P- ISSN 1991-8941 E-ISSN 2706-6703 Journal of University of Anbar for Pure Science (JUAPS) Open Access
2011,(5), (3) :50-56

Effects of Parallel Processing Implementation on Balanced Load-Division Depending on Distributed Memory Systems

Subhi R. M. Zebari* Numan O. Yaseen**

*Foundation of Technical Education/Arbil - Amedi Technical Inst.
**Amedi Education Directory.

ARTICLE INFO

Received: 29 / 5 /2011
Accepted: 15 / 11 /2011
Available online: 14/6/2012
DOI: 10.37652/juaps.2011.44313

Keywords:
Parallel Processing Implementation ,
Balanced Load-Division ,
Distributed Memory Systems.

ABSTRACT

Complex problems need long time to be solved, with low efficiency and performance. Therefore, to overcome these drawbacks, the studies went toward the approaches of breaking the problem into independent parts, and treating each part individually in the way that each processing element can execute its part of the problem simultaneously with the others.Parallel processors are computer systems that consist of multiple processing units connected via some interconnection network and the software needed to make the processing units work together. Parallel processing is divided into three types; Shared, Distributed and Hybrid memory systems.In this paper, distributed memory systems addressed depending on client/servers principles, the network can contain any number of nodes; one of them is a client and the others are servers. The algorithms used here are capable of calculating the (Started, Terminated, Consumed -CPU and Total Execution- times and CPU usage) of servers and the Client's -CPU and total execution- times. This work addresses an improved approach for problem subdivision in balanced form 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. We addressed Matrix-Algebra case-study to display the effect of balance load-division for this approach. The obtained results are checked and monitored by special programming-checking-subroutines through many testing-iterations and proved a high degree of accuracy. All of these algorithms implemented using Java Language.

Introduction

To speed-up the execution of a program, the program divided into multiple fragments that can be executed simultaneously, each on its own processor. A program be executed across n processors might execute n times faster than it would using a single processor [1].

A Parallel System is a combination of a parallel algorithm and a machine on which it operates. Both factors count with several variables. Parallel algorithms can be specified using a wide range of models and paradigms. Supporting architectures, even though they all count with more than one processor, they can be different in several dimensions, such as in a control mechanism, address space organization, processors granularity and interconnection network[2].

The main argument for using Multi-Processors (MPs) is to create powerful computers by simply connecting multiple processors.

The MP is expected to reach faster than the fastest single-processor system. In addition, the MP consists of a number of single processors expected to be more cost-effective than building a high-performance single processor. Another advantage of the MP is fault tolerance. If a processor fails, the remaining processors should be able to provide continued service, albeit with degraded performance [3].

Parallel Processing

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 [4].

Uses or applications for PP come from two different areas; on one hand, there are high performance systems for speeding up compute-intense

* Corresponding author at: Foundation of Technical Education/Arbil - Amedi Technical Inst, Iraq.E-mail address: subhizebari@yahoo.com

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 common place today under standard operating systems such as Linux and Windows, with parallel software design becoming more and more important [5]. Although the use of multiple processors can speed−up many operations, most applications cannot yet benefit from parallel processing. Parallel Processing is appropriate only if: (The application has enough parallelism to make a good use of multiple processors. In part, this is a matter of identifying portions of the program that can execute independently and simultaneously on separate processors) [1].

Taxonomy of Computer Architecture

In 1996, Michael J. Flynn created one of the earliest classification systems for parallel (and sequential) computers and programs, now known as Flynn's taxonomy [6]. Flynn’s classification scheme based on the notion of a stream of information. Two types of information flow into a processor: instructions and data. The instruction stream defined as the sequence of instructions performed by the processing unit. The data stream defined as the data traffic exchanged between the memory and the processing unit [3]. Parallel computers are classified along data and instruction axes data stream as bellow [7]:

a. Single Instruction Single Data (SISD).
b. Single Instruction Multiple Data (SIMD).
c. Multiple Instruction Single Data (MISD).
d. Multiple Instruction Multiple Data (MIMD).

Parallel Programming and Implementation

One way to solve a problem fast is to break the problem into pieces, and arrange for all the pieces to be solved simultaneously. The more pieces, the faster the job goes-up to a point where the pieces become too small to make the effort of breaking-up and distributing worth the bother. These simple, obvious observations underlie all of parallel programming. A "parallel program" is a program that uses the breaking-up and handing-out approach to solve large or difficult problems [8]. Parallel programs are intended for execution on many processors simultaneously. Each processor works on one piece of the problem, and they all proceed together. In the best case, n processors focused on a single problem will solve it n times faster than any single processor [8].

Parallel programming is a viable method for solving computationally intensive problems in various fields. In electrical engineering, for instance, solving power systems network equations is an area where parallel algorithms are being developed and applied. A popular approach to implementing parallel algorithms is to employ a cluster or a network of parallel computers. With the advances made in computer hardware and software, it is now quite a simple matter to configure a computer network and program it to solve problems cooperatively. The parallel software simulation is interesting application that has been implemented on a computer cluster. The common programming paradigm for this type of parallel algorithm is either multiple program multiple data or single program multiple data [9]. Because of the importance of improving the performance and the quality of the solutions, researchers have proposed different approaches to parallelize multi-objective evolutionary algorithms [10].

Client/Server Principles

Client/server computing enables the use of low-cost hardware and software, increases local autonomy and ownership of data, and offers better performance and higher availability. It is used to build many different types of application, from corporate distributed online transaction processing and data warehousing applications, to departmental and groupware systems. These applications often involve a wide range of different hardware and software; powerful machines, departmental servers, desktop systems, or even network computers. Some people argue that the advent of Internet/Intranet and Web technology signal the demise of client/server, but closer examination shows that this technology is just another form of client/server computing. In fact, many organizations are interested in connecting new Web-based applications to existing client/server systems. Client/server, therefore, is likely to be with us for some time to come, and will be used for developing an ever-increasing set of complex and interconnected applications [11].

Early implementers of client/server applications focused primarily on fast application development, and on the cost savings provided by the use of cheaper hardware and software. The first client/server systems paid little attention to good architecture design, systems management, or even performance. Experience has shown that designers and developers ignore these issues at their peril [11]. In a document-partitioned index, each index server is responsible for a subset of the documents in the collection. Each incoming user query is received by a front-end server, the receptionist, which forwards it to all n index nodes, waits for them to process the query, merges the search results received from the index nodes, and sends the final list of results to the user [12].

The System-Structure and Proposed Algorithms

In a distributed memory system, each process has its own address space and communicates with other processes by message-passing (sending and receiving messages). Each processor has its own local memory; the processors connected to each other. In distributed memory system, there is no limitation on number of processors and memory modules because the servers are connected as cluster-network, which can be extended to any required number.

In this work, number of servers in the cluster-network is 16-servers of identical properties. The proposed algorithms have two main parts; the first one relates to hardware of the work, and the second is about the software that guides these hardware components and manages the passing of messages between client-side and servers-side.

Hardware Part

The hardware part constructed of client-side and servers-side, this network is designed according to star topology. In such works, 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, for more accurate-results with acceptable comparisons, it is preferable to depend on identical-computers. Therefore, in this work all computers for both sides are identical completely, which have the following properties: (CPU: Core 2 Due, Speed: 2.6 MHz, RAM: 2 GB, and HD: 120 GB).

Client-side has only one host, which controls the sending of message passing operations to other side. Client-host contains the main program that can treat with all server-hosts individually, subgroups, or all of them. The secondary storage of the client-host stores the original data related with the addressed case study that must be send to other side, also stores the receiving results calculated by the servers-side.

Servers-side consists of sixteen hosts connected in the way to get a cluster of 16-sockets. Each socket contains a program constructed from many sub-programs and functions that have the ability of receiving data, making the required processing, calculating the results, and then returning them to the client-side. Servers-side can stores the received data and the determined results on their secondary storages, or returns them directly to client-side.

Software Part

As the hardware part consists of two sides, the software part also consists of two sides, which are client-side-software and servers-side-software.

Client-side-software represents the main-program, which is responsible of the following tasks:

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

Servers-side-software represents the programs that service the commands issued from the main program (i.e. client program). The software at each server-host is responsible of the following tasks:

1. Detecting the connection status of the client-host.
2. Deciding which sockets to work according to number of server-sockets sent by the client, taking into consideration that may be several of these server-sockets be out of work for certain numbers of server-sockets, for example; if number of server-sockets is 2, then only servers (1 & 2) will work and the servers (3 to 16) will be out of 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 will treat with that part of data that selected for it by the client-host.
7. Sending the calculated results to client-host, knowing that these results will be arranged in a form to be managed and assembled by the client-host in a suitable manner.
8. Each server-program contains all subroutines of the same case study. This gives the server-program the ability of treating with any selected part of data and chose the appropriate subroutine to calculate the required results.

Messages Transferred Between Client-side and Servers-side

There are two types of messages related to this approach of PP which are (control-messages and data-messages).

Control Messages:

Control messages issued by client-host and sent to server-sockets. These messages control the management of the processing overall the network and monitor the performance of the hosts especially server-hosts. This work uses the following control messages:

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

Data Messages:

Data messages; issued by client-side and/or servers-side. These messages carry specific data, which help running processes at server-sockets if the messages are issued by client-host. Also, may be representing specific results if the messages issued by server-sockets. This work uses the following data messages:

1. Starting task time (issued by client).
2. Starting CPU time (issued by each server).
3. Size of data-arrays that must be generated by client then used by servers).
4. Names of files containing these data-arrays used by both client-side and servers-side.
5. Starting running time (issued by each server).
6. Size of data-arrays that must be generated by servers after processing and used by client later for rearrangement.
7. Names of files containing these data-arrays to be used by both client-side and servers-side.
8. Terminating CPU time (issued by each server).
9. Consumed CPU time (issued each server).
10. Consumed running time (calculated by each server).
11. CPU usage percentage ratio (calculated by each server).

Matrix Algebra Case-Study

Matrix algebra is the famous type case-study related to PP. However, in this work, there are sixteen servers used and it can be N-servers according to the capacity of the laboratory of these experiments as shown in Figure (1).

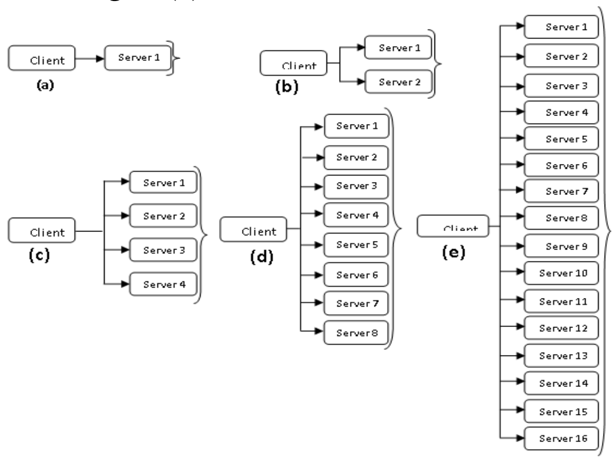

Figure (1): Cases of Matrix Algebra Algorithm. (a) One-server. (b) two-servers. (c) four-servers. (d) eight-servers. (e) sixteen-servers. Number of servers=2^m, where m= {0, 1, 2, 3 and 4}

The algorithm is designed to treat with two original matrices of square order (4096, 4096) as maximum depending on the host's RAM. This means that each matrix will contains (16,777,216) elements. Therefore, there will be (33,554,432) elements divided into sub parts (sub-matrices) to perform the (Addition, Subtraction and Multiplication) operations using (one, two, four, eight and sixteen) server-sockets. Figure (2) represents a sample of load-division among 16-servers in balance form.

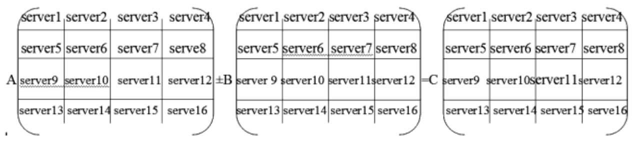

Figure (2): Sample of load-division representation among 16-servers for Matrix Algebra Algorithm Case-Study

Results and Discussion

The obtained results are stored in tables and plotted as shown in Figures (3 to 13). The results are divided into two main groups; first one is related with the average values of timing and usages for servers-side which are represents the average of related times or usages for all servers as an acceptable value to be depended, these values are plotted as shown in Figures (3, 4, 5, 8, 9 and 10), Figure (11) represents the average of CPU-Usage for all servers and for all sets of elements. The second group of the results is an additional assessment of performance of this work in the view of the latest returning results by the servers-side which is named here as Maximum-Values. These results are shown in Figures (6, 7, 12 and 13).

The results that represent the three subsets of matrix orders {(64, 128, 256), (512, 1024) and (2048, 4096)} are plotted in three separated Figures which are (3, 4 and 5) respectively, this arrangement is dependent because of the high-gap of obtained-results among these sub-sets which cannot be cleared in one figure, these results are very acceptable.

The results of Figures (6 and 7) represent the maximum consumed CPU time also are very acceptable, in this case the results of orders less than (512) which are small-loads are ignored because of the instability of decreasing the time with the increasing number of servers. This is applied also on the results of Figures (12 and 13) that related with maximum total execution-time of the program.

Results of Figures (8 to 10) represent total execution-time of matrix-algebra operations which are very acceptable.

Figure (11) illustrates the average CPU-usage of all servers with all tested sets of matrix-orders; it represents the relationship between CPU-usage and number of servers to determine the Average CPU-usage according to all cases of matrix-orders. It is clear that CPU-usage is increasing with increasing of the load for the same number of used-servers. It is expected that for each certain number used servers, the value of CPU usage will increase by increasing the load. This is clear with the cases of high-number of servers (i.e. > 2 servers), but for (≤ 2 servers) the average of CPU-usage is unstable and may be the changing is independent on this manner because the value of CPU-usage is affected by any instance under-running tasks depending on the computer status, also may be the nature of the data that under-processing effects on the value of CPU-usage.

Conclusions

The main points arise from the research employed in this paper can be summarized by the following:

1. Distributed memory systems addressed depending on client/servers principles with a network consists of seventeen nodes one of them is a client and the others are servers.
2. The algorithms used here are capable of calculating: the (Started, Consumed, and Terminated) times for (CPU and total execution), CPU usage of servers, and (CPU and total execution) times for the Client.
3. The algorithms are designed in very active programming-routines to get minimum loss of spend-time through the running state (at both Client and Servers sides).
4. Matrix Algebra Case-study is addressed, and there are many general algorithms and other related algorithms. All these algorithms are designed and tested completely by this work.
5. The obtained results are checked and monitored by special programming-checking-subroutines through many testing-iterations and proved a high degree of accuracy.
6. The results showed that parallel-processing operations are ineffective and inefficient with small load applications, and this efficiency is growing with increasing the task load. So, the highest load task will be implemented in high efficiency and the lowest load task implemented with low efficiency, taking in the consideration number of servers used.
7. All depended algorithms in this work are built in NetBean-Java Language.

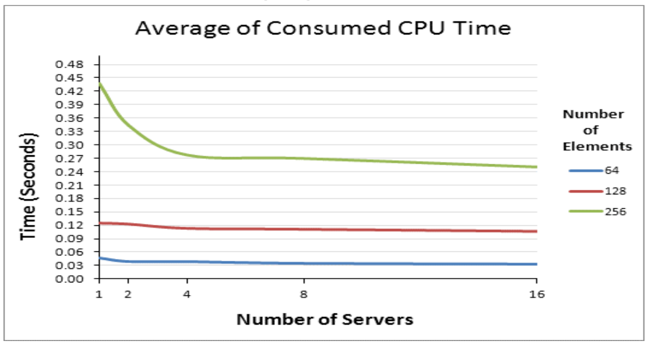

Figure (3): Average Consumed CPU Time for Matrix-Algebra of (64, 128 and 256 Orders)

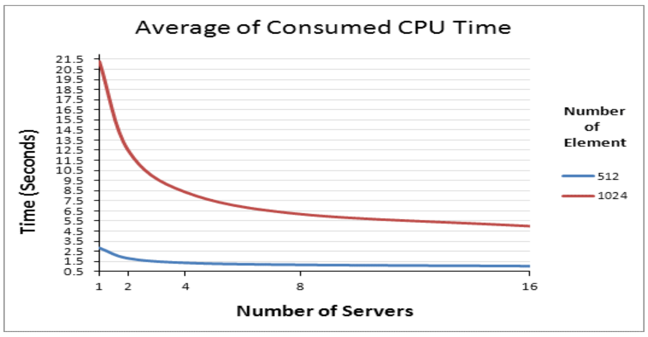

Figure (4): Average Consumed CPU Time for Matrix-Algebra of (512, and 1024 Orders).

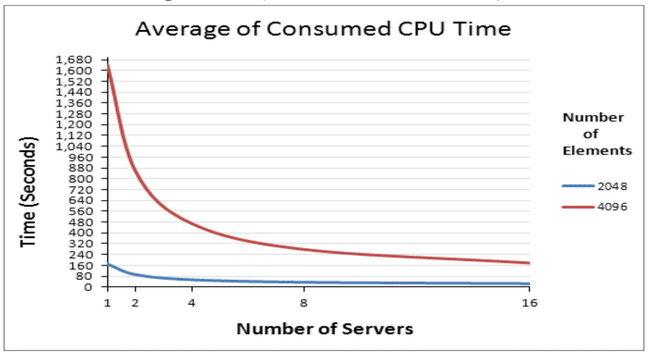

Figure (5): Average Consumed CPU Time for Matrix-Algebra of (2048, and 4096 Orders).

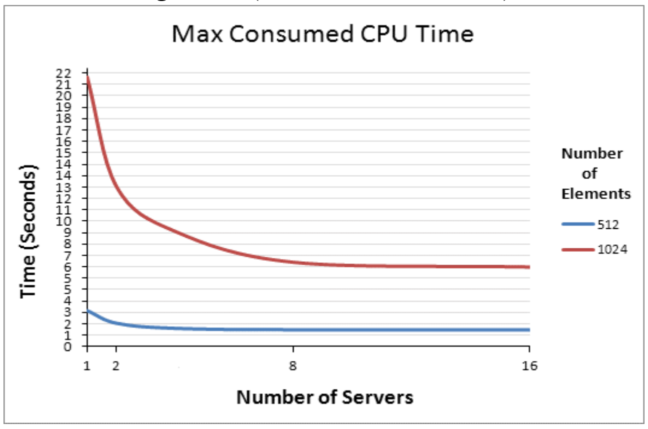

Figure (6): Maximum Consumed CPU Time for Matrix-Algebra of (512, and 1024 Orders)

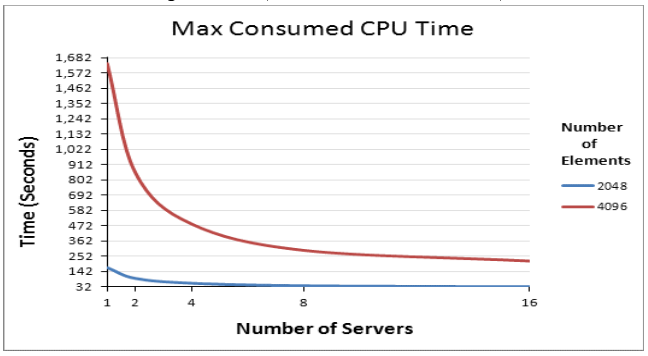

Figure (7): Maximum Consumed CPU Time for Matrix-Algebra of (2048, and 4096 Orders)

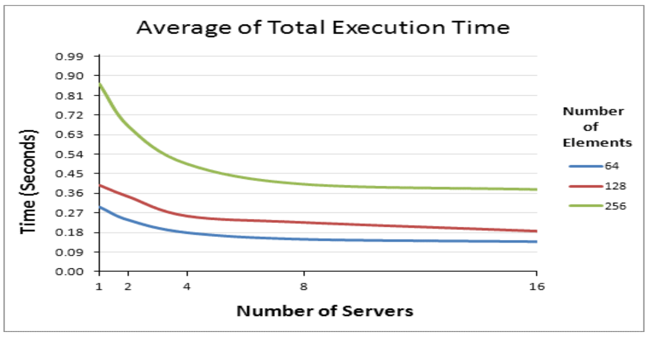

Figure (8): Average Total Execution Time for Matrix-Algebra of (64, 128, and 256 Orders)

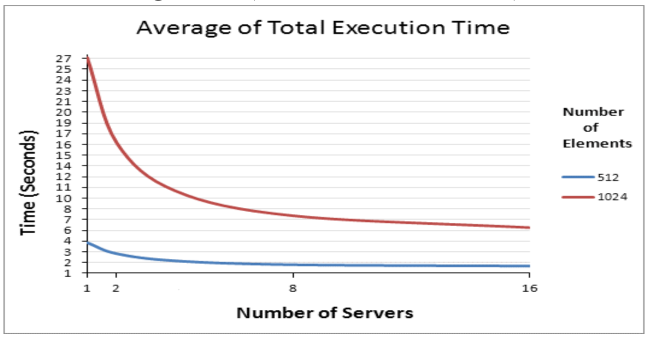

Figure (9): Average Total Execution Time for Matrix-Algebra of (512 and 1024 Orders)

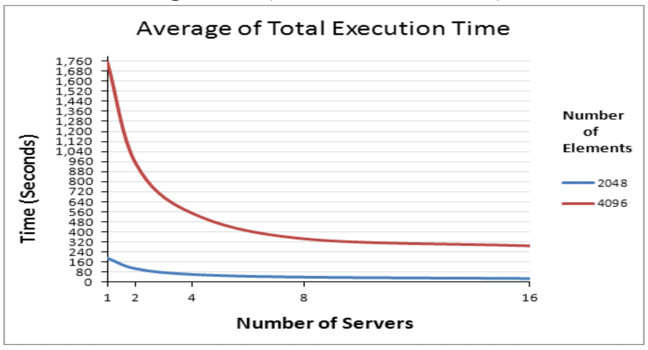

Figure (10): Average Total Execution Time for Matrix-Algebra o f (2048 and 4096 Orders)

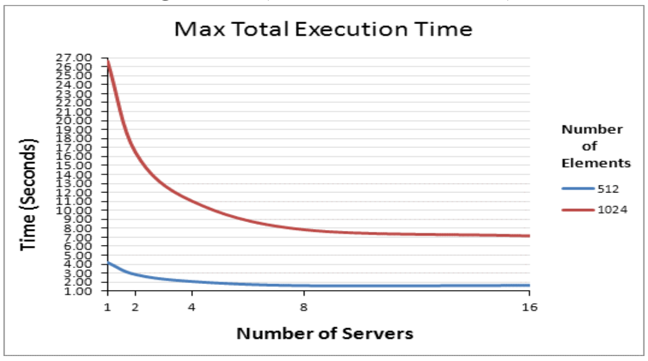

Figure (11): Maximum Total Execution Time for Matrix-Algebra of (512 and 1024 Orders)

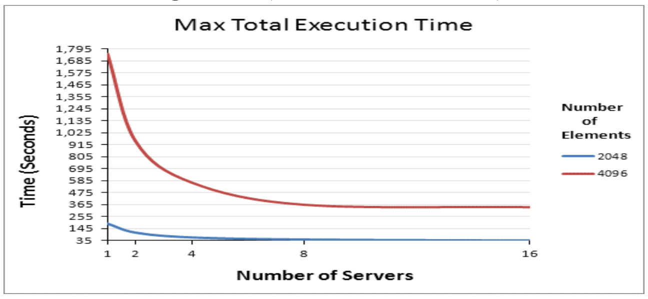

Figure (12): Maximum Total Execution Time for Matrix-Algebra o f (2048 and 4096 Orders)

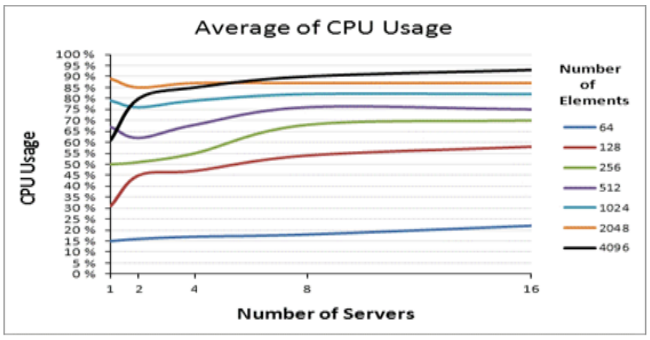

Figure (13): Average CPU-Usage for Matrix-Algebra of (64, 128, 256, 512, 2048 and 4096 Orders)

References

[1] Hank Dietz,hankd@engr.uky.edu, "Linux Parallel Processing HOWTO", http://aggregate.org/LDP/, v2.0, 28-06, 2004.

[2] Marcelo R. Naiouf, Parallel processing. "Dynamic Load Balance in Sorting Algorithms", University Nacional de La Plata, Facultad de Ciencias Exactas, September 2004.

[3] H. El-Rewini and M. Abd-El-Barr ," Advanced Computer Architecture and Parallel Processing", ISBN 0-471-46740-5 John Wiley & Sons, Inc, 2005.

[4] Eitan Frachtenberg, "Job Scheduling Strategies for Parallel Processing", JSSPP, June 17, 2007.

[5] Professor Thomas Braunl , "PARALLEL PROCESSIING: Parrallllell Computterr Arrchiittecttturre and Parrallllell Sofffttwarre Desiign", Book, University of Western Australia, 2010.

[6] Ameya Waghmari, "What is Parallel Processing", BE SCE Roll No. 41, 2000.

[7] Mohamed Iskandarani and Ashwanth Srinivasan, "Introduction To Parallel Computing, Notes on Parallelization Strategies", November 12, 2008.

[8] Nicholas Carriero and David Gelernter, "HOW TO WRITE PARALLEL PROGRAMS", Book, Massachusetts Institute of Technology, 1992.

[9] Y. F. Funga, M. F. Ercanb, Y. S mChonga, T. K. Hoa, W. L. Cheunga and G.Singha, "Teaching parallel computing concepts with a desktop computer", The Hong Kong Polytechnic University, 2003.

[10] Dr. Tran, Van Hoai, "Parallel Computing", HCMC University of Technology, 2010.

[11] Chris Loosley and Frank Douglas, "High-Performance Client/Server", John Wiley & Sons © 1998.

[12] DRAFT, "Information Retrieval: Implementing and Evaluating Search Engines", MIT Press, 2010.

تأثيرات تنفيذ المعالجة المتوازية على التقسيم المتوازن للحمل أعتماداً على أنظمة الذاكرة الموزعة

نعمان عمر ياسين صبحي رفيق محمد زيباري

E.mail:subhizebari@yahoo.com

الخلاصة

المشاكل المعقدة تحتاج إلى وقت طويل لكي تحل، مع كفاءة وأداء قليلين. لذلك، للتخلص من هذه المساوئ الدراسات ذهبت بأتجاه مناهج تجزئة المشكلة إلى أجزاء مستقلة، ومعاملة كل جزأ على حدة بحيث أن كل عنصر معالجة يمكن أن ينفذ الجزأ المخصص له من المشكلة بشكل آني مع باقي العناصر .المعالجات المتوازية هي أنظمة حاسبات تحتوي على وحدات متعددة المعالجة مرتبطة مع بعضها عن طريق شبكة ترابط متداخلة وأيضاً البرامجيات المطلوبة لجعل وحدات المعالجة تعمل مع بعضها. المعالجة المتوازية تقسم إلى ثلاثة أنواع : أنظمة الذاكرة الـ (المشتركة، الموزعة والمختلطة).في هذا البحث، تم تناول أنظمة الذاكرة الموزعة أعتماداً على مادئ العميل/الملقمات، الشبكة يمكن أن تحتوي على أي عدد من العقد؛ أحداها هو العميل والبقية هي ملقمات. الخوارزميات المستخدمة هنا قادرة على حساب (الأزمنة:البدائية، المنتهية، المستغرقة من قبل وحدة المعالجة المركزية وكذلك التنفيذ الكلي. أضافة إلى نسبة أستغلال وحدة المعالجة المركزية) للملقمات. وكذلك الزمن المستغرق من قبل وحدة المعالجة المركزية والزمن الكلي للعميل. هذا البحث يتناول منهج مطور لتقسيم المشكلة بهيئة متوازنة وتصميم خوارزميات مرنة للأتصال بكفاءة بين جانب–العميل وجانب–الملقمات بحيث يتم التخلص من مشاكل ماديات مكونات الشبكة ومشاكل ارسال الرسالة. نحن تناولنا هنا حالة دراسة جبر المصفوفات لعرض تأثير تقسيم–الحمل المتوازن لهذا المنهج. تم فحص ومراقبة النتائج المستحصلة بواسطة برامج فرعية خاصة للفحص من خلال التكرارات–الأختبارية وبرهنت درجة عالية من الدقة. جميع هذه الخوارزميات تم تنفيذها بأستخدام لغة جافا.