Evaluation of Matrix Multiplication on an MPI Cluster

Sherihan Abu ElEnin, Mohamed Abu ElSoud

Faculty of computers and Information, Mansoura university, Egypt

moh_soud@mans.edu.eg

***Abstract*-- An mpi cluster is a group of computers which are loosely connected together to provide fast and reliable services. Clusters use in many scientific computing, such as the matrix multiplication. Our experiment is based on the master – slave model in homogenous computers to compute the performance of experiment. We compute the execution time for many examples to compute the speed up. The developed performance model has been checked and it has been shown that the parallel model is faster than the serial model and the computation time was reduced.**

***Index Term*-- Matrix multiplication, Cluster of computers, Message Passing Interface, Master/slave algorithm, Performance evaluation.**

1. INTRODUCTION

The use of cluster of PC for high performance computing tasks has become very popular recently, because they provide a high performance to a much lower price than dedicated multicomputers. [1] New networking techniques have been developed in the past few years to increase performance and to combine the relatively cheap high processor performance with adequate networking performance. Message passing on cluster computers is the main programming paradigm used for high performance scientific computing. [2] 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 measure-modify design cycle to achieve this. Message Passing Interface is a specification for message passing libraries, designed to be a standard for distributed memory, message passing, parallel computing. [3] The goal of the Message Passing Interface simply stated is to provide a widely used standard for writing message-passing programs. The interface attempts to establish a practical, portable, efficient, and flexible standard for message passing.

In this paper we study implementation for matrix multiplication on a cluster of computers and develop a performance model for this implementation. This paper concentrates on the evaluation of cluster performance. It is organized as follows. In section 2, we define what is cluster? Why we choose MPI? How to implement MPI? In section 3, Matrix Multiplication (MM) is discussed in details. In section 4, the algorithm which our experiment depends on it and the codes of master and slaves. In section 5, we describe the performance model of Matrix Multiplication. In section 6, the results of experiment are shown and explained in details. Finally, the conclusion of this paper and the future work for application are given in section 7.

2. PRELIMINARIES

Clusters have been becoming a mainstream architecture of high-performance computing systems for their high scalability and performance/cost ratio. Nowadays, clusters have been widely used in the fields of technical computing, Internet service, and database applications. [4] Workstation cluster systems are very attractive for distributed and parallel processing, since it is easy to obtain parallel processing environment. [5] Cluster is a system comprised of two or more individual computers or systems (often called "nodes") which work in conjunction with each other to execute applications or perform other work. Cluster

systems can deliver similar or better performance and reliability than traditional mainframes, supercomputers and fault-tolerant systems with a much lower hardware cost.

The PC clusters main advantages compared to commercial supercomputers and computers are the following: [6]

1. Low cost: *from 3 to 10-15* times less expensive compared to commercial computers and supercomputers.
2. Scalability: clusters have a very flexible architecture; you can build systems ranging from few computers (3 or 4) up to many tens of units (64 - 128) with an almost proportional cost.
3. Ease of upgrading and maintenance: a cluster can be progressively updated by repairing or upgrading its components with the *commodity hardware*, easily available on PC market.
4. Standard parallel platform: parallel programming software available for clusters is the same present on commercial supercomputers and computers (ex. PVM and MPI). So the programs can be compiled on both platforms without any change.
5. Open Source software: the most of the software installed on clusters is open-source, freely available, extremely tested and reliable, and continuously updated.

Parallel virtual Machine (PVM) and Message Passing Interface (MPI) are the two systems for programming clusters. Both PVM and MPI are systems designed to provide users with libraries for writing portable, heterogeneous, MIMD programs. The Message Passing Interface (MPI) is a set of API functions enabling programmers to write high performance parallel programs that pass messages between serial processes to make up an overall parallel job.

There are 10 reasons to prefer MPI over PVM Plain list:[7]

1. MPI has more than one freely available, quality implementation.
2. MPI defines a 3rd party profiling mechanism.
3. MPI has full asynchronous communication.
4. MPI groups are solid, efficient, and deterministic.
5. MPI efficiently manages message buffers.
6. MPI synchronization protects 3rd party software.
7. MPI can efficiently program MPP and clusters.
8. MPI is totally portable.
9. MPI is formally specified.
10. MPI is a standard.

There are various implementations available for MPI; the popular are LAM/MPI & MPICH. MPICH is a freely available, complete implementation of the MPI specification, designed to be both portable and efficient. MPI can be thought of as a small specification, because any complete implementation need only provide the following operations:

- MPI_INIT : MPI_Init(&argc,&argv);
- MPI_COMM_SIZE : MPI_Comm_size(MPI_COMM_WORLD,&numtasks);
- MPI_COMM_RANK : MPI_Comm_rank(MPI_COMM_WORLD,&taskid);
- MPI_SEND : MPI_Send(&offset, 1, MPI_INT, dest, mtype, MPI_COMM_WORLD);
- MPI_RECV : MPI_Recv(&offset, 1, MPI_INT, source, mtype, MPI_COMM_WORLD, &status);
- MPI_FINALIZE : MPI_Finalize();

Clusters are composed of commodity hardware and software components. The hardware components include standalone computers (i.e., nodes) and networks. Cluster nodes can be PCs, workstations, and SMP's. Networks used for interconnecting cluster nodes can be local area networks such as Ethernet and Fast Ethernet, system area networks such as Myrinet and Quadrics switches, or upcoming InfiniBand communication fabric. [8] Various operating systems, including Linux, Solaris, and Windows, can be used for managing node resources. The communication software can be based on standard TCP/IP or user-level messaging layers such as VIA. Suitable references for this unit are [9] and [10] System-level middleware offers Single System Image (SSI) and high availability infrastructure for processes, memory, storage, I/O, and networking. The single system image illusion can be implemented using the hardware or software infrastructure.

Our cluster platform consists of:

1) Hardware Components

- Nodes : 2 – 12 IBM PCs as in figure (1).

(Intel Pentium IIII 2.4 GHz processor, Intel RAM 256 MB SDRAM).

- Interconnection Network : Fast Ethernet .

2) Middleware : Message passing libraries (MPI).

3) Software Components

- Operating System : Red Hat Linux 9
- Tools : Programs written in C++.

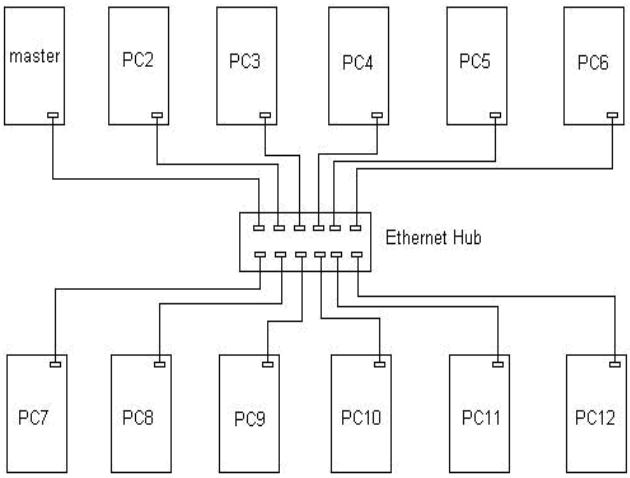


Fig. 1. Cluster Platform

3. MATRIX MULTIPLICATION

Matrix multiplication (MM) is an important linear algebra operation. A number of scientific and engineering applications include this operation as a building block. Due to its fundamental importance, much effort has been devoted to studying and implementing MM. MM has been included in several libraries. Many MM algorithms have been developed for parallel systems. Express multiplication of two matrices as dot product of vectors of matrix rows and columns. That is to compute some cell c_{ij} of matrix C, we take the dot product of row i of matrix A with column j of matrix B as figure (2).

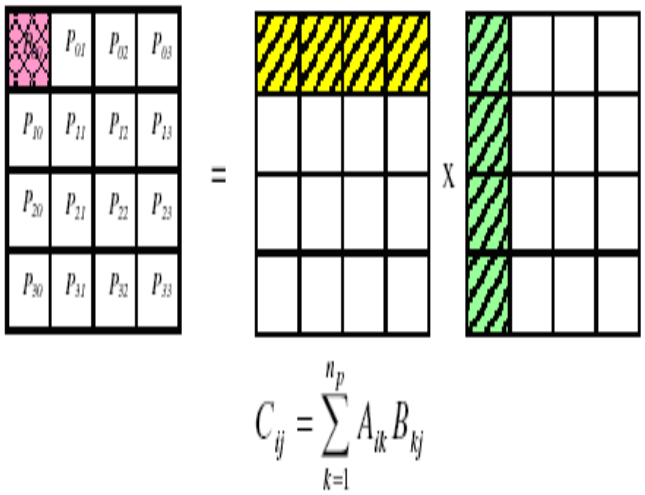


Fig. 2. Matrix multiplication

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. [11] 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. 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.

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

Its algorithm requires n^3 multiplications and n^3 additions, leading to a sequential time complexity of $O(n^3)$. Let's consider what we need to change in

order to use MPI.

4. MASTER / SLAVE MODEL

In 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. This model is described in figure (3).

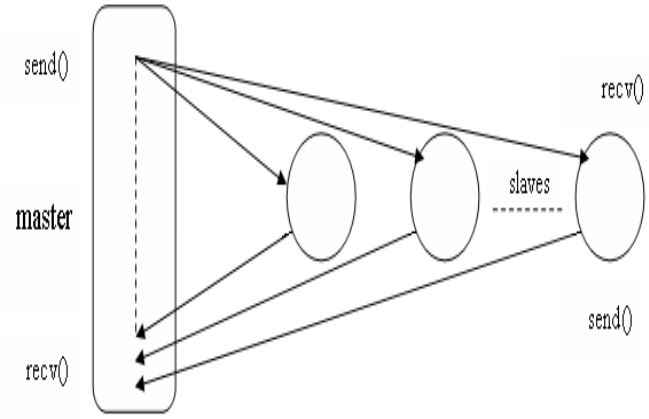


Fig. 3. Master / Slaves Model

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.

4.1 Control Flow of Matrix Multiplication

1) The Master process for each job first sends one matrix of the job pair, and certain number of rows of the other matrix based on the number of slaves and the offset of the row within the cut matrix.[13]
2) Each Slave process receives one entire matrix of the job and receives a certain number of rows of the other matrix based on the number of slaves. It then computes the rows of the result matrix for the number of rows of the cut matrix it received and sends it back to the master.
3) The master process now collects rows of the result matrix from the slave along with its offset in the result matrix.

The algorithm employs a master process and an array of independent slaves processes. Slaves process i computes row i of result matrix c. In order to compute it, it needs row i of source matrix a and the entire source matrix b. Each slave receives these values from a separate master process. The slave then computes its row of results and sends them back to the master. The master process initiates the computation and gathers and prints the results. The master also first sends each slaves the appropriate row of a and all of b. Then the master waits to receive a row of c from every slave.

4.2 The master task code

```
/******** master task ********/
if (taskid == MASTER) {
 printf("Numberofslavestasks= %d¥n",numslaves);
 for (i=0; i<NRA; i++)
 for (j=0; j<NCA; j++)
    a[i][j]= i+j;
 for (i=0; i<NCA; i++)
    for (j=0; j<NCB; j++)
        b[i][j]= i*j;
 /* send matrix data to the slaves tasks */
 averow = NRA/numslaves;
 extra = NRA%numslaves;
 offset = 0;
 mtype = FROM_MASTER;
 for (dest=1; dest<=numslavess; dest++) {
  rows = (dest <= extra) ? averow+1 : averow;
  printf("sending %d rows to
task %d¥n",rows,dest);
  MPI_Send(&offset, 1, MPI_INT, dest, mtype,
           MPI_COMM_WORLD);
  MPI_Send(&rows, 1, MPI_INT, dest, mtype,
           MPI_COMM_WORLD);

MPI_Send(&a[offset][0],rows*NCA,PI_DOUBLE,
           dest,mtype, MPI_COMM_WORLD);
  MPI_Send(&b, NCA*NCB, MPI_DOUBLE,
dest,
           mtype, MPI_COMM_WORLD);
  offset = offset + rows;
  }
 /* wait for results from all slaves tasks */
 mtype = FROM_SLAVES;
```

```
for (i=1; i<=numslaves; i++) {
  source = i;
  MPI_Recv(&offset, 1, MPI_INT, source, mtype,
                MPI_COMM_WORLD, &status);
  MPI_Recv(&rows, 1, MPI_INT, source, mtype,
                MPI_COMM_WORLD, &status);

      MPI_Recv(&c[offset][0],rows*NCB,PI_DO
      UBLE,
     source,mtype,MPI_COMM_WORLD,
      &status);
}
```

4.3 The slaves task code

```
/********** slaves task ************/
if (taskid > MASTER) {
   mtype = FROM_MASTER;
   MPI_Recv(&offset,1,MPI_INT,MASTER,
mtype,
                MPI_COMM_WORLD, &status);
   MPI_Recv(&rows,1, MPI_INT, MASTER,
mtype,
                MPI_COMM_WORLD, &status);
  MPI_Recv(&a,rows*NCA,MPI_DOUBLE,
  MASTER,mtype,MPI_COMM_WORLD,
&status);
  MPI_Recv(&b,NCA*NCB,MPI_DOUBLE,
  MASTER,mtype,MPI_COMM_WORLD,
    &status);
  for (k=0; k<NCB; k++)
   for (i=0; i<rows; i++) {
     c[i][k] = 0.0;
     for (j=0; j<NCA; j++)
       c[i][k] = c[i][k] + a[i][j] * b[j][k];
  }
  mtype = FROM_SLAVES;
  MPI_Send(&offset,1, MPI_INT, MASTER,
mtype,
            MPI_COMM_WORLD);
  MPI_Send(&rows, 1, MPI_INT, MASTER,
mtype,
             MPI_COMM_WORLD);
  MPI_Send(&c,rows*NCB,MPI_DOUBLE,
          MASTER,mtype,
MPI_COMM_WORLD);
```

5. MEASURING PERFORMANCE

Performance is of paramount importance in parallel programming. The reason we are in the game of writing parallel programs is either to solve a problem faster than on a serial computer, or to solve a larger problem than could previously be done. [12]

Measuring the performance of a parallel program is a way to assess how well and how efficient our efforts have been at dividing the application into modules cooperating with each other in parallel.

The most visible and easily recorded metric of performance is the execution time. By measuring how long the parallel program needs to run to solve our problem, we can directly measure its effectiveness. To find out how much better our program does on the parallel machine, compared to a program running on only one processor, taking the ratio of the two is a natural solution. This measure is called the speedup, and is associated with the number of processors in the machine.

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

Speedup = T(1) / T(N)

Where T(N) represents the execution time taken by the program running on N processors and equal communication time plus computation time, and T(1) represents the time taken by the best serial implementation of the application measured on one processor.

We will refer to a program has having a speedup of x on N processors, for example. We will also investigate other performance metrics. Each one will provide us with a different dimension, or different view point to observe the "performance" of the system. The metrics are all related to each other, but highlight different components of the dynamics at work during the execution of the parallel program.

In our experiment we addressed the following issues: [13]

1) Performance: As the number of slaves increases the time taken for job computation decreases.

2) Load Distribution: the job will be distributed among the free slaves. In RMI load of multiplying the matrices was distributed uniformly among all the slaves. In MPI the load of multiplying the matrices was distributed by allowing each process to compute only a certain number of rows of the result matrix.

3) Scalability: We tested the scalability of the master-slave computing paradigm with one master

and multiple slaves in MPI. Performance of MPI increases as the number of slave processes increases.

We develop an analytical performance model to describe the computational behavior of the parallel matrix multiplication implementations. We consider the matrix multiplication product C = A×B where the size of matrices A, B, and C are n×n. The number of computers in the cluster is denoted by p and we assume that p is power of 2.

6. RESULTS

We indicated to the scalability of the parallel solution with increasing number of processors and problem size. So the communication time will be high with increasing problem size and system size.

In our experiment, If number of processors are increased, the speedup is increased also.

The execution time = communication time + computation time.

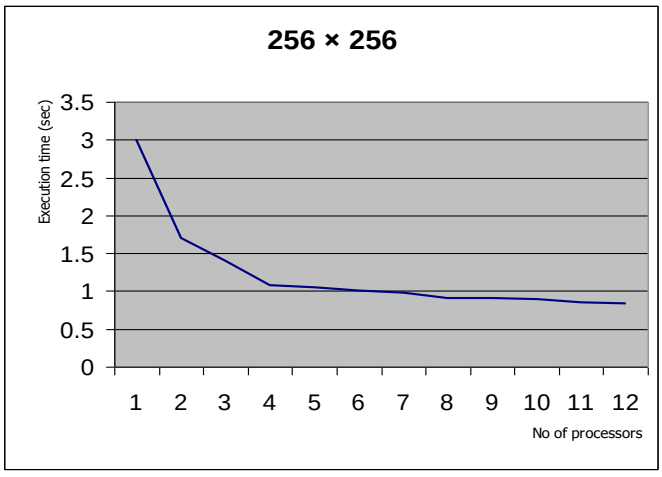
Multiplication code; //Computation.

MPI_Gather(); //Communication.

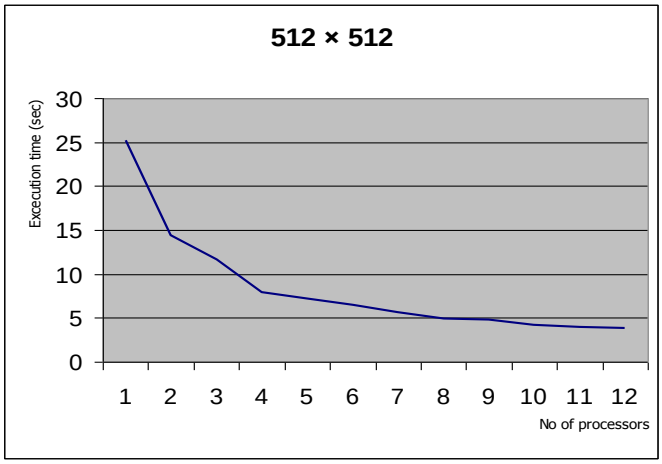
The computation time is stable but the communication time is not stable and made overhead when number of nodes is increased.

In Matrix 256 × 256: The execution time in one processor is 3.007 seconds, but when It executed in different number of processors the time is decreased as shown in figure (4a).

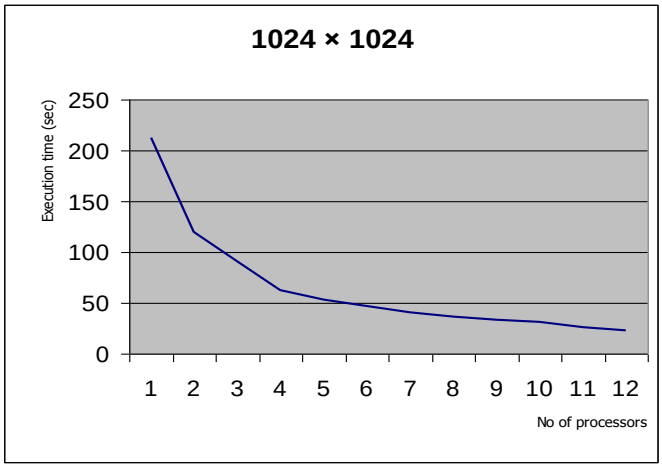
In Matrix 512 × 512: The execution time in one processor is 25.258 seconds, but when It executed in different number of processors the time is decreased as shown in figure (4b).



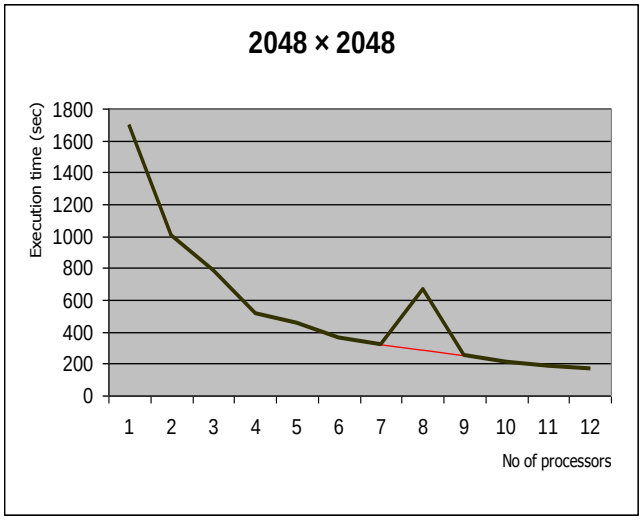
(a) The execution time of MM of size 256



(b) The execution time of MM of size 512



(c) The execution time of MM of size 1024



(d) The execution time of MM of size 2048

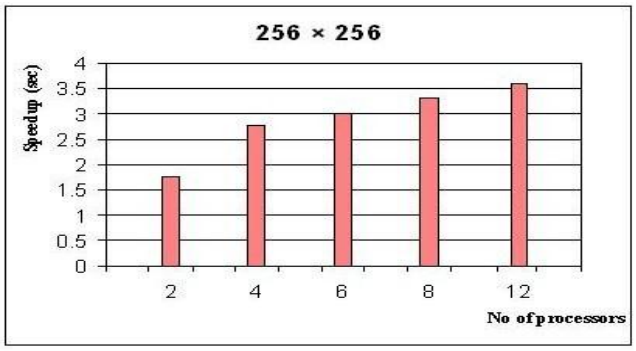
Fig. 4. Execution time of our experiments

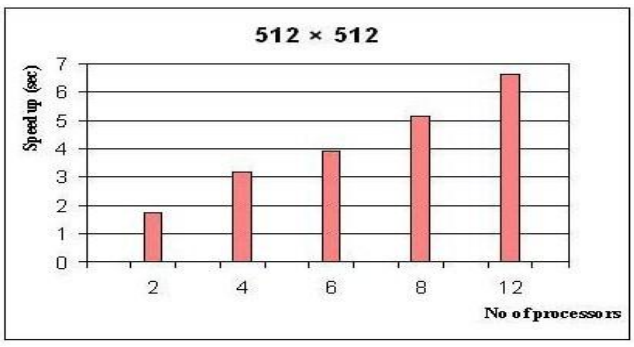
In Matrix 1024 × 1024: The execution time in one processor is 212.112 seconds, but when It executed in different number of processors the time is decreased as shown in figure (4c).

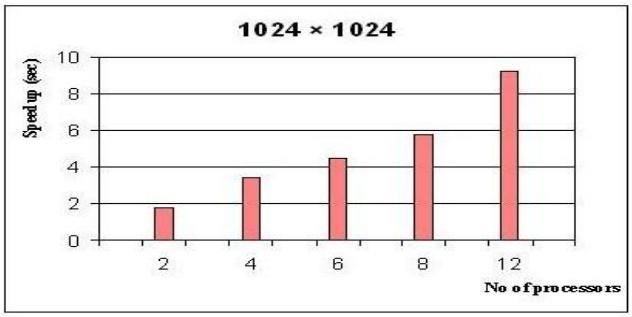
In Matrix 2048 × 2048: The execution time in one processor is 1697.97 seconds, but when It

executed in different number of processors the time is decreased, except the execution time in 8 processors there are overhead in communication time so the time is increased as shown in figure (4d).

We compare our results with **"IMPLEMENTING MATRIX MULTIPLICATION ON AN MPI CLUSTER OF WORKSTATIONS"** Theodoros Typou, Vasilis Stefanidis, Panagiotis Michailidis and Konstantinos Margaritis, Parallel and Distributed Processing Laboratory (PDP Lab),University of Macedonia, Greece, International Conference "From Scientific Computing to Computational Engineering", September, 2004; as shown in figure (5a and 5b) with platform cluster of homogeneous workstations connected with 100 Mb/s Fast Ethernet network and the homogeneous cluster consists of 32 Pentium workstations based on 200 MHz with 64 MB RAM..







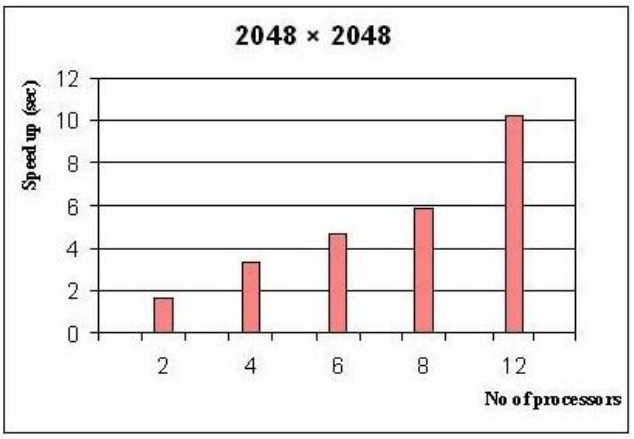
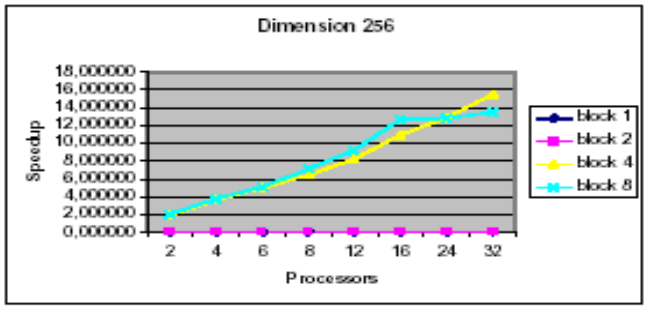
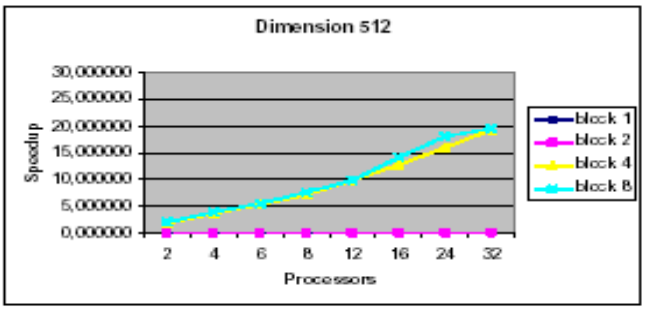
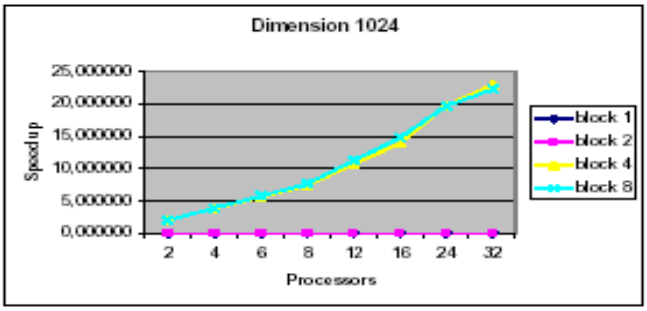


Fig. 5. a Our experimental speedup of MM







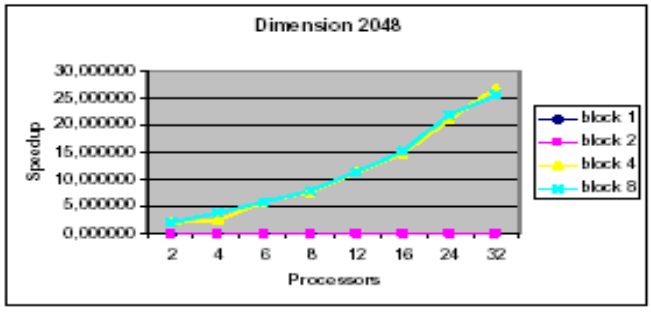


Fig. 5. b Another experimental speedup of MM

7. CONCLUSIONS AND FURTHER WORK

In this paper we implemented and analyzed the parallel matrix multiplication implementations on a cluster of computers and we used master/slaves algorithm to achieve high performance. It has been shown that the performance of parallel model is reliable and safe time more than the serial model. Because of high performance of mpi cluster, more applications can depend on this system to save money and time. Our experiment does not achieve the peak point which the performance can reach the high value because the number of PCs is not more. If the number of nodes is between 4 to 100, we think the performance can be determined well than our model.

Finally, the proposed matrix multiplication implementations could be executed on a cluster of heterogeneous computers in the future. Another algorithm can be used to achieve better results than our results.

REFERENCES

[1] M.Poeppe,S.Schuch,T.Bemmerl; "a message passing interface library for inhomogeneous coupled clusters", Parallel and Distributed Processing Symposium, 2003. Proceedings. International Volume , Issue , 22-26 April 2003.

[2] D.A.Grove and P.D. Coddington ;"Communication Benchmarking and Performance Modelling of MPI Programs on Cluster Computers", Distributed and High Performance Computing Group represents work by the DHPC members at the University of Adelaide 2 November 2003.

[3] http://www.mhpcc.edu/training/workshop/mpi/MAIN.html

[4] Qinghua Ye, Limin Xiao, Dan Meng, Wen Gao, Yi Liang, Ying Jiang: "A Powerful Service-Based Application Management System for Clusters" , 31st International Conference on Parallel Processing Workshops (ICPP 2002 Workshops), 20-23 August 2002, Vancouver, BC, Canada. IEEE Computer Society 2002, ISBN 0-7695-1680-7.

[5] Hiroyuki Okuno ;"Fast parallel processing on Workstation cluster", Advanced Institute of Science and Technology, Japan 1998.

[6] http://www.aethia.com/en/cluster_pc_en.shtml

[7] http://www.lam-mpi.org/mpi/mpi_top10.php

[8] Apon, A.; Buyya, R.; Hai Jin; Mache, J. "Cluster computing in the classroom: topics, guidelines, andexperiences", Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on Volume , Issue , 2001 Page(s):476 – 483.

[9] www.infinibandta.org

[10] R. Buyya (ed.), High Performance Cluster Computing: Systems and Architectures, Prentice Hall, 1999. (chapters 1, 9, 10) and [G. Pfister, In Search of Clusters, Prentice Hall, 1998].

[11] Chao-Tung Yang and Yao-Chung Chang; " A Linux PC Cluster with Diskless Slave Nodes for Parallel Computing",The Ninth Workshop on Compiler Techniques for High-Performance Computing,Institute of Information Science,Academia Sinica,2003

[12] http://maven.smith.edu/~thiebaut/transputer/toc.html RohitKelapure,Vikrant Colaso,"Performance Issues In Distributed Frameworks For Large-Scale Distributed Processing", VikrantRohit Report 2004.