Effect of Parallel Distributions System on Computer Efficiency and Performance

Jehan K. Shareef
Media Faculty/ Thi-Qar University,
Computer Center/ Thi-Qar University
jihansh@yahoo.com

Abstract

In this paper, the distribution parallel system tool MPI (Message Passing Interface) to solve squared matrixes product was used. In order to make parallel tasks, the work will be implemented at least by using two computers. Also, several tasks will be included in this project to estimate the Speedup of the processor (*p*), Efficiency of the processor (*p*), fraction *f* of sequential work and fraction of parallel part of the work using Amdahl's law. Moreover, this paper will estimate many approaches which are: running the application on 1 processor for 3 different values of matrix elements *N* (for example 1000 ×1000, 2000 × 2000, and 3000 × 3000), running the application on 2, 3 or more processors for the same values of *N*, estimating fraction *f* of sequential part, estimate *f* for 3 different values of N by having measured times, finding out whether this fraction depends on the problem size N, and more operation to complete the task required.

Keywords: computer efficiency , MPI, and parallel distribution system.

1- Introduction

Computer efficiency is considered by the volume of useful work accomplished by a computer system or computer network compared to the time and resources used (Allen, 1994) (Kolmogorov, 1965). Good computer performance measures by (Li & Vitanyi, 1997):

a) Short response time for a given piece of work.
b) Rate of processing work.
c) Low utilization of computing resource (s).
d) High availability of the computing system or application.
e) Highly compact of data compression and decompression.
f) High bandwidth / short data transmission time.

Parallel distributed processing model is a class of objectively motivated information processing models that attempt to model information processing by the way it actually takes place in the brain (Rumelhart , Hinton, & McClelland, 1986) (Shannon, 1948). This model was developed because of conclusions that a system of

neural connections gives the idea to be distributed in a parallel array in addition to serial pathways (Jordan & Alaghband, 2003) (Tanenbaum, 2005).

The work in this paper will describe parallel matrix multiplication program and the way that is implemented it. MPI (Message Passing Interface) method used to implement the parallelism. The research contains characteristics estimation to explain problem of parallel multiplication of matrices. The program is written by using C language and MPICH2.1.4 is used.

The result of matrix multiplication is a matrix whose elements are found by multiplying the elements within a row from the first matrix by the associated elements within a column from the second matrix and summing the products (Lerner & Trigg, 1991).

2-The Materials and Methods of this Search

2-1 Description of the Algorithm for the Matrix Multiplication

In the serial matrix multiplication implementation, it requires that same equation to be applied sequentially to calculate every entry of the resulting matrix. Traditionally, software has been written for serial computation:

1. To be run on a single computer having a single Central Processing Unit (CPU);
2. A problem is separate to many discrete series of instructions.
3. Instructions are executed one by one.
4. Only one instruction may execute at any moment in time.

A parallel computing architecture allows all entries processed concurrently at each operation. In the simplest sense, parallel computing is the simultaneous use of multiple compute resources to solve a computational problem:

a. To be run using multiple CPUs
b. A problem is broken into discrete parts that can be solved concurrently
c. Each part is further broken down to a series of instructions
d. Instructions from each part execute simultaneously on different CPUs.

A parallel solution to the matrix multiplication problem can be given with the following algorithm:

Imagine that there are 1 Master processor and some slaves: main ( ) {

- Get the number of processors; (Processor_Count)
- Get the processor id; (Processor_Rank)
- Initialize Matrix A and Matrix B
- Divide Matrix A between all the slaves

- Send the matrix A between the processors and determine the offset and number of rows for each processor;
- Broadcast Matrix B to all the slaves
- Calculate each entry of matrix C by using the block of rows of matrix A assigned to the current processor and matrix B;
- if (processor id= ! master)for each slave untill all slaves have handed back the processed data calculate Matrix Multiplication and put the result in Matrix C; if (processor id= =master)print each entry of matrix C;}

The master is responsible for both the data Input /Output (I/O) and the calculations of the entries of matrix C, where the workers are responsible for the calculations of the entries only. In the algorithm, both master and workers calculate their own data then, master receives the results from the workers and prints them together with its own.

2-2 - The Parallel Facilities for Implementation this Work

2-2-1 MPI Application

Message Passing Interface (MPI) is a message passing library interface specification. MPI addresses principally the message passing parallel programming model, in which data is moved from the address space of one process to that of another process through supportive operations on each process.

Message Passing Interface (MPI) is a specification for a standard library for message passing that was defined by the MPI forum, a broadly based group of parallel computer vendors, library, writers, and applications specialists (Gropp & et al., 1996). MPI is combine interface, the protocol and semantic specifications for how its features must behave in any implementation (such as a message buffering and message delivery progress requirement). MPI includes point-to-point message passing and collective (global) operations, all scoped to a user-specified group of processes.

The highest advantages of creating a message passing standard are portability and ease of use. MPI simply stated is to improve an extensively used standard for writing message passing programs. As such the interface should establish a practical, portable, efficient, and flexible standard for message passing.

2-2-2 The Reason for Using MPI in this Work

MPI provides a powerful, efficient, and portable way to express parallel programs. MPI achieves portability by providing a public domain, platform-independent standard of message passing library. It specifies this library in a language-independent form, and provides FORTRAN and C bindings. Due to these reasons, MPI has gained wide acceptance in the parallel computing community and it is available on a wide variety of platforms, ranging from massively parallel systems to network of computers, or workstations.

MPI provides nearly 200 functions, but a wide range of parallel programs can be solved using its six basic functions that initiate and terminate a computation, identify processes, and send and receive messages. These six functions are:

- *MPI_INIT* Initiate an MPI computation
- *MPI_COMM_SIZE* Determine the number of processes
- *MPI_COMM_RANK* Determine the process identifier
- *MPI_SEND* Send a message
- *MPI_RECV* Receive a message
- *MPI_FINALIZE* Terminate the computation
-

The structure works functions in this paper are as the following:

a. Initial MPI Calls

The first MPI call in a program must be MPI_INIT to initialize the environment. This is usually followed by a call to MPI_COMM_SIZE to determine the number of processes taking part in the computation, and a call to MPI_COMM_RANK to find out the rank of the calling process.

b. Point-to-Point Communication Calls

Involve sends and receives between two processors. There are two basic categories of sends and receives, which are either blocking or non-blocking. A blocking call is one that returns when the send (or receive) is complete. A non-blocking call returns immediately and it is up to the programmer to check for the completion of the call. There are many different types of communication modes, but only the blocking standard send MPI_SEND and receive MPI_RECV is used in this project.

c. Leaving MPI Call

MPI_FINALIZE is the only routine that completes the program. All programs must call MPI_FINALIZE as the last call to an MPI library routine. It cleans up all MPI states and shuts down a computation. Beside these functions MPI defines a timer, MPI_WTIME, which is very convenient for performance debugging. It returns a floating-point number of seconds, representing wall clock time. To time a computation, MPI_WTIME can be called just before the computation starts, and again just after the computation ends, then the performance is the difference between the two times.

2-3 Introduction MPICH2

MPICH is a freely available, complete implementation of the MPI specification, designed to be both portable and efficient. MPICH2 is extension of MPI1 with high performance and widely portable implementation of message passing interface (MPI) standard (Gropp & et. al., 2007). The goal of MPICH2 are (1) to provide an MPI implementation that supports different computation and communication platforms

including commodity clusters (desktop systems, shared-memory systems, multicore architectures), high-speed networks and propriety high-end computing, (2) to enable cutting-edge research in MPI through an easy-to-extend modular framework for other derived implementations

2-3-1 Installation and Configuring MPI with MPICH2

The MPICH2 setup can be downloaded from the website http://www.mcs.anl.gov/research/projects/ mpich2/ (Wong, 2012). This version is 32 bit edition. The installation phase proceeds with the following steps;

- Run the MPICH2 installation executable to start the installation procedure. Then, it will appear the start menu read the license agreement and click (*I Agree*) to go for the next stage.
- There is a step that the process manager setup remind the user to remember the passphrase given. The default value of the passphrase is "behappy".
  After the mpich2 installation is done, do the following steps.
- Add the following to system path (in windows Seven: right click My Computer/Properties/Advanced System Settings/ Advanced/ Environment Variables)
  C:\Program Files\MPICH2\bin
- Edit the path variable for either the system or user variables (figure (1)). This also works on XP and Vista.

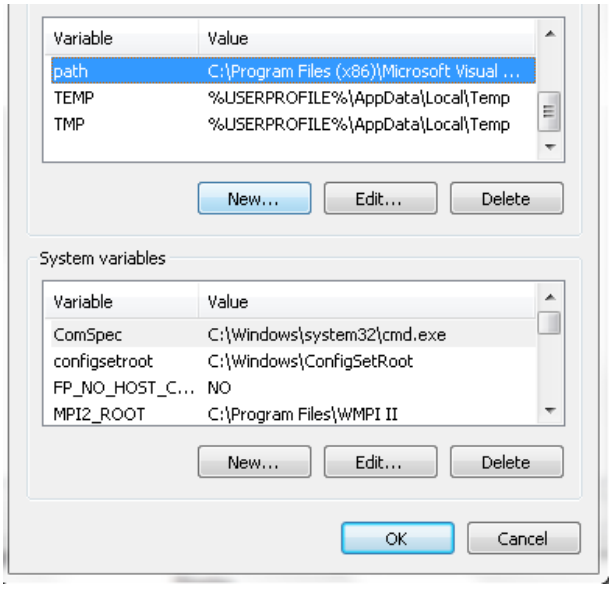

Figure (1): Path variable

- Add the following exceptions to your firewall:
  C:\Program Files\MPICH2\bin\mpiexec.exe
  C:\Program Files\MPICH2\bin\smpd.exe

Usually, you will receive a message to unblock these programs. But in case you don't, you have to add them manually.

- Open port 8676. Go to

C:\Program Files\MPICH2\bin\wmpiconfig and select "port" at the top of the list.

2-3-2 Configuring Visual Studio 2010 to Run C programming Language

Now, to setup the environment in Visual Studio we have to tell the compiler where to find the MPI header files and libraries, so, do the following:

1- Run Visual Studio 2010

2- Click on project Name, and select menu “Project” and choose “Properties”. Click “Configuration Properties”, then choose C/C++ and click on “General”. On “Additional Include Directories”, add (or browse) the following entry: *C:\Program Files\MPICH2\include*. See (figure (2)).

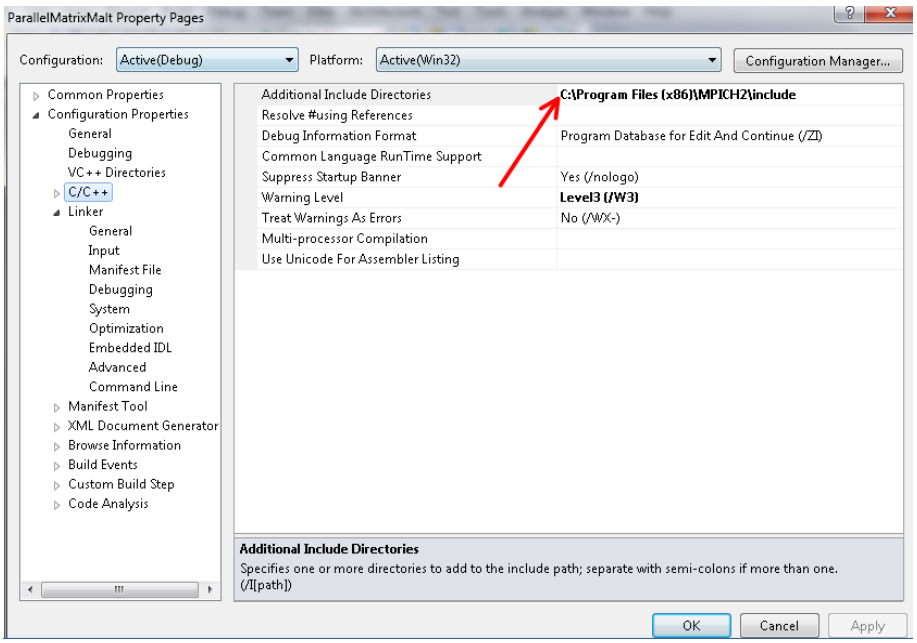

Figure (2): Finding include path

3- Select “Linker”> on the left side. Choose “Additional Dependencies “ by add “mpi.lib” and click OK button as shown in Figure (3).

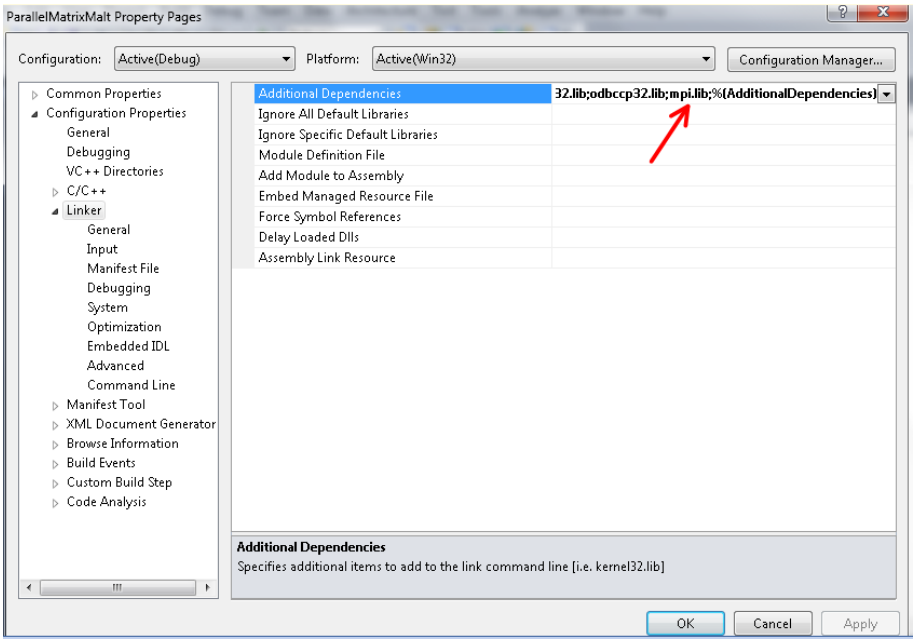

Figure (3): Adding mpi.lib

4- In the last step as figure (4), Select "Linker" tree menu and choose “General”. In “Additional Library Directories” add the following entry:

C:\Program Files\MPICH2\lib

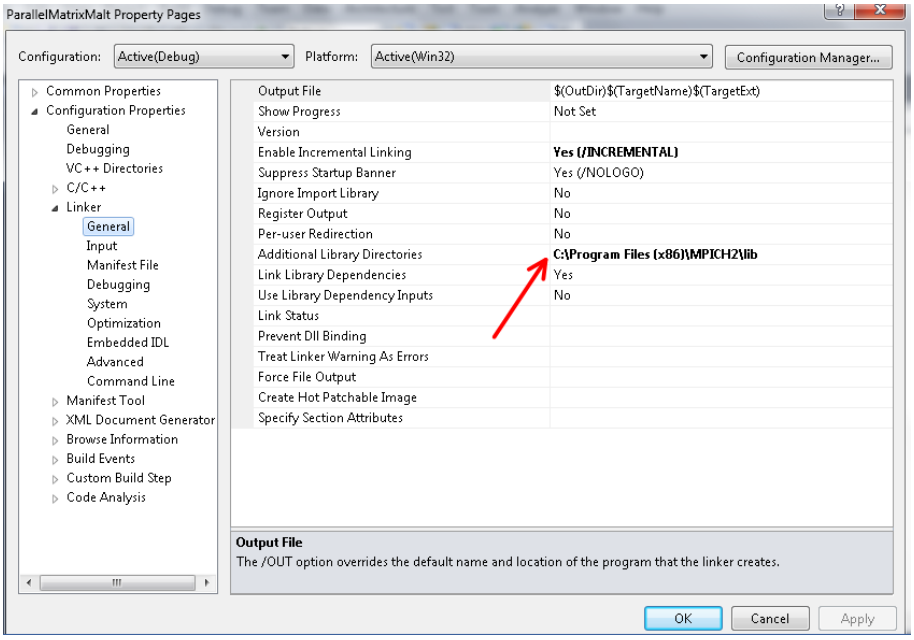

Figure (4): Finding lib path

Then click Ok to finish the steps.

3- The Developing Program

3-1 Network connection

To execute a program using MPICH 2, at first you should make a connection between computers, if you are going to build a LAN network, you should use cross cable, unless you can make a wireless network. For wireless network, you can create an ad hoc network, then set IP address to each machine, choose one of them as the master and the others as the salves. After making connection, each machine should ping with the others.

3-2 Cluster Configuration

The following steps should perform on all machines:

- Create an administrator user for each PCs with the same username and password, then login to the users on each machine and go on the next steps
- Register a user account with MPI. Go to for each PC:*C:\Program Files\MPICH2\bin\ wmpiregister*

Type in the user account and password that will be used to start MPI jobs. You should have the same user account and password on all your cluster machines. Click *Register* to register each PC, which will join to the execution. Configure each PC with the following structure:

- From Start/MPICH run the wmpiconfig.exe on each PCs
- From the top list, choose WORKGROUP and click on "get host button" and then click on scan button. It begins to scan all the machine that are used in performing the program, after finishing the scan , all the node should be green otherwise there is a mistake with MPICH, there is an error filed in the window that help you to recognize the problem. You should check the network to have the right connection between PCs. If the problem did not solve you should uninstall MPICH and install it again.

4-Running the Task of this Paper

After debugging the C code in Visual Studio 2010, an exe file will create in debug folder, Start running the exe file, in the master machine. Go to this path (figure (6)):

C:\Program Files\MPICH2\bin\wmpiexec

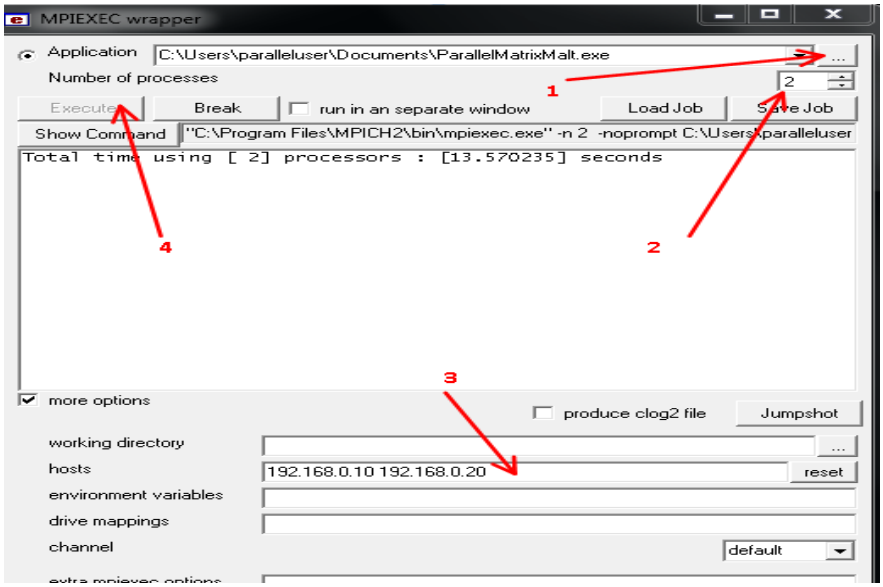

Figure (6): Running the program with wmpiexec

As in figure (6) above do the following:

1- Load the exe file (1)

2- Determine the number of processes (2) , Check the "more option" (if you are going to run the program on just one computer, go to the step 4)

3- In host you should type IPs of machines that are using to run the program, separate each of them with a space. (3)

4- Press the Execute bottom (4)

If you look at task manager, in tab process, you will see that a process has created on each machine, depends on the number of processes that you have entered in field process, there are processes on every machine, for example if the number of process is equal 4 and you have 2 machines, when you see tab process on task bar of each machine, there are two processes there, that means processes created along with the master process. All PCs, both master and slave, will join to the computation. Each PC reads the number of processes and its own process's rank. Then, each PC initializes the values of matrix A and matrix B, gets its own block of rows and calculates the entries of matrix C associated with its own block. After the simulation, the slave sends its results to the master and terminates. On the other hand, master collects the results and terminates. After doing the multiplication, a massage will show on the screen that says the time of computation.

5- Descrıptıon of the Conducted Experıments and the Results

Firstly, in order to do simulation the connection between computers must be done (there is not difference between local connection or wireless connection). Then run the program in master computer, so the work is distributed into PCs that are connected. After calculations, each PC sends the result to master PC and then the master PC displays all results.

For comparing the results that we were found, more than one PC are used with the same size of matrix to find the improved result. İt is clear that performance the task using multiple PCs is better than 1 PC, for instanse, with N=2000 ×2000,running the program by 1 PC is required 22.32 second rather than it is need 16.43 second within 2 PCs, table (1) below shows the overall results:

Table (1): T1 and Tp of the task

Matrix sizes(A and B)	1 pc T(1)	2 pcT(p) 2 processes
N=1000	T1= 6.984459 sec	Tp= 7.053245 sec
N=2000	T1= 22.3214 sec	Tp= 16.43122 sec
N=3000	T1= 55.45678 sec	Tp= 32.5697 sec

The performance of 1 PC and 2 PC is calculated as,

$$S(p) = T(1)/T(p) \quad \text{..... (1)}$$

Where T(1) is the time of run the program on 1 Pc and T(P) is the time to run the program with muliple PCs. After finding T (1) and T (P), we could then calculate the fraction on serial work done on the program using Amdahl's law.

$$f = \frac{T_p - T_1/p}{[T_1(1-1/p)]} \quad \text{…..…… (2)}$$

When
N=1000,
f(1) = (7.053245-6.984459/2)/ 6.984459(1-1/2) = 1.019
N=2000,
f(2) = (16.43122-22.3214/2)/ 22.3214(1-1/2) = 0.472
N=3000,
f(3)= (32.5697-55.45678/2)/ 55.45678(1-1/2) = 0.174

In the table (2) below, it can be seen that how speed up is improved with more PCs. For instance, the speed up of using 2 PCs is more than speed up of using 1 PC.

Table (2): Speed up and efficiency of computer

Matrix sizes (A and B)	Speed up S(P)=T(1)/T(P)	Efficiency E(P)=S(P)/P
N=1000	0.990	0.495
N=2000	1.3584	0.6792
N=3000	1.7027	0.85135

Briefly, the speedup gained will increase when the matrix sizes and the number of PCs are increased. Similarly, the efficiency will increase when the number of processor increases with the increasing matrix sizes.

6- Conclusıons

This paper is inspired from the necessity to use computing ability to address computation problems like multiplication two matrixes of large size (a size more than 3000 ×3000). It is mainly covers several techniques to facilitate use of computer clusters in satisfying computational task.

This work defined away to perform parallel multiplication via the MPI over a reliable software package that permits us to take advantage of an underlying broadcast medium, performance measurement demonstrates the feasibility of such an approach and show reasonably effective of using multiple PCs for patterns of communication required by representative parallel applications.

In conclusion, using parallel processing tools (MPI) with the applications gives better performance including SpeedUp, Efficiency and so on. In this project, the necessarity of MPI (Message Passing Interface) to run the application is improved. Also, when matrix size increase more than 2000×2000 the parallel calculation has better result i.e. It is good in high numeric size. On the other hand, the speedup and efficiency are increase, when the numbers of PCs are increase with increasing the matrix size at the same time.

Finally, this research gives us a clear understanding of parallel programming and how to implement it on a distributed environment. We can develop a parallel program and run it on multiple computers.

References

Allen, A. O. (1994). *Computer Performance Analysis with Mathematica.* Academic Press.

Gropp, W., & et. al. (2007). *MPICH2 User's Guide.* U.S.A: Mathematics and Computer Science Division DAC Program.

Gropp, w., & et al. (1996). *MPI: A Message-Passing Interface Standard.* University of Tennessee.

Jordan, H., & Alaghband, G. (2003). *Fundamentals of Parallel Processing.*

Kolmogorov, A. N. (1965). Three Approaches to the Quantitative Definition of Information. *Probl. Inform.Transm*(1), 3-11.

Lerner , R. G., & Trigg, G. L. (1991). *Encyclopaedia of Physics* (2nd ed.). VHC publishers.

Li, M., & Vitanyi, P. (1997). *An Introduction to Kolmogorov Complexity and Its Applications* (2nd ed.). New York, USA: Springer-Verlag.

Rumelhart , D. E., Hinton, G. E., & McClelland, J. L. (1986). *A General Framework for Parallel Distributed Processing.* MIT Press: Cambridge, MA.

Shannon, C. E. (1948). A Mathematical Theory of Communication. *Bell Sys. Tech. J.*(27), 379–423.

Tanenbaum, A. S. (2005). *Structured Computer Organization.* NJ, USA: Prentice Hall PTR: Upper Saddle River.

Wong, P. (2012). *MPICH*. Retrieved from http://www.mcs.anl.gov/research/projects/mpich2/

تأثير نظام التوزيعات المتوزاية على كفاءة الحاسوب وادائه

جيهان كاظم شريف
جامعة ذي قار/ رئاسة الجامعة

الخلاصة

في هذا البحث تم استخدام اداة نظام التوزيع المتوازي **MPI (Message Passing Interface)** لحل مسألة ضرب مصفوفتين مربعة ذات ابعاد كبيرة. من اجل تنفيذ النظام المتوازي ، تم تطبيق العمل على كومبيوترين على الاقل . كذلك هذا البحث يتضمن تخمين سرعة الكومبيوتر ،كفائته ، وايجاد نسبة وقت التنفيذ في كومبيوترواحد الى وقت التنفيذ في كومبيوترين (***f* fraction**) باستخدام قانون **Amdahl**. العمل نفذ على كومبيوتر واحد مع تغيير حجم المصفوفة المربعة (مثلا بحجم 1000×1000، 2000×2000 ، 30000×3000) ومن ثم على كومبيوترين وايجاد وقت التنفيذ في كل حالات الاختبار وتحديد هل ان (***f*** **fraction**) يعتمد على حجم المصفوفة ام لا. اضافة الى العديد من العمليات التي تم اختبارها لاكمال هذا البحث .

الكلمات المفتاحية : اداة **MPI** ، كفائة الكومبيوتر ، و نظام التوزيع المتوازي .