP).
16. Terminating CPU time (issued by client).
17. Terminating task time (issued by client).
18. Consumed CPU time (calculated by client).
19. Consumed task time (calculated by client).

Control Messages:

Control messages issued by client-host and sent to server-hosts and in turn to their LPs. These messages control the management of the processing overall the network and monitor the performance of the hosts especially servers-hosts and their LPs as shown in Figure (5). The current study uses the following control messages:

1. Connection status of each server-host, whether it is ready or not.
2. Selecting number of server-hosts to participate in the task.
3. Selecting number of server-LPs of each selected server to participate in the task.
4. Selecting and/or deselecting any server-host or any server-LP to be ready for communication with client-side.
5. Sending the starting-signal and/or termination-signal to any selected server.

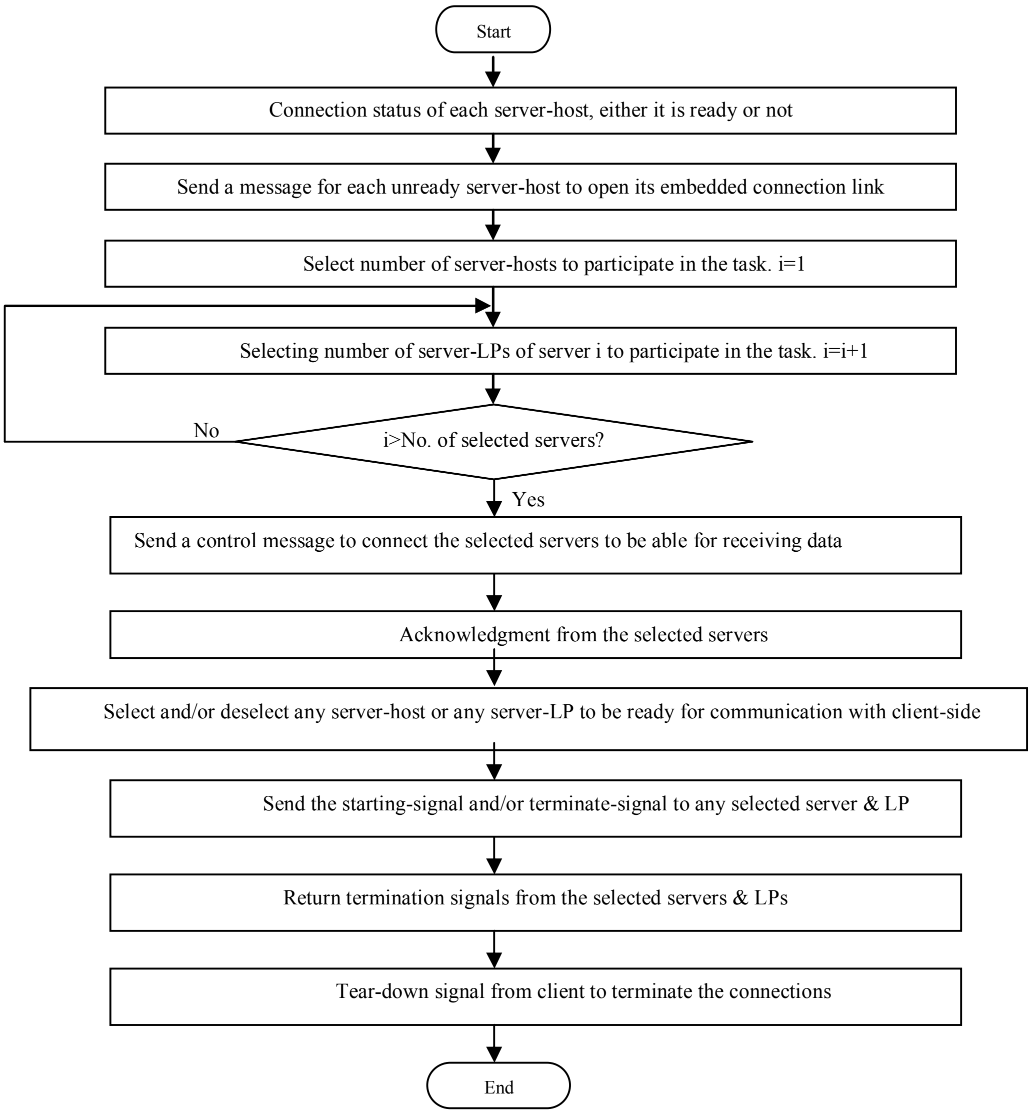

Start
Connection status of each server-host, either it is ready or not
Send a message for each unready server-host to open its embedded connection link
Select number of server-hosts to participate in the task. i=1
Selecting number of server-LPs of server i to participate in the task. i=i+1
i>No. of selected servers?
No
Yes
Send a control message to connect the selected servers to be able for receiving data
Acknowledgment from the selected servers
Select and/or deselect any server-host or any server-LP to be ready for communication with client-side
Send the starting-signal and/or terminate-signal to any selected server & LP
Return termination signals from the selected servers & LPs
Tear-down signal from client to terminate the connections
End

Figure (5): Flowchart of Control-Messages

1. CASE STUDY: (UNBALANCED LOAD-DIVISION) MATRIX MULTIPLICATION

Using one client and two servers, server-1 has 8 LPs and server-2 has 2 LPs. There are 10 square matrices must be multiplied by other 10 matrices to obtain 10 resultant matrices. These operations will be repeated for different orders to illustrate the effects of hybrid parallel processing approach with increasing the load. Tables (1 and 2) represent distribution of the load using (one server and two servers) respectively.

[E1], [E2],..., [E10]
[F1], [F2],..., [F10]
[M1], [M2],..., [M10]
$[M_i]= [E_i]*[F_i]$
i=1, 2, ..., 10

Table (1): Ten matrices on server-1

Server 1 No. of LP(s)	LP 0	LP 1	LP 2	LP 3	LP 4	LP 5	LP 6	LP 7
1	**10**							
2	**5**	**5**						
3	**3**	**3**	**4**					
4	**2**	**2**	**3**	**3**				
5	**2**	**2**	**2**	**2**	**2**			
6	**1**	**1**	**2**	**2**	**2**	**2**		
7	**1**	**1**	**1**	**1**	**2**	**2**	**2**	
8	**1**	**1**	**1**	**1**	**1**	**1**	**2**	**2**

Table (2): Eight matrices on server-1 and two matrices on server-2

Server 1 No. of LP(s)	LP 0	LP 1	LP 2	LP 3	LP 4	LP 5	LP 6	LP 7	Server 2 No. of LP(s)	LP 0	LP 1
1	**8**								**1**	**2**	
2	**4**	**4**							**2**	**1**	**1**
4	**2**	**2**	**2**	**2**					**2**	**1**	**1**
8	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**2**	**1**	**1**

4.1 RESULTS OF IMPLEMENTING A CASE-STUDY ON THE PROPOSED HYBRID-MEMORY SYSTEM

Depending on the proposed algorithms explained in section 6; a general case study adopted here which deals with unbalanced load-division related with matrix-multiplication operations.

First stage includes information about; number of servers, number of processors, Start and End times for (Real-Consumed-CPU, total execution which called Thread-Total-Execution) with their (Average and Maximum) values, CPUs usages, Client-Waiting-time, and Starting of Matrix-Creation.

In this paper there are three hosts used which are (Client-host: Core 2 Duo with 2 CPUs, server1-host: Core i7 with 8 CPUs, and server2-host: Core 2 Duo with 2 CPUs). Hence, the case study will be applied on these three hosts. The system-information detected from this system shown in Figure (6) which was taken during the running as print-screen image. The properties of these hosts can be summarized as follows:

1. Client-Host: Core 2 Duo with 2 CPUs: frequency of each CPU=2.00 GHz, RAM=2 GB.
2. Server1-Host: Core i7 with 8 CPUs: frequency of each CPU=1.6 GHz, RAM=4 GB.
3. Server2-Host: Core 2 Duo with 2 CPUs: frequency of each CPU=2.2 GHz, RAM=2 GB.

At starting, the user must enter the order of the matrices that will be created by the program and filled with random values of elements. Then, number of servers will be selected, and number of CPUs for each server must be selected. The system is error handling for values related with number of servers and CPUs. The system will start processing by selecting Start option, then the values related with timings and CPU-usage (explained above) will be appeared and recorded in suitable tables to be manipulated and plotted later.

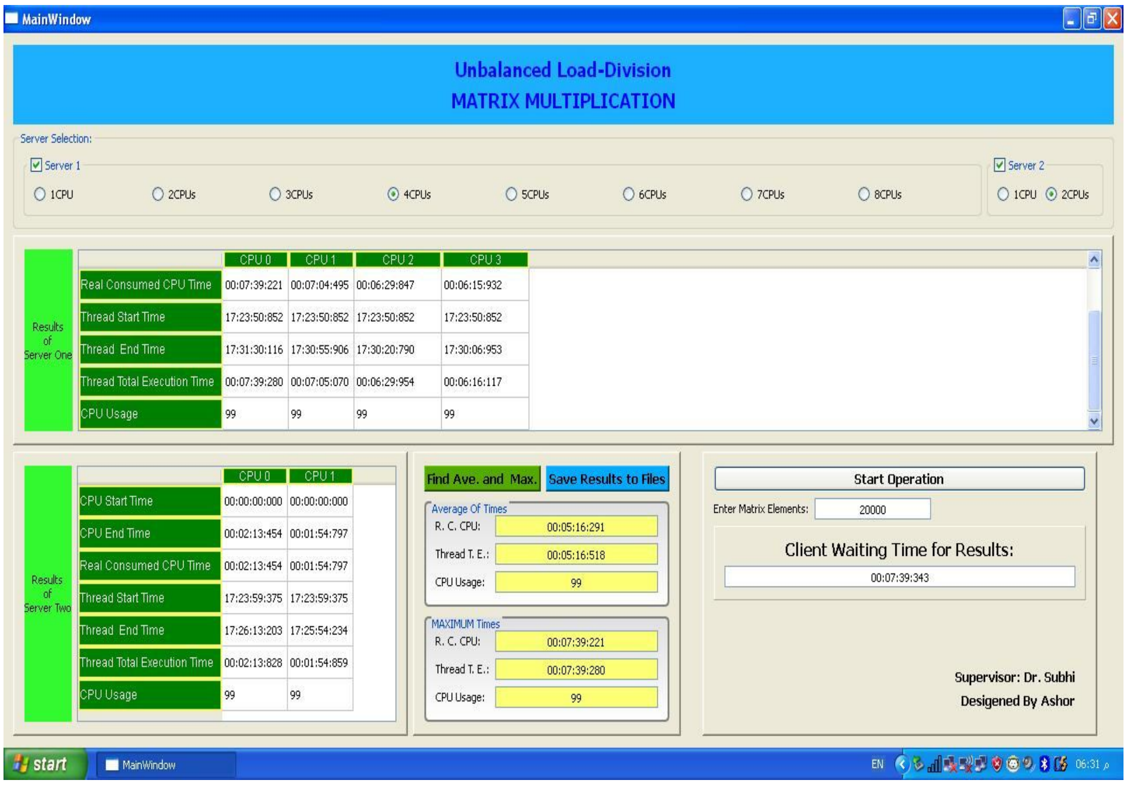
This case study depends on multiplying two square matrices using one client and two servers, server-1 has 8 LPs and server-2 has 2 LPs. There are 10 square matrices with different options of orders (1000, 10000, 20000 and 45000) elements.

Depending on maximum matrix-order (45,000*45,000): there will be need to treat with (2,025,000,000 elements) for each matrix and (20 * 2,025,000,000 = 40,500,000,000 elements) for all of 20 matrices, adding to them the results of the calculated matrices which are 10 matrices and equal (10 * 2,025,000,000 = 20,250,000,000 elements), and as overall there will be (40,500,000,000 + 20,250,000,000 = 60,750,000,000 elements). While for minimum matrix-order (1,000*1,000) as overall there will be (30,000,000 elements).

The results are divided into two main categories (using one-server and using two-servers). These categories can be subdivided into two main groups appeared in Tables (3 to 18) and plotted as shown in Figures (7 to 14).

The first group is related with the average values of consumed CPU-time values for all participated LPs at servers-side, which are acceptable values to be depended. These values are illustrated in Tables (3 to 10), Tables (3 to 6) represent the results when using only one-server while Tables (7 to 10) represent the results when using two-servers. These results are plotted as shown in Figures (7 to 9) for one-server and Figures (10 to 12) for two-servers.

The second group is related with maximum consumed CPU-time values for servers-side. This is an additional assessment of performance of this work in the view of the latest returning results by the servers-side which represents the longest values of consumed CPU-times for all servers as acceptable values to be depended; these values are illustrated in Tables (11 to 18), the results of only order=45000 been selected to be plot as shown in Figure (13) for one-server and Figure (14) for two-servers.



Server Selection:

[✓ Server 1]
1CPU 2CPUs 3CPUs [✓ 4CPUs] 5CPUs 6CPUs 7CPUs 8CPUs

[✓ Server 2]
1CPU [✓ 2CPUs]

Results of Server One

	CPU 0	CPU 1	CPU 2	CPU 3
Real Consumed CPU Time	00:07:39:221	00:07:04:495	00:06:29:847	00:06:15:932
Thread Start Time	17:23:50:852	17:23:50:852	17:23:50:852	17:23:50:852
Thread End Time	17:31:30:116	17:30:55:906	17:30:20:790	17:30:06:953
Thread Total Execution Time	00:07:39:280	00:07:05:070	00:06:29:954	00:06:16:117
CPU Usage	99	99	99	99

Results of Server Two

	CPU 0	CPU 1
CPU Start Time	00:00:00:000	00:00:00:000
CPU End Time	00:02:13:454	00:01:54:797
Real Consumed CPU Time	00:02:13:454	00:01:54:797
Thread Start Time	17:23:59:375	17:23:59:375
Thread End Time	17:26:13:203	17:25:54:234
Thread Total Execution Time	00:02:13:828	00:01:54:859
CPU Usage	99	99

Average Of Times

R. C. CPU:	00:05:16:291
Thread T. E.:	00:05:16:518
CPU Usage:	99

MAXIMUM Times

R. C. CPU:	00:07:39:221
Thread T. E.:	00:07:39:280
CPU Usage:	99

Enter Matrix Elements: 20000

Client Waiting Time for Results: 00:07:39:343

Supervisor: Dr. Subhi
Desigened By Ashor

Figure (6): The GUI of the System during program running, two servers when 4-CPUs of server1 and 2-CPUs of server2 participate with solving the problem with Order of 20000.

Table (3): Average Values of Order 1000(Server: Core i7)

Number of CPU's	Thread Total Execution Time (Second)	Real Consumed CPU Time (Second)	CPU Usage %
1 CPU	0.125	0.125	100
2 CPUs	0.117	0.117	100
3 CPUs	0.078	0.078	100
4 CPUs	0.074	0.074	100
5 CPUs	0.063	0.063	100
6 CPUs	0.058	0.058	100
7 CPUs	0.052	0.052	100
8 CPUs	0.032	0.030	96

Table (4): Average Values of Order 10000(Server: Core i7)

Number of CPU's	Thread Total Execution Time (Second)	Real Consumed CPU Time (Second)	CPU Usage %
1 CPU	113.359	113.350	100
2 CPUs	95.067	95.060	99.5
3 CPUs	56.577	56.571	100
4 CPUs	50.879	50.876	99.75
5 CPUs	43.599	43.593	99.6
6 CPUs	40.753	40.745	99.6667
7 CPUs	31.927	31.918	99.5714
8 CPUs	30.253	30.249	99.875

Table (5): Average Values of Order 20000(Server: Core i7)

Number of CPU's	Thread Total Execution Time (Second)	Real Consumed CPU Time (Second)	CPU Usage %
1 CPU	1058.618	1058.608	99
2 CPUs	923.795	923.790	100
3 CPUs	522.237	522.229	99.3333
4 CPUs	477.931	477.905	99.5
5 CPUs	401.167	401.144	99.4
6 CPUs	376.213	376.189	99.1667
7 CPUs	290.140	290.095	99.4286
8 CPUs	272.356	272.287	99

Table (6): Average Values of Order 45000(Server: Core i7)

Number of CPU's	Thread Total Execution Time (Second)	Real Consumed CPU Time (Second)	CPU Usage %
1 CPU	14718.485	14717.431	99
2 CPUs	12485.178	12484.753	99
3 CPUs	7157.054	7156.776	99
4 CPUs	6053.696	6053.167	99
5 CPUs	5246.346	5245.987	99
6 CPUs	4723.912	4723.238	99
7 CPUs	3740.553	3738.349	99
8 CPUs	3547.192	3541.042	99

Table (7): Average Values of Order 1000
(Servers: Core i7 & Core 2 Duo)

Core i7	Core 2 Duo	Thread Total Execution Time (Second)	Real Consumed CPU Time (Second)	CPU Usage %
1 CPU	1 CPU	0.070	0.070	100
2 CPUs	2 CPUs	0.051	0.051	100
4 CPUs	2 CPUs	0.047	0.047	100
8CPUs	2 CPUs	0.030	0.029	96.8

Table (8): Average Values of Order 10000
(Servers: Core i7 & Core 2 Duo)

Core i7	Core 2 Duo	Thread Total Execution Time (Second)	Real Consumed CPU Time (Second)	CPU Usage %
1 CPU	1 CPU	56.995	56.990	100
2 CPUs	2 CPUs	47.382	47.320	99
4 CPUs	2 CPUs	33.073	33.070	99.8333
8CPUs	2 CPUs	23.539	23.526	99.5

Table (9): Average Values of Order 20000
(Servers: Core i7 & Core 2 Duo)

Core i7	Core 2 Duo	Thread Total Execution Time (Second)	Real Consumed CPU Time (Second)	CPU Usage %
1 CPU	1 CPU	509.485	509.391	99.5
2 CPUs	2 CPUs	433.144	433.107	99.25
4 CPUs	2 CPUs	303.467	303.293	99.3333
8CPUs	2 CPUs	210.175	210.096	99.1

Table (10): Average Values of Order 45000
(Servers: Core i7 & Core 2 Duo)

Core i7	Core 2 Duo	Thread Total Execution Time (Second)	Real Consumed CPU Time (Second)	CPU Usage %
1 CPU	1 CPU	6985.378	6984.444	99
2 CPUs	2 CPUs	5676.693	5676.044	99
4 CPUs	2 CPUs	3952.135	3951.574	99
8CPUs	2 CPUs	2705.737	2700.848	99

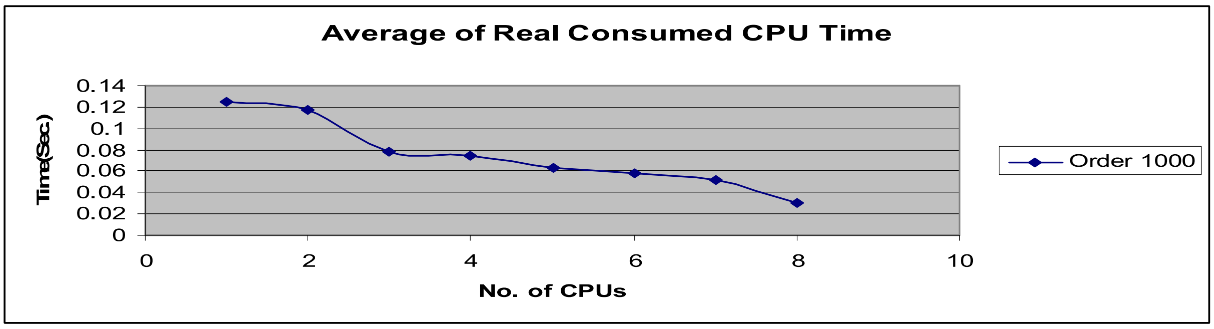


Figure (7): Average of Real consumed CPU time of Order 1000 (Server: Core i7)

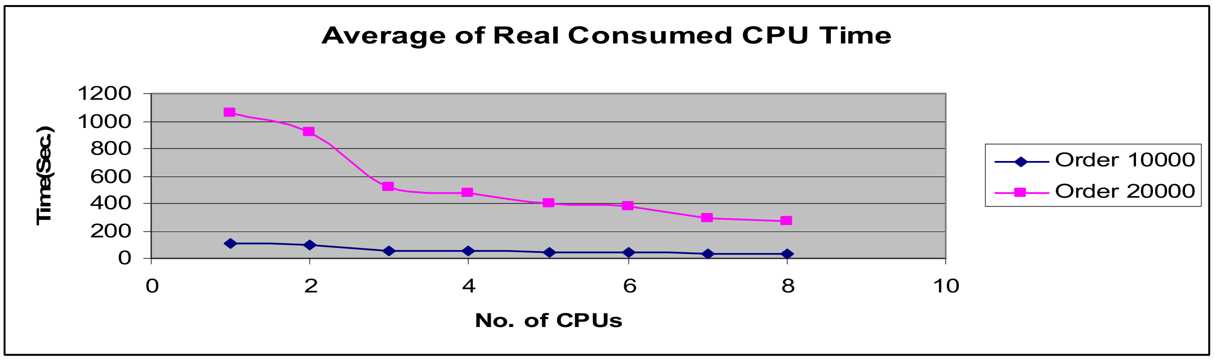


Figure (8): Average of Real consumed CPU time of Orders 10000, 20000 (Server: Core i7)

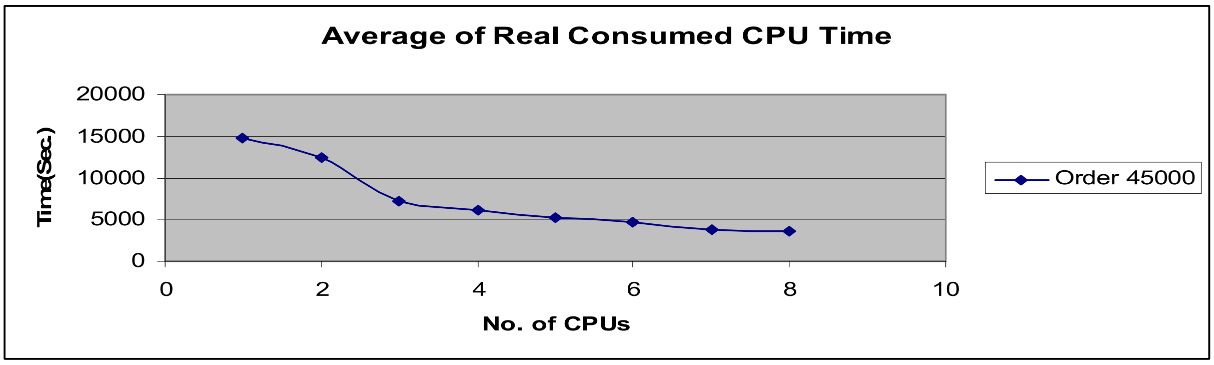


Figure (9): Average of Real consumed CPU time of Order 45000 (Server: Core i7)

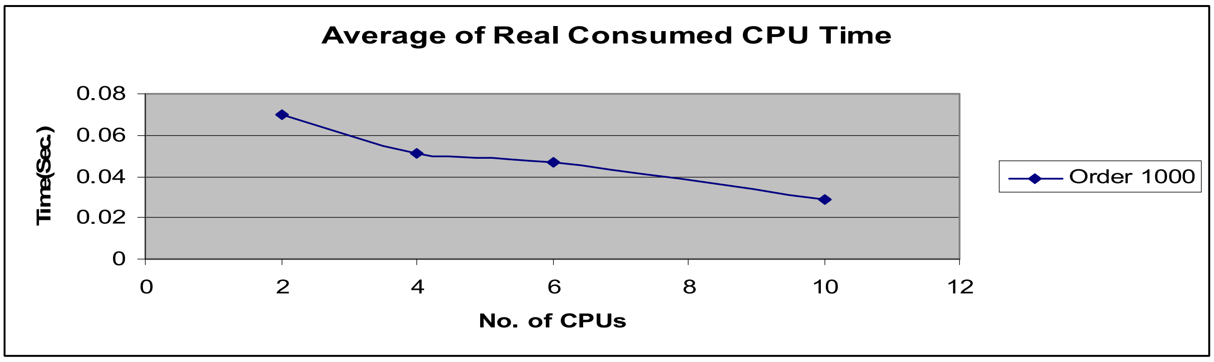


Figure (10): Average of Real consumed CPU time of Order 1000 (Servers: Core i7 & Core 2 Duo)

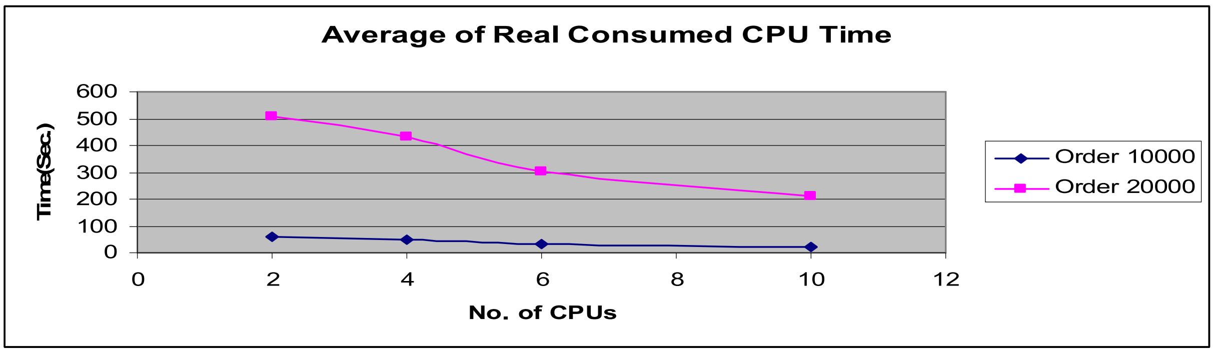


Figure (11): Average of Real consumed CPU time of Orders 10000, 20000 (Servers: Core i7 & Core 2 Duo)

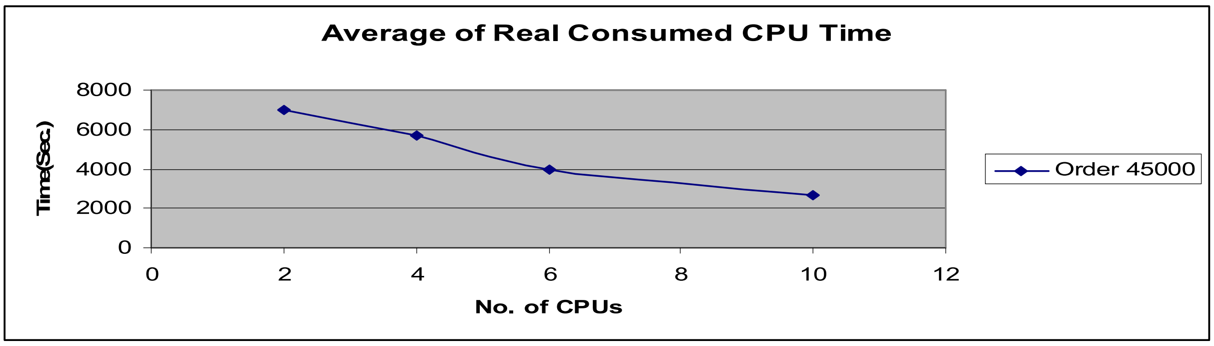


Figure (12): Average of Real consumed CPU time of Order 450000 (Servers: Core i7 & Core 2 Duo)

Table (11): Maximum Values of Order 1000 (Server: Core i7)

Number of CPU's	Client waiting Time (Second)	Thread Total Execution Time (Second)	Real Consumed CPU Time (Second)	CPU Usage %
1 CPU	0.125	0.125	0.125	100
2 CPUs	0.125	0.125	0.125	100
3 CPUs	0.110	0.093	0.093	100
4 CPUs	0.094	0.078	0.078	100
5 CPUs	0.094	0.065	0.065	100
6 CPUs	0.088	0.063	0.063	100
7 CPUs	0.083	0.062	0.062	100
8 CPUs	0.078	0.047	0.047	100

Table (12): Maximum Values of Order 10000 (Server: Core i7)

Number of CPU's	Client waiting Time (Second)	Thread Total Execution Time (Second)	Real Consumed CPU Time (Second)	CPU Usage %
1 CPU	113.375	113.351	113.350	100
2 CPUs	99.344	99.326	99.311	100
3 CPUs	67.391	67.376	67.376	100
4 CPUs	70.375	57.501	57.500	100
5 CPUs	64.625	51.948	51.933	100
6 CPUs	47.531	47.502	47.500	100
7 CPUs	42.063	42.042	42.012	100
8 CPUs	39.594	39.562	39.560	100

Table (13): Maximum Values of Order 20000(Server: Core i7)

Number of CPU's	Client waiting Time (Second)	Thread Total Execution Time (Second)	Real Consumed CPU Time (Second)	CPU Usage %
1 CPU	1058.625	1058.618	1058.608	99
2 CPUs	964.266	964.253	964.250	100
3 CPUs	625.890	625.873	625.861	100
4 CPUs	538.532	538.513	538.485	100
5 CPUs	486.984	486.979	486.970	100
6 CPUs	439.719	439.702	439.674	100
7 CPUs	385.656	385.638	385.632	100
8 CPUs	355.734	355.712	355.683	99

Table (14): Maximum Values of Order 45000(Server: Core i7)

Number of CPU's	Client waiting Time (Second)	Thread Total Execution Time (Second)	Real Consumed CPU Time (Second)	CPU Usage %
1 CPU	14718.657	14718.485	14717.431	99
2 CPUs	13032.422	13032.328	13031.856	99
3 CPUs	8339.187	8338.449	8337.786	99
4 CPUs	6849.469	6849.348	6849.256	99
5 CPUs	6349.313	6349.149	6348.368	99
6 CPUs	5551.062	5550.957	5550.438	99
7 CPUs	4951.937	4951.885	4951.207	99
8 CPUs	4716.141	4716.091	4714.694	99

Table (15): Maximum Values of Order 1000(Servers: Core i7 & Core 2 Duo)

Core i7	Core 2 Duo	Client waiting Time (Second)	Thread Total Execution Time (Second)	Real Consumed CPU Time (Second)	CPU Usage %
1 CPU	1 CPU	0.125	0.109	0.109	100
2 CPUs	2 CPUs	0.094	0.093	0.093	100
4 CPUs	2 CPUs	0.079	0.078	0.078	100
8CPUs	2 CPUs	0.074	0.047	0.047	100

Table (16): Maximum Values of Order 10000(Servers: Core i7 & Core 2 Duo)

Core i7	Core 2 Duo	Client waiting Time (Second)	Thread Total Execution Time (Second)	Real Consumed CPU Time (Second)	CPU Usage %
1 CPU	1 CPU	84.375	84.365	84.360	100
2 CPUs	2 CPUs	85.609	83.195	83.165	99
4 CPUs	2 CPUs	45.813	45.786	45.780	100
8CPUs	2 CPUs	30.016	29.984	29.906	100

Table (17): Maximum Values of Order 20000(Servers: Core i7 & Core 2 Duo)

Core i7	Core 2 Duo	Client waiting Time (Second)	Thread Total Execution Time (Second)	Real Consumed CPU Time (Second)	CPU Usage %
1 CPU	1 CPU	761.875	761.859	761.850	100
2 CPUs	2 CPUs	759.594	759.581	759.569	100
4 CPUs	2 CPUs	434.563	434.539	433.933	100
8CPUs	2 CPUs	273.469	273.453	273.377	100

Table (18): Maximum Values of Order 45000(Servers: Core i7 & Core 2 Duo)

Core i7	Core 2 Duo	Client waiting Time (Second)	Thread Total Execution Time (Second)	Real Consumed CPU Time (Second)	CPU Usage %
1 CPU	1 CPU	10938.813	10937.990	10937.480	99
2 CPUs	2 CPUs	10242.625	10242.547	10242.402	99
4 CPUs	2 CPUs	5684.750	5684.252	5684.100	99
8CPUs	2 CPUs	3510.797	3510.794	3504.079	99

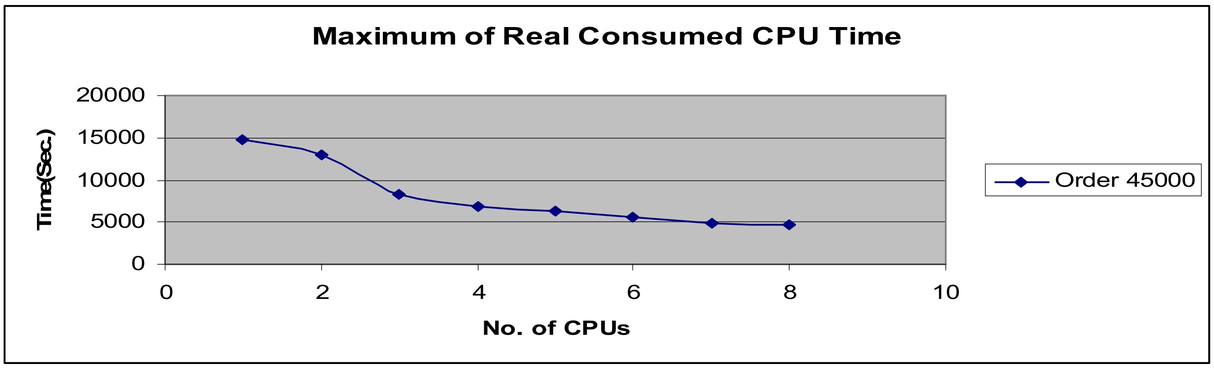


Figure (13): Maximum of Real consumed CPU time of Order 45000 (Server: Core i7)

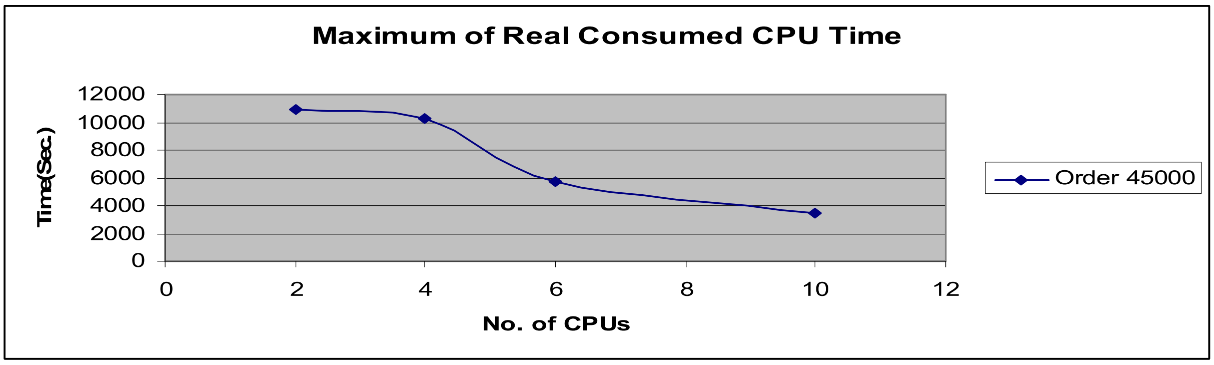


Figure (14): Maximum of Real consumed CPU time of Order 45000 (Servers: Core i7 & Core 2 Duo)

4.2 Discussion

There are results of both (Average and Maximum)-values of Real Consumed CPU for servers-side. Figures (7 to 9 and 10 to 12) illustrate the effects of parallel processing approach on the *average values* of real consumed CPU time when using a hybrid memory system. According to the principles of the parallel processing, the elapsed execution time must be reduced with increasing number of participated CPUs for solving the same problem. This is appeared clearly as shown in the figures, it is shown that the average consumed CPU time in the case of *one-server and one-CPU* have the greatest value (14717.431 seconds), and when using *two-servers with 10-CPUs* have the smallest value (2700.848 seconds). It is clear that the speed of processing increased by (5.4492) times. This is depending on using only two servers and one of them has only 2-CPUs, and this ratio will be increased more when using more than two servers with many-CPUs.

This arrangement of curves (three figures for each timing type) was dependent because of the high-gap of obtained-results among the four orders. These implemented results are in agreement with the principles of parallel processing approaches. So, results of order (1000) plotted on separate figure, results of both orders (10,000 and 20,000) plotted on another figure, and finally results of order (45,000) on separate figure.

As explained above, Figures (13 and 14) illustrate the effects of parallel processing approach on the *maximum values* of real consumed CPU time when using a hybrid memory system. Depending on these results the speed of processing increased by (4.2001) times.

In order to make a comparison between the results of this paper with those of the previous works, there is no exactly such case study has been applied by other previous work using hybrid parallel processing systems. So, the comparison can be applied between the results obtained by one-server using only one-CPU and those of all-servers using all-CPUs as explained above.

5.CONCLUSIONS

The most important points concluded from this paper can be summarized as follows:

This paper produced a complete system with its algorithms for parallel processing operations using hybrid memory system approach.

Depending on most related programming functions that treat directly with CPU, additional enhancements occurred with the proposed algorithms related with; overcoming the limitation of data-size of used matrices with the case study which reached to (45000*45000), and overcoming the problems of servers-side running program complexities to be run automatically which will be more useful and less delay.

Capability of determining and calculating the related timing-values of (program execution and CPU usage) for both (average and maximum) values in high precise.

This application software is implemented successfully on various multi-core systems (such as those having: 2 and 8 cores). And can be implemented on the network with any number of server-hosts, so that each of these hosts has many LPs which provide a very high speed of processing in a parallel manner.

One of the important concluded points here is the ability to forcing the threads into the system in parallel to more than one host and more than one LP at servers-side.

6. References

Alaa Ismail El-Nashar, "To Parallelize or Not to Parallelize, Speed Up Issue", International Journal of Distributed and Parallel Systems (IJDPS) Vol.2, No.2, 2011.

Byoung-Do Kim and John Cazes, "Hybrid Programming on Multi-core, Multi-socket Cluster System", The University of Texas at Austin, Texas advanced computing center, 2009.

Farah H. Asaad, "Shared Memory Performance Analysis on Parallel Processing Applications", MSc Thesis, University of Zakho, Dec. 2011.

Georg Hager, ET.Al, "Communication Characteristics and Hybrid MPI/ OpenMP Parallel Programming on Clusters of Multi-core SMP Nodes", Cray User Group Proceedings, 2009.

Glenn Luecke, ET.Al, "Performance Analysis of Pure MPI versus MPI+OpenMP for Jacobi Iteration and a 3D FFT on the Cray XT5", Cray User Group Proceedings, 2010.

Gregory O. Khanlarov, ET. Al, "Parallel DSMC on Shared and Hybrid Memory Multiprocessor Computers", Springer-Verlag Berlin Heidelberg, 2000.

Manish Parashar and Xiaolin Li, "Advanced Computational Infrastructures for Parallel and Distributed Adaptive Applications", John Wiley & Sons, Inc, 2010.

May P. Zaw and Su Myat M. Soe, "Design and Implementation of Client Server Network Management System for Ethernet LAN", World Academy of Science, Engineering and Technology 48, 2008.

Numan O. Yaseen, " Diagnostic Approach for Improving the Implementation of Parallel Processing Operations ", MSc Thesis, University of Zakho, October 2010.

Robert Granat, ET.Al, "A Novel Parallel QR Algorithm for Hybrid Distributed Memory HPC (High Performance Computing) Systems", Department of Computing Science, UMEA University, Sweden, 2009.

Saifuddin Ahmed, "Cluster Sampling", the Johns Hopkins University, 2009.

Wesley M. Eddy and Mark Allman, "Advantages of Parallel Processing and the Effects of Communications Time", NASA Glenn Research Center Report Number CR-209455, 2000.

Yang Qu, ET.Al, "Cluster Polygonization and Qualitative Cluster Reasoning: Overview", International Journal of Advancements in Computing Technology, Volume 3, Number 3, April, 2011.

Zryan N. Rashid, "Client/Servers Clustering Effects on CPU Execution-Time, CPU Usage and CPU Idle Depending on Activities of Parallel-Processing-Technique Operations", MSc Thesis, University of Sulaimani, Jan. 2012.

أسلوب محسن لتنفيذ عمليات تقسيم-الحمل الغير متوازن على أنظمة المعالجة المتوازية الهجينة

الخلاصة

يتم تصميم أنظمة الكمبيوتر الحديثة وفقا لتكوينات متعددة المعالجات. المعالجات المتعددة تمكن الخيوط المتعددة من أن تنفذ في وقت واحد مع قدرة تنفيذ الخيوط التابعة لنفس العملية ليتم تشغيلها على معالجات مختلفة في نفس الوقت. يتناول هذا البحث بناء برمجيات-تطبيقية التي سيتم تنفيذها على أنظمة هجينة الذاكرة اعتمادا على مبادئ العميل/الخادم، ويمكن أن تحتوي الشبكة على أي عدد من العقد، واحدة منهم هي العميل والبقية هي الخوادم.
تم تقديم أسلوب محسن لدراسة حالة (ضرب المصفوفات) وذلك بتقسيم المشكلة على أساس تجزأة الحمل الغير متوازنة. تم التغلب على كثير من العيوب السابقة، مثل تقييد حجم المصفوفة، تأثير متعددة النوى مع الأنظمة الموزعة وإجبار العمليات والخيوط من بين معالجات الأنظمه متعددة النواة. وهكذا، أصبح أتجاه-الاتصال من جانب العميل تجاه الخوادم والعكس بالعكس أكثر قوة عن طريق ربط واجهة-تمرير-الأشارة (MPI) مع تلك متعدد-المعالجات-المفتوحة (OpenMP). يتم تنفيذ خوارزميات هذه البرمجيات-التطبيقية عن طريق المولد Quasar Toolkit (QT) وتطبيق المستحدث باستخدام مكتبة ++C ومكتبة QT. ويتم تنفيذ البرمجيات-التطبيقية للحصول على سرعة عالية مع الحد الأدنى من الوقت، والكشف عن آثار هذا النظام على وزمن تنفيد وحدة المعالجة المركزية ومدى استخدام (أستغلال) CPU ، النتائج هي مقبولة جداً وتم تقليل زمن المعالجة بمقدار 5.4492 مرة مقارنة مع تلك النتائج في حالة عدم أعتماد المعالجة المتوازية الهجينة.

ڕێکەکا گەشەپێکری بو ئەنجامدانا کارێن دابەشکرنا باری یێ نەراستی یێك لسەر سیستەمێن چارەسەرکرنا هزرا تێکەل

پوخته

سیستەمێن کومپیوتەری یێن نوی تێنە دیزاینکرن لدویڤ پێکهاتنا هەمەجوریا کارا. هەمەجوریا کارا تێلن کو داڤێن هەمەجوری بهێنە ئەنجامدان بئێك کات دگەل شیانا ئەنجامدانا داڤێن ئێك کار دا بهێنە دابەشکرن لسەر جوراوجوریا CPU ب ئێك کات. ئەڤ تویژینە بەحسێ ئاڤاکرنا پروگرامێن راهێنانێ ئەوێن کو بهێنە بکارئینان لسەر سیستەمێن چارەسەرکرنا هزرا تێکەل لسەر بنیاتێن client/server، و چێدبیت تور پێك بهێت ل هەژمارەکا گریا، ئێك ژوان client و یێت مایی serverبن.

ڕێکەکا گەشەپێکری هاتە پێشکێشکرن بۆ نموونا خوێندنێ (Matrix multiplication) ب پارچە پارچەکرنا ئاریشێ لسەر بنیاتێن پارچە پارچەکرنا باری یا هندی ئێك نە، زور کێماسیێن بەرێ هاتنە چارەسەرکرن، وەکو گرێدانا قەبارا Matrix، رویدانا هەمەجورێ ناوك دگەل سیستەمێن دابەشکری و نەچارەکرنا کارا و داڤا بەینا CPU یێن سیستەمێن هەمەجورێ ناوك. و هوسا، بەرێ پەیوەندیا ل لایێ client بۆ لایێ server و بەروڤاژی ب هێزتر دروست کرن ب ڕێکا پەیدابوونا Quasar Toolkit (QT) و راهێنانێن نوی ب ناڤێ کتێبخانێن C++ و QT.

پرۆگرامێن راهێنانێ تێنە بکارئینان ب لەزاتیەکا زور دگەل کێمترین کاتی، و دەستنیشانکرنا رویدانا ئەڤی سیستەمی لسەر کاتێ کارئینا CPU و راددا مفادانێ لـ CPU، ئەنجام یێت زور پەسەندکرینە و دەمێ پرۆسێسکرنێ هاتە کێمکرن ب تەمەتی 5.4492 جاران بەرامبەری وان ئەنجامێت بەردەست کەڤن بێ ڕێکا چارەسەرکرنا هزرا تێکەل.