Evaluation Performancre of Multiplication two N×N Matrices using Different Number of Processors

Osamah Yaseen Fadhil
Dept. Computer engineering
Eastern Mediterranean University, Baquba, Iraq

Abstract— **Most parallel matrix multiplication algorithms use matrix decomposition based on the number of processors available. These include the systolic algorithm, Cannon's algorithm, Fox and Otto's algorithm, PUMMA (Parallel Universal Matrix Multiplication), SUMMA (Scalable Universal Matrix Multiplication), and DIMMA (Distribution Independent Matrix Multiplication). Each of these algorithms uses the matrices decomposed into sub-matrices. During execution, a processor calculates a partial result using the sub-matrices it currently has access to. It successively performs the same calculation on new sub-matrices, adding the new results to the previous. When all multiplication is complete, the root processor assembles the partial results and generates the complete matrix. In this paper, a program was designed to measure efficacy, speedup and other evaluation elements with Multiplication of two N*N matrices using different number of processors.**

Keywords—.NET Remoting;Server Process; client Process; Sequential Time; Speedup;

XI. INTRODUCTION

Matrix multiplication is commonly used in the areas of graph theory, numerical algorithms, digital control, and signal processing. Multiplication of large matrices requires a lot of computation time as its complexity is O(n3), where n is the dimension of the matrix. Because most current applications require higher computational throughputs, many researchers have tried to improve the performance of matrix multiplication. Even with improvements such as Strassen's algorithm for sequential matrix multiplication, performance is limited. For this reason, parallel approaches have been examined for decades. In this paper, Evaluation of Multiplication of two N*N matrices was done with (2, 4, 6, 8) number of processors [1].

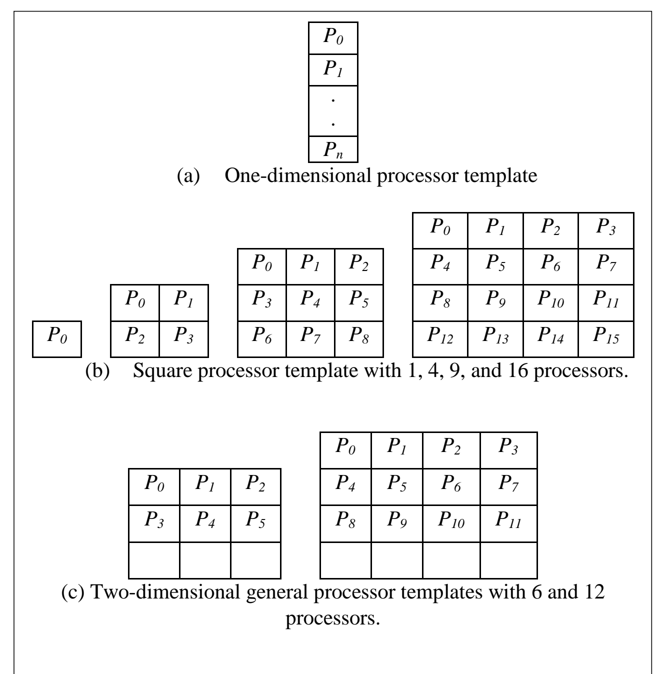
XII. ALGORITHM FOR MATRIX MULTIPLICATION (TASK FORMULATION)

A. *Matrix Decomposition for Parallel Algorithm*

To implement the matrix multiplication, the A and B matrices are decomposed into several submatrices. Four methods of matrix decomposition will be used in this study: one-dimensional decomposition, two-dimensional square decomposition, and two-dimensional general decomposition, and two-dimensional scattered decomposition. These are described below. One-dimensional decomposition: Here, the matrix is horizontally decomposed as shown in Fig 1-a. The ith processor holds ith Asub and Bsub and communicates them to two neighbor processors, i.e., to the (i-1) th and (i+1) th processors. The 0th processor and (n-1)th processor communicate with each other as in a ring topology.

Two-dimensional square decomposition: Here, the matrix is decomposed into square processor template as shown in Fig 1-b. Since a maximum of 16 processors will be used for this study, 1, 4, 9 and 16 processor templates are used. Each processor communicates with its four neighbors, i.e., north, south, west and east of itself as in a two dimensional torus. Two-dimensional general decomposition: Here the matrix is decomposed into two-dimensional processor template. This decomposition allows the square processor templates as well as 2×3, and 3×4 with 6 and 12 processors respectively. Each processor communicates with its four neighbors just as square decomposition. Fig 1-c shows 2×3, and 3×4 processor templates. Two-dimensional scattered decomposition: Here, the matrix is divided into several sets of blocks. Each set of blocks contains as many elements as the number of processors, and every element in a set of blocks is scattered according to the two-dimensional processor templates. The two-dimensional processor templates contain 2×2, 2×3, 3×3, 3×4, and 4×4 structures for 4, 6, 9, 12, and 16 processors, respectively. Fig 1-d shows an example of a 6×6 matrix distributed onto a 2×3 processor template[1].

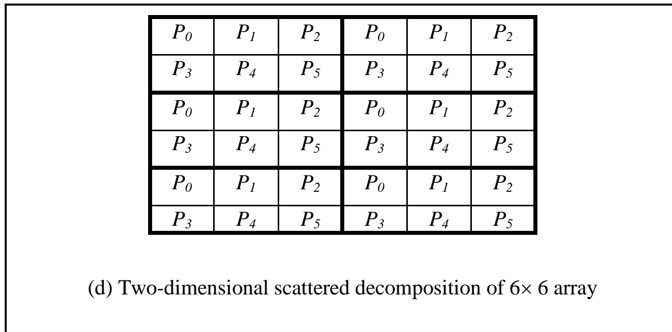
IJERT



(a) One-dimensional processor template

(b) Square processor template with 1, 4, 9, and 16 processors.

(c) Two-dimensional general processor templates with 6 and 12 processors.



(d) Two-dimensional scattered decomposition of 6× 6 array

Fig 1. Matrix decomposition

XIII. DESCRIPTION OF PARALLEL FACILITIES

Distributed Component Object Model (DCOM) and Dot Net Remoting are popular distributed technologies introduced by Microsoft. Both of these technologies enable inter-process communication across application domains. DCOM was the newer version of Network OLE, and was designed to work on multiple network protocols, including HTTP.

Dot Net Remoting is a new technology introduced with the Microsoft Dot Net Framework. It not only adds the features that were missing in DCOM (as listed below) but is a completely new and flexible architecture that allows users to customize solutions according to the problems at hand. Differences between the two technologies are as follows in Table 1[2].

TABLE.1 Differences between Distributed Component Object Model (DCOM) and Dot Net Remoting technologies

Features	***Dot Net Remoting***
Protocol support	Uses TCP or SOAP, depending on the problem.
Firewall support	Uses HTTP protocol for remoting to work easily across firewalls.
Cross-platform	Supports cross-platform communication.
Maintainability	Easy deployment either through XML-based configuration files or programmatically; easy maintenance with configuration files.
Object invocation	Client request will fail if the remoting server is not already started, and if the remoting component is not hosted in Internet Information Server.
Security	Security depends on the host of the Dot Net Remoting Object, for example, IIS.
Features	Dot Net Remoting

IJERT

A..NET Remoting with Visual Basic:

.NET Remoting Technology enables application communication. It is a generic system for different applications to communicate with one another. .NET objects are exposed to remote processes on the network. .NET Remoting allows interprocess communication on the same computer, on the same network, or even across separate networks. Remote objects are accessed through Channels. Channels physically transport the messages to and from remote objects. There are two existing channels TcpChannel and HttpChannel in .NET Remoting. Distributed computing is an integral part of almost every software development. (Before .Net Remoting, DCOM was the most used method of developing distributed application on Microsoft platform.) The two processes can exist on the same computer or on two computers connected by a LAN or the internet. We have used VISUAL BASIC Programming Language and .NET libraries for .NET Remoting[3].

As Fig.2 shows, when client calls a method, client sends request through the channel to the server. Then client receives the response sent by the server process. In .Net Remoting, a remote call to an object on a machine across network is transparent to the client application.

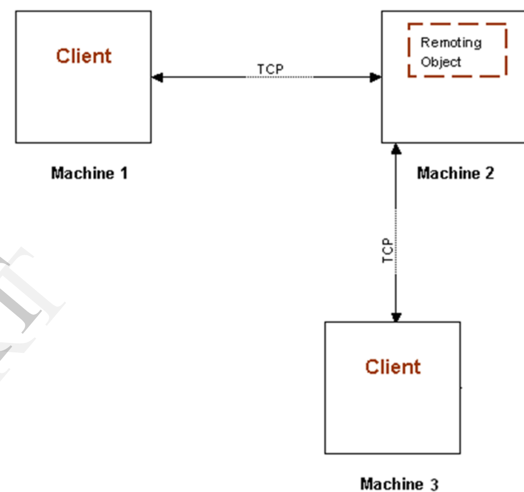


Fig 2. .NET Remoting System

B.. Singleton Objects

We need to create a server object that will act as a listener to accept remote object requests. For array multiplication, we used the TCP/IP channel. We created an instance of the channel for use by clients at a specific port. We used Singleton object where there is only one instance of the object is used for all clients. First client that reaches to the server creates the object on the server and all other clients use the same object. Singleton objects are those objects that service multiple clients and hence share data between client requests. They are useful in cases in which data needs to be shared explicitly between clients.

V. DESCRIPTION OF THE MATRIX MULTIPLICATION PROGRAM:

A. Server Process

In the main process on the server, we define the array size N, as seen in fig 4. The values for the arrays are entered one by one or in case of large number it will be created serially from one to end of value. All the information regarding matrices A, B and C are stored in the remote object created on the server process. We need to run the server process to assign tasks to the other processes. There is only one server process. As soon

as we see “Server started” message, client processes on the network start execution in parallel for array multiplication[4].

B. Client process

In our developed program, each client process copies its share of the columns on array B and all elements of array A (broadcasting of aik) in the remote object to its object. Each client process does its share(columns) of array multiplication. Each client process updates its share of array C in the remote object. Server process shows the final state of array C. As seen fig 5, Client process above has computed third column of array C because the server computed first and second columns[4].

VI.CHARACTERISTICS ESTIMATION

The program has been tested for many different values and under different conditions, with different kinds of processors. But in general we have taken the slowest time i.e. the times here are for the slowest processor that we have used.

A. Order of algorithm using "big O":

The formulated algorithm consisted of three nested loops, these loops calculates the two matrix multiplication. So, in this case the algorithm will be of the order of $O(N^3)$. The serial part of the program does not count so much in the program as these three loops so even if we add the serial part it will be the in the same order, i.e. does not change so much.

1) Results of One Processor:

We have started our experiments with different sizes of the arrays but with one processor. These results can be seen in Table 2.

IJERT

Table 2. Time for one processor.

Test #	Array size	Test Time
1	10	8
2	25	23
3	50	66
4	100	201.8

We have said above that the algorithm is in the order of $O(N^3)$, so to increase the array size from 10 to 25 , we have doubled the size of the array so the time when the size is 5 should be multiplied by 2.5^3 to get the theoretical results when the size of the array is 25. So for an array size of 50 the time should be multiplied by 5^3 and so on for the rest of the tests. The table 3 shows the theoretical results and the tests results:

Table 3. Theoretical and tests values for one processor.

Test #	Array size	Theoretical time	Test Time
1	10	8	8
2	25	125	23
3	50	1000	66
4	100	8000	201.8

A closer examination of the above results one can see that the values of test results are less than the theoretical results. This is shows that not all the program is working under the order of $O(N^3)$, because some parts of the program are done in the order of $O(N^2)$. This confirms our result of estimating the order of the algorithm of $O(N^3)$.

2) Testing With More than One Processor:

We have noted the measurements tests for up to 8 processors, even though the program can run on any number of processors the user like. So its free to choose the array size and the number of processors the users like.

a) Results of Two Processors:

The test results when only two processors are working are shown in table 4. These results are measured in seconds.

Table 4. Time for two processors.

Test #	Array size	Test Time
1	10	7
2	25	19
3	50	40
4	100	121

b) Results of four Processors:

The test results when only four processors are working are shown in table 5. These results are measured in seconds.

Table 5. Time for four processors.

Test #	Array size	Test Time
1	10	6
2	25	15
3	50	24
4	100	108

c) Results of six Processors:

The test results when only six processors are working are shown in table 6. These results are measured in seconds.

Table 6 Time for six processors.

Test #	Array size	Test Time
1	10	5
2	25	12
3	50	18
4	100	98

d) Results of eight Processors:

The test results when only eight processors are working are shown in table 7. These results are measured in seconds.

Table 7. Time for eight processors.

Test #	Array size	Test Time
1	10	4
2	25	9
3	50	14
4	100	79

B. *Fraction of Sequential Part:*

Using Amdahl's law we have obtained the following results for the above measurements, by using the following formula [5] (1):

$$F= ((T_p- (T_1/p)))/ (T_1*(1-1/p)) \qquad (1)$$

The times found in table (8) are the sequential times for all the tests we have conducted above and these times have been measured in seconds to be consistent with all the measurements we have got so far.

Table 8. Sequential Time for (2, 4, 6, 8) processors.

array size	2	4	6	8
10	0.1875	0.375	0.381944	0.328125
25	0.163043478	0.301630435	0.295894	0.233016
50	0.053030303	0.085227273	0.088384	0.076231
100	0.049801784	0.213887512	0.265802	0.233167

IJERT

C. *Dependence of Sequential Time:*

We have calculated the sequential time for every measurement we have made. From table 8, one can see that when the number of processors increases the sequential time decreases a very small amount and this is natural when we have a number of processors.
So one can conclude that the dependence of the f(N) is linearly dependent on the problem size, so one can say that its in the order of O(k), where k is constant.

D. *Speedup:*

Having measured the times for T_1, T_2, T_3, T_4, and T_5 we have calculated the speedup for the different values we have got using the formula (2):

$$Speedup = (T1 / T_p) \qquad (2)$$

Where, T_p is the time of parallel processing using P processors [5]. The results obtained are tabulated in table 9

TABLE 9 Speedup Data

Array size	1	2	4	6	8
10	8	1.142857	1.3333333	1.6	2
25	23	1.210526	1.5333333	1.916666	2.555555
50	66	1.65	2.75	3.666666	4.714285
100	201.8	1.667769	1.8685185	2.059183	2.554430

Fig (3) below shows the variation of speedup with different number of processors i.e. from 2 to 8.

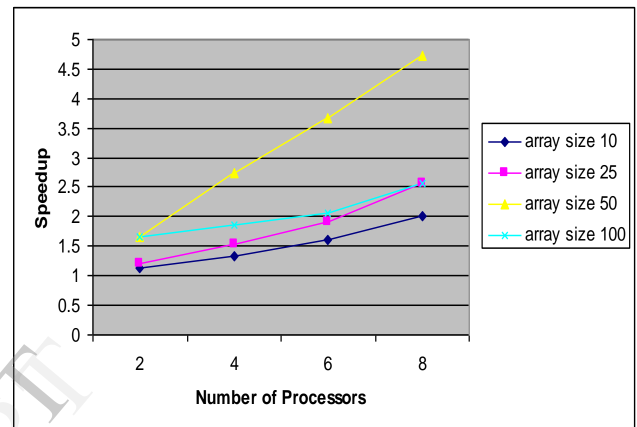


Fig 3. Show Speedup for different numbers of Processors.

E. *Efficiency:*

Having measured the times for T_1, T_2, T_4, T_6, and T_8 we have calculated the efficiency for the different values we have got, using the formula (3):

$$Efficiency = (Speedup / P) \qquad (3)$$

Where P is the number of processors used. The results obtained are tabulated in table 10 [5].

Table 10. Efficiency data.

Array size	1	2	4	6	8
10	10.74	0.571429	0.333333333	0.26666667	0.25
25	91.34	0.605263	0.383333333	0.31944444	0.31944444
50	196.48	0.825	0.6875	0.61111111	0.58928571
100	607.1	0.833884	0.46712963	0.34319728	0.3193038

Fig (4) below shows the variation of efficiency with different number of processors i.e. from 2 to 8.

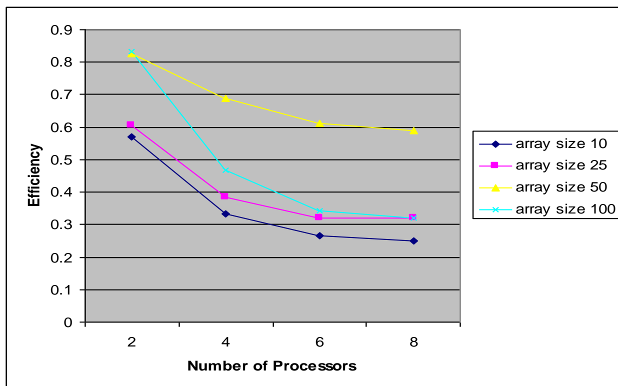


Fig4 Show the efficiency for different number of Processors.

Fig (5) below shown the time for 100 x 100 Matrix Multiplications with different number of processors i.e. from 2 to 8.

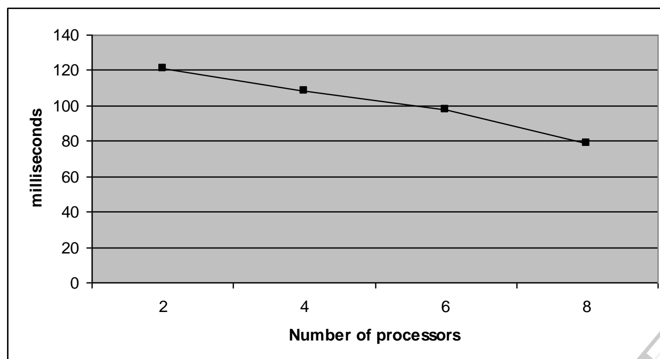


Fig 5. Shows Time for 100 x 100 Matrix Multiplications.

IJERT

.

VI. CONCLUSIONS:

Consolation has been drawn, that from fig. 5 shown the performance of our program. As when we increase the number of processors the time for multiplying 100 x 100 array has decrease of about 2.5 times which can be reflected in the Speedup diagram in Fig. 3. The order of the algorithm in the "big O" notation is $O(N^3)$. The program is scalable, which means that when increase the number of processors we have a reasonable speedup and a good efficiency. The program can handle large dimensions of arrays. The program is a user friendly as the user can decide the size of his array and the number of processors he/she wants to multiply that array on.

REFERENCES

[1] G. H. Golub and C.F. Van Loan, "Matrix Computations", 3rd Ed, The Johns Hopkins University Press, 1996.
[2] R.Stephens,"Visual Basic 2008 Programmer's preference (Programmer to Programmer)", WROX Publishing, First Edition (2008).
[3] Michael Halvorson, "Microsoft Visual Basic 2008 Step by Step", first edition 2008.
[4] J. M. Ortega, "Introduction to Parallel and Vector Solution of Linear Systems", Plenum Press, 1988.
[5] J. M. Ortega and C. H. Romine, "The ijk forms of factorization methods II: parallel systems", Parallel Computing 7:149-162, 1988.