PARALLEL MATRIX MULTIPLICATION IMPLEMENTATION IN DISTRIBUTED ENVIRONMENT THROUGH RMI

[1]NAKKEERAN.M and [2]Dr.RM.CHANDRASEKARAN.

[1]M.E-Computer Science and Engineering (Final Year), Annamalai University, Chidambaram, India-608002.

[2]Professor, Department of Comp. Science and Engg, Annamalai University, Chidambaram, India-608002.

E-mail: nareek_sm@yahoo.co.in, aurmc@sify.com.

ABSTRACT

This paper proposes to solve the parallel matrix multiplication implementation in a distributed environment through RMI based on JAVA threads. The application distributes the products of rows and columns on different machines. One server and two clients are run to find the product of matrix multiplication. The server distributes the determine blocks of rows and columns on the registered clients. The clients return their product blocks to a server, which calculate the final product of matrix multiplication. Applications of this type will allow loaded servers to transfer part of the load to clients to exploit the computing power available at client side. The time of matrix multiplication with size of 200 X 200 and 500 X 500 is reduced by 61.34% and 36.67% respectively by using 2-clients in comparison to sequential program and this time can be decreased more in the case of increasing the number of clients.

Keywords: *Distributed Environment, RMI, Java Threads.*

I. INTRODUCTION AND RELATED WORKS

Matrix multiplication (MatMul) is one of the most fundamental operations in linear algebra operation. MatMul serves as the primary operational component in many different algorithms, including the solutions for systems of linear equations of the determinant a matrix, and the transitive closure of a graph. Due to its fundamental importance, much effort has been devoted to studying and implementing MatMul which has been included in several libraries. Many MatMul algorithms have been developed for parallel systems [1-4].

Traditional methods for distributed application are decomposed the entire task, which introduces various overheads. The most important are the communication and load balancing overheads. For example, partitioning MatMul into sub-matrix blocks and decomposing the MatMul operation are often technology dependent. Applying special implementations for sub-matrix

blocks may improve performance since the workload of sub-matrix operations may vary. The information about the workload of a task for matrix operation may not be available at compile time or even at the time of initiating subroutines. It may be available only after these routines have been executed. Since this increases the complexity of load balancing, it is often ignored.

Most parallel algorithms are optimized based on the characteristics of the targeting platform. The PC cluster computing platform has recently emerged as a viable alternative for high-performance and low-cost computing [2].

Generally, the PCs in a cluster have a lot of resources that can be used simultaneously. They have relatively weak communication capabilities. They lack high performance implementation support for data communications compared to supercomputers. They only support some communication channels implemented by software that capitalizes on Ethernet connections. MatMul operations are embedded in many host programs.

The main reason for using parallel processing is to reduce the computation time required for what would otherwise be very long-running programs. Because poorly parallelized code tends to offer little performance benefit, there

is great incentive to ensure that parallel programs are highly optimized. Unfortunately, a lack of sufficiently accurate and easy-to-use performance prediction methods for parallel programs has necessitated resort to a very time-consuming, which modifies design cycle to achieve this [5].

The biggest price we had to pay for the use of a PC cluster was the conversion of an existing serial code to a parallel code based on the message-passing philosophy. The main difficulty with the message passing philosophy is that one needs to ensure that a control node (or master node) is distributing the workload evenly between all the other nodes (the compute nodes). Because all the nodes have to synchronize at each time step, each PC should finish its calculations in about the same amount of time. If the load is uneven (or if the load balancing is poor), the PCs are going to synchronize on the slowest node, leading to a worst-case scenario. Another obstacle is the possibility of communication patterns that can deadlock [5]. A typical example is if PC A is waiting to receive information from PC B, while B is also waiting to receive information from A.

JAVA provides Remote Method Invocation (RMI) to allow one JAVA Virtual Machine (VM) to invoke methods running on another VM. RMI applications are often comprised of two separate programs, which are server and client. A typical server application creates some remote objects, makes references to them accessible, and waits for clients to invoke methods on these remote objects. A typical client application gets a remote reference to one or more remote objects in the server and then invokes methods on them. RMI provides the mechanism by which the server and the client communicate and pass information back and forth. Such an application is sometimes referred to as a distributed object application [6-8].

The related work [1- 4] deals with moving application from one machine to another in a set of machines. In the previous distributed modes it is not very easy for a programmer to write an application a part of which can be executed on remote side and result of computation can be combined in original program to compute the final result, i.e. there is no method level distributed available. We have applied object mobility to LAN architecture. This allows development of applications that can be easily load balanced.

In this paper we study implementation for matrix multiplication on distributed systems using RMI based on JAVA threads, which distribute the load between the server and the clients. This system provides the user a level of control over the distribution of the program

The rest of this paper is organized as follow. Section 2 introduces the techniques of matrix multiplication. Section 3 presents the implementation architecture. Section 4 shows the experimental setup and results. Finally the conclusion is provided in Section 5.

II. MATRIX MULTIPLICATION TECHNIQUES

Consider the product of matrix multiplication C = A*B where A, B, C are matrices of size n X n as shown in Figure 1.

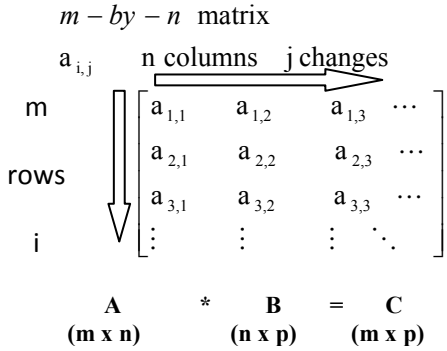


Figure 1 Matrix Multiplication

For notation purpose it has been referred has m-by-n matrix as "(m*n)". Similarly, the multiplication of an m-by-n matrix with n-by-p matrix is denoted as "(m x n) *(n x p)". By Initial assumption, (n x kn) * (kn x n) Matrix Multiply would require, at minimum, k-times the duration of an (n x n) * (n x n) Matrix Multiply. Based on assumption, the notation is given as:

$$[A_{11}] \times [B_{11}] = [A_{11} * B_{11}]$$

Whereas:

$$[A_{11}A_{12\ldots}\ A_{1f}] \times [B_{11}] = [A_{11}*A_{11} + A_{12}*A_{21} + \ldots + A_{1f}*B_{f1}]$$
$$[B_{21}]$$
$$[\ldots\ldots]$$
$$[B_{f1}]$$

there would ultimately be k-times the number of $A_{1x}*B_{x1}$ operations (as well as k-times the number of addition operations, assuming that the resulting matrix had its element values initialized to the value of zero) involved in an (n x kn) * (kn x n)

Matrix Multiply, versus an (n x n) * (n x n) Matrix Multiply, based on the example given above.

Next subsections present the various methods that used to find the matrix multiplication.

2.1 Sequential Method

The matrix operation derives a resultant matrix by multiplying two input matrices a and b, where matrix a is a matrix of N rows by P columns and matrix b is of P rows by M columns. The resultant matrix c is of N rows by M columns. The serial realization of this operation is quite straightforward as listed in the following:

```
for (k=0; k<M; k++)
  for (i=0; i<N; i++){
    c[i][k]=0.0;
      for(j=0;j<P;j++)
        c[i][k]+ =a[i][j]*b[j][k];
}
```

The above algorithm requires n^3 multiplications and n^3 additions, leading to a sequential time complexity of O (n^3).

2.2 Master Slave Model

The matrix multiplication algorithm is implemented in MPI using the straight forward algorithm based on the master-slave paradigm [8]. Master Slave computing paradigm, which also called replicated slave computing is consists of broken many computational problems into smaller pieces that can be computed by one or more processes in parallel. The computations are fairly simple, which usually compute-intensive, region of code. The size of loop is quite long.

Figure 2. shows a Master-Slave computing paradigm, a Master process takes the work performed in the computationally intensive loop and divides it up into a number of tasks that it deposits into a task bag. One or more processes, known as slaves, grab these tasks, compute them and place the results back into a result bag. The Master process collects the results as they are computed and combines them into something meaningful such as a vector product.

Message Passing Interface is a widely used standard for writing message-passing programs to establish a practical, portable, efficient, and flexible standard for message passing. The master creates a set of random matrices. Each matrix multiplication job consists of pair of matrices to be multiplied. For each job the master sends one entire matrix to each slave and distributes the rows of the matrix among the slaves. In this way matrix multiplication jobs are computed in a parallel fashion as follow;

1. The master process for each job, which sends the first matrix from the pair of matrices multiplication joined with a certain number of rows of the other matrix depending on the number of slaves.

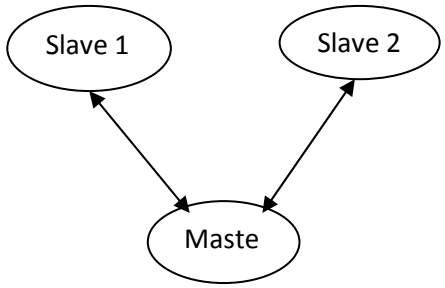


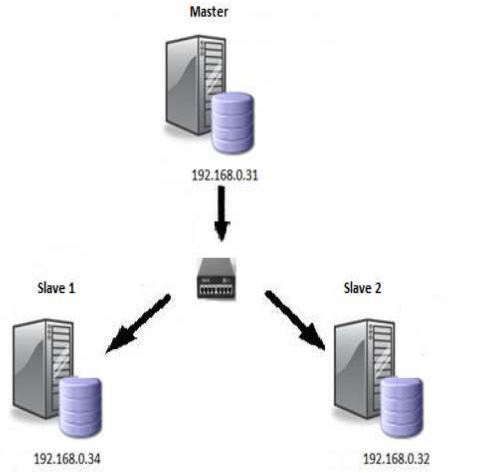
Figure 2. Master Slave

2. Each Slave process receives one entire matrix and a certain number of rows of the other matrix based on the number of slaves. Thus it computes the rows of the resulting matrix and sends it back to the master.
3. The master process collects the rows of resulting matrix from the slaves.

III. IMPLEMENTATION ARCHITECTURE

In this work the integration of sequential development and master-slave models to calculate MatMul by using RMI Java threads. In this proposed model, the server determines the distributed numbers of rows from the first matrix and the columns of the second matrix depending on the balance of workload on registered clients. For example, when n = 20, which mean the size of 2-matrix multiplication is 20 X 20. The Server splits the job into 10 tasks (say) which mean each job gets 2 numbers of rows. If the registered client's number is 2 then server distribute the 10 jobs in to 5 each to the distributed clients. Each client multiplies 10 rows with its 10-columns and returns its product to the server in one thread. In the case of n = 21, the above scenario will be done and the twenty first row will multiply with the columns of second matrix on the last client. In the heterogeneous matrix, which mean the number of rows is not the same number of columns, the multiplication will done because the load

distribution depend on the number of rows in the first matrix with its columns of the second matrix. One server and 2-clients are used to implement



MatMul as shown in Figure 3. RMI implementation algorithm is shown as follow;

Figure 3 RMI Server - Client Architecture

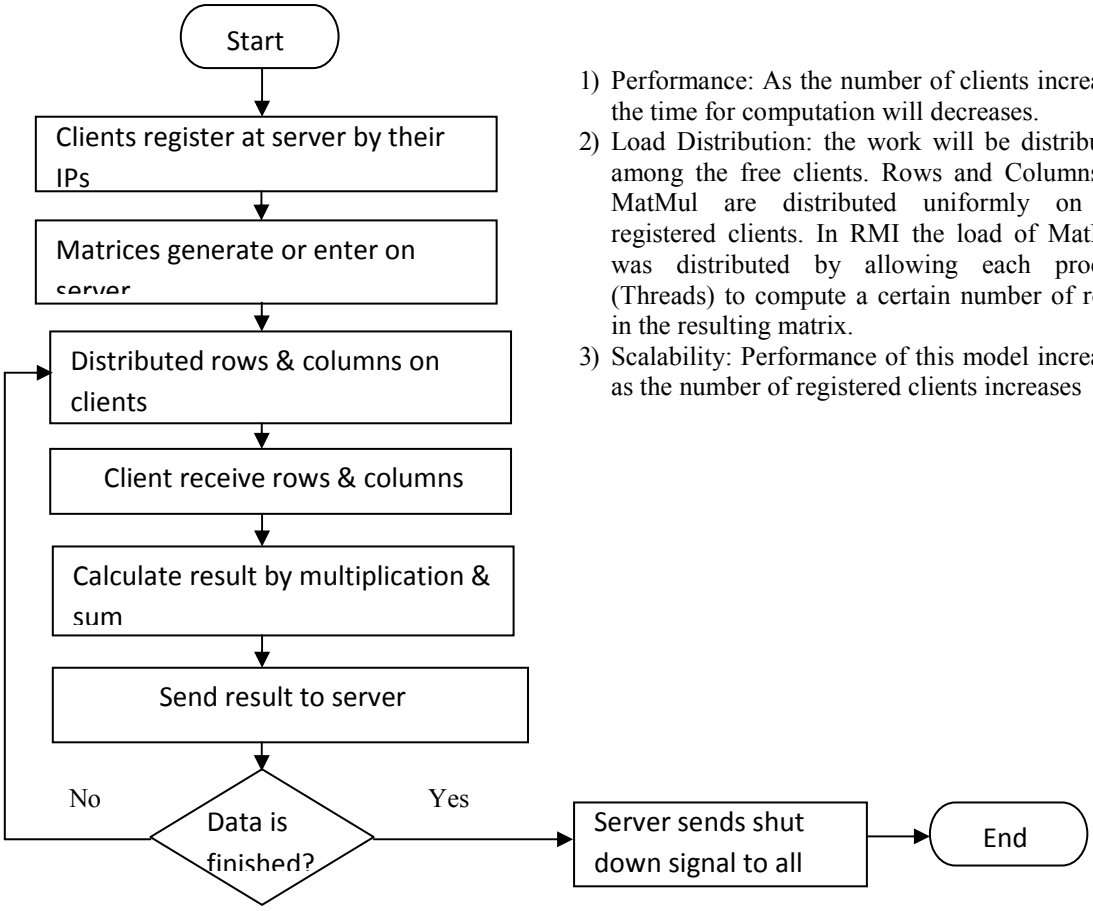


Figure 4 Flow Diagram of MatMul Algorithm

Step 1 Client discovery;
Client will register itself with the server to take a task from it
Step 2 Generate Matrices;
Server generates two matrices randomly or getting them as inputs
Step 3 Data distribution
Server will distribute number of rows from first matrix and its corresponding columns of the second matrix on clients that it has been registered using Java threads.
Step 4 Sever waits for result;
Server will waits results from clients and append it in result matrix.
Step 5 Results collecting;
Server will collect the results that sent by each client and compute the time that taken by each client and compute all time taken in this process
Step 6 Shutdown;
Finally, server will send shutdown to all clients.

Figure 4 shows the flow diagram of the above algorithm. In this work following issues are addressed:

1) Performance: As the number of clients increases the time for computation will decreases.
2) Load Distribution: the work will be distributed among the free clients. Rows and Columns of MatMul are distributed uniformly on all registered clients. In RMI the load of MatMul was distributed by allowing each process (Threads) to compute a certain number of rows in the resulting matrix.
3) Scalability: Performance of this model increases as the number of registered clients increases

IV. EXPERIMENTAL SETUP AND RESULTS

The analytical performance model describes the computational behaviour of matrix multiplication implementations. Consider the matrix multiplication product C = A*B where the size of matrices A, B, and C are n X n. Implementing the MatMul by using JDK 1.6 on LAN of 3-PC (1- Server and 2 Clients) Intel Core 2 Duo Processor, 2.93 GHz with 2 GB RAM.

Table 1 show the tabulated result of this measuring of Serial Vs Parallel program is a way to assess how well and efficient this development have been divided the big application into small modules cooperating with each other in parallel. The most easily recorded metric of performance is the execution time by measuring the time consumed in the execution of parallel program with its Client Connectivity makes directly measures its effectiveness. To find out how much better for our proposed does on the parallel machine, which it compared with running an application on only one processor. The ratio of execution time is taken into the account, which is called the speedup.

Speedup (S) = (Serial Execution Time) / (Parallel Execution Time)

$$\text{Speedup } (S) = T(1)/T(N)$$

Where T (N) represents the execution time taken by the program running on N processors, and T (l) represents the time taken by the best serial implementation of the application measured on one processor.

$$\text{Efficiency, } E(\%) = S / p$$

Where S is the Speedup and p is the number of processors.

Here, the no of processors involved is three which accounts for 1 –Server and 2 Client. Table 1 shows speedup is 1.84 and 1.10 for MatMul with the size of 200 X 200 and 500 X 500 respectively (or) reduced the Multiplication time by 61.34% and 36.67% respectively.

V. CONCLUSION AND FUTURE WORK

In this paper the parallel matrix multiplication is implemented and analyzed on distributed systems. This mechanism will make it easier to automatic migrate the computation load to client. It has been shown that execution time decreases by 61.34% and 36.67% for MatMul with size of 200 X 200 and 500 X 500 respectively. Future work will apply this implementation on any practical application like weather prediction, databases systems, data compression and others with increasing the numbers of clients.

Table 1 shows the comparison result of Serial Vs Parallel program with number of client connected in Distributed Environment.

Size of the Matrix (n*n)	Sequential Program Time (s)	Parallel Connectivity		
		Two number of Clients Connected		
		Time (s)	Speedup (S)	Efficiency E, (%)
200*200	26.36	14.28	1.84	61.34
500*500	124.35	112.89	1.10	36.67

REFERENCES:

[1] Tinerti, F., Quijano, A., Giusti, A., Luque, E. "Heterogeneous networks of workstations and the parallel matrix multiplication " Proceedings of the Euro PVM/MPI 2001, Springer-Verlag, erlin, pp. 296-303. 2001.

[2] T. Typou, Vasilis stefanids, P. Michailidis, and K. Margaritis "Implementing Matrix Multiplication on An Cluster of Workstation", 1st IC-SCCE, Athens, 8-10, Sep. ,2004.

[3] Ju-wook Jang, Seonil Choi and Viktor K. Prasanna, *Energy-Efficient Matrix Multiplication on FPGAs,* IEEE Transactions on VLSI (TVLSI), Vol. 13, No. 11, pp. 1305-1319, November 2005.

[4] Carrio and Geleertner "How to write Parallel Programs, A Guide to the Perplexed" ACM Computing Serveys, Vol. 21, No. 3, Sep. 1999.

[5] Coulouris, et al, Distributed Systems Concepts and Design, 3rd Edition, Addision Wesely, Person Education 2001.

[6] Maassen, J., Nieuwpoort, R. V., Veldema, R., Bal, H., and Kielmann, T., Wide-Area Parallel Computing in Java, Proceedings of the ACM Conference on Java Grande, San Francisco, CA,(1999), pp. 8-14, 1999.

[7] Grama, A., Gupta, A., Karypis, G., and Kumar, V., Introduction to Parallel Computing (Second Edition), Pearson Education Limited, Harlow, England, (2003), pp. 345-349, 2003.

[8] Wilkinson, B., Allen, Parallel Programming: Techniques and Applications Using Networking.