KMITL Sci. J. Vol. 5 No. 1 Feb. 2005

PARALLEL MATRIX MULTIPLICATION ON A CLUSTER OF PCs

Phairoj Samutrak*, Jureeporn Boonniyom, and Jeeraporn Srisawat

Department of Mathematics and Computer Science
Faculty of Science
King Mongkut's Institute of Technology Ladkrabang (KMITL)
Ladkrabang, Bangkok 10520,
THAILAND

ABSTRACT

This paper presents the implementation of the coarse-grained parallel matrix multiplication (C = A x B) with two ways of data partitioning on a cluster of PCs. In the past, most existing studies proposed the medium-grained parallel matrix multiplications on the hypercube-connected or mesh-connected parallel computers. We propose to study and implement the practical parallel matrix multiplication based on the MPMD model on the cluster of PCs using MPI (Message Passing Interface) standard. In particular, two data partitioning schemes for decomposing matrix A and matrix B with balancing workload are presented: 1) the row-block partitioning and 2) the checkerboard-block partitioning. Moreover, we also introduce a modified parallel matrix multiplication to cover an approach of the parallel all-pair shortest paths. Finally, the system performance of sequential and parallel processing of the matrix multiplication have been compared and evaluated in terms of response time, speedup, and efficiency. Based on our experimental results, the system performance of the matrix multiplication was improved up to 50% when the number of processors (p) were increased by one

KEYWORDS: Parallel matrix multiplication, data-block partitioning, an approach of parallel all-pair shortest paths, a cluster of PCs, MPMD (Multiple Program Multiple Data), MPI (Message Passing Interface)

*Corresponding author. Tel: 013826529,E-mail:s6063611@kmitl.ac.th,p_samutrak@hotmail.com

1. INTRODUCTION

The matrix multiplication is one of the most common operations in science and engineering applications. Some matrix multiplication applications operate on very large matrix sizes (nxn) and hence the sequential processing cannot finish in reasonable time. In the past, a number of efficient parallel matrix multiplication methods were proposed as medium-grained computing for specific-interconnection parallel systems to process in fast time. Those existing methods are parallel matrix multiplications on the hypercube-connected parallel computers [1], [7], parallel matrix multiplications on the mesh-connected parallel computers [4], [5], and parallel matrix multiplication on the systolic-array parallel computers [8]. Another approach, a scalable parallel matrix multiplication based on theoretical PRAM model [6], was also introduced to be applied on distributed memory parallel computers (DMPC). Recently, matrix multiplication on hetorogeneous platforms [3] and the implementation of heterogeneous computing model of matrix multiplication [9] on a cluster of workstations were proposed.

This paper presents a practical coarse-grained parallel matrix multiplication (C = A x B) on a cluster of PCs using very high-speed network and MPI (Massage Passing Interface) standard [2]. First, we introduce and also implement our coarse-grained parallel matrix multiplication, based on two ways of data (matrix A and matrix B) decomposition: 1) row-block partitioning and 2) checkerboard-block partitioning. Second, we present a simple and practical approach of the parallel all-pair shortest paths, which is based on the modified parallel matrix multiplication. Then, system performance of sequential and parallel processing of the matrix multiplication have been compared and evaluated in terms of response time, speedup, and efficiency.

The remainder of this paper is organized as follows. Section 2 presents our two practical coarse-grained parallel matrix multiplication methods, namely: 1) row-block-partitioning matrix multiplication (rbmm) and 2) checkerboard-block-partitioning matrix multiplication (cbb). Section 3 illustrates an application of our parallel matrix multiplication, which is an approach of parallel all-pair shortest paths. Section 4 shows the experimental results of the system performance evaluation of sequential matrix multiplication and two proposed parallel matrix multiplication methods. Section 5 gives the conclusion of this study and discusses our future study

2. COARSE-GRAINED PARALLEL MATRIX MULTIPLICATION

In general, matrix multiplication (C_{nxn}) of matrix A_{nxn} and matrix B_{nxn} is computed in the following digram,

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \ldots & a_{1n} \\ a_{21} & a_{22} & a_{23} \ldots & a_{2n} \\ a_{31} & a_{32} & a_{33} \ldots & a_{3n} \\ \ldots & & & \\ a_{n1} & a_{n2} & a_{n3} \ldots & a_{nn} \end{pmatrix} \times \begin{pmatrix} b_{11} & b_{12} & b_{13} \ldots & b_{1n} \\ b_{21} & b_{22} & b_{23} \ldots & b_{2n} \\ b_{31} & b_{32} & b_{33} \ldots & b_{3n} \\ \ldots & & & \\ b_{n1} & b_{n2} & b_{n3} \ldots & b_{nn} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} & c_{13} \ldots & c_{1n} \\ c_{21} & c_{22} & c_{23} \ldots & c_{2n} \\ c_{31} & c_{32} & c_{33} \ldots & c_{3n} \\ \ldots & & & \\ c_{n1} & c_{n2} & c_{n3} \ldots & c_{nn} \end{pmatrix}$$

where for all i, j = 1, 2,..., n, $c_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j}$. Time complexity of the sequential matrix multiplication is $O(n^3)$ whereas time complexity of the parallel matrix multiplication using p processors (p = n) is $O(n^2)$.

Next we present our two coarse-grained parallel matrix multiplication methods, which are implemented by using C programming language with MPI standard. The MPI program consists of one master and p workers (or computing processors).

2.1) Parallel Matrix Multiplication based on Row-block Partitioning (rbmm)

For the row-block partitioning matrix multiplication (rbmm) method, each processor (p_i) in the worker group receives $\lceil n/p \rceil$ rows of matrix A and the whole matrix B from the master and computes $\lceil n/p \rceil$ rows of matrix C (or row $i\lceil n/p \rceil$ to $(i+1)\lceil n/p \rceil - 1$) and return its results to the master. The MPI code (Algorithm 1) and an example of balancing workload for each p_i (i.e., $n = 4$ and $p = 4$) are illustrated as follows:

P0	(0,0)	(0,1)	(0,2)	(0,3)
P1	(1,0)	(1,1)	(1,2)	(1,3)
P2	(2,0)	(2,1)	(2,2)	(2,3)
P3	(3,0)	(3,1)	(3,2)	(3,3)

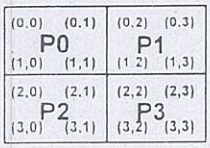
```
Master (Supervisor Node)
Start MPI
Create Data
Send Data to Workers
MPI_Send(&offset,1,MPI_INT,dest,mtype,MPI_COMM_WORLD);
MPI_Send(&rows,1,MPI_INT,dest,mtype,MPI_COMM_WORLD);
MPI_Send(&a12, NRA/2*NCA, rows*NCA,MPI_DOUBLE,dest, mtype, MPI_COMM_WORLD);
MPI_Send(&b13, NCA*NCB/2, MPI_DOUBLE, dest, mtype, PI_COMM_WORLD);

Wait for Result
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, MPI_DOUBLE, source, mtype, MPI_COMM_WORLD, &status);

Worker (Computing Node)
Receive Data 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 (&a12, NRA/2*NCA, MPI_DOUBLE, MASTER, mtype, MPI_COMM_WORLD, &status);
MPI_Recv((&b13, NCA*NCB/2, MPI_DOUBLE, MASTER, mtype, MPI_COMM_WORLD, &status);

Perform Matrix Multiplication
    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_WORKER;

Send Data to Master
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);
```

Algorithm 1: MPI code for parallel matrix multiplication based on row-block partitioning.

The rbmm method gives a simple data partitioning and workload computing. Here, workers do not have to communicate since the whole matrix B is broadcasted from the master to all workers. The parallel MPI-based matrix multiplication also presents this way.

2.2) Parallel Matrix Multiplication based on Checkerboard-Block Partitioning (cbb)

The cbb method (checkerboard-block-partitioning matrix multiplication) is introduced to reduce the redundant copy of matrix B in all workers, as required in the rbmm method. In this case, each processor (p,) in the worker group receives $n/\sqrt{p}$ rows of matrix A and $n/\sqrt{p}$ columns of matrix B from the master and computes sub-matrix ($n/\sqrt{p}$ x $n/\sqrt{p}$) of matrix C in checkerboard fashion and return its results to the master. However, in order to perform no communication among workers, some partial rows of matrix A and some partial columns of matrix B are replicated but less than the way that the rbmm method does. The MPI code and an example of workload for each pi (i.e., n = 4 and p = 4) are illustrated as follows:




```
Master (Supervisor Node)
Start MPI
Create Data
Partition Data
for (i=0, i<NRA/2, i++)
   for (j=0, j<NCA, j++) {
      a12[i][j]= a[i][j];
      a34[i][j]= a[NRA/2+i][j];   }
for (i=0, i<NCA, i++)
   for (j=0, j<NCB/2, j++)  {
      b13[i][j]= b[i][j];
      b24[i][j]= b[i][NCA/2+j];   }

Send Data to Workers
MPI_Send(&offset, 1, MPI_INT, 1, mtype, MPI_COMM_WORLD);
MPI_Send(&rows, 1, MPI_INT, 1, mtype, MPI_COMM_WORLD);
MPI_Send(&a12,NRA/2*NCA,MPI_DOUBLE,1,mtype,MPI_COMM_WORLD);
MPI_Send(&b13,NCA*NCB/2,MPI_DOUBLE,1,mtype,MPI_COMM_WORLD);

Wait for Result
for (i=1, i<=numworkers, 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[0][0], NRA*NCB, MPI_DOUBLE, source, mtype, MPI_COMM_WORLD, &status);
}

Worker (Computing Node)
Receive Data from 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(&a12, NRA/2*NCA, MPI_DOUBLE, MASTER, mtype, MPI_COMM_WORLD, &status);
MPI_Recv(&b13, NCA*NCB/2, MPI_DOUBLE, MASTER, mtype, MPI_COMM_WORLD, &status);

Perform Matrix Multiplication
   for (k=0; k<NCB/2; k++)
      for (i=0; i<NRA/2; i++) {
         c[i][k] = 0.0;
         for (j=0; j<NCA; j++)
            c[i][k] = c[i][k] + a12[i][j] * b13[j][k];   }
   mtype = FROM_WORKER;

Send Data to Master
MPI_Send(&offset,1,MPI_INT,MASTER,mtype,MPI_COMM_WORLD);
MPI_Send(&rows,1,MPI_INT,MASTER,mtype,MPI_COM_WORLD);
MPI_Send(&c, NRA*NCB, MPI_DOUBLE, MASTER, mtype, MPI_COMM_WORLD);
```

Algorithm 2: MPI code for parallel matrix multiplication based on checkerboard-block partitioning.

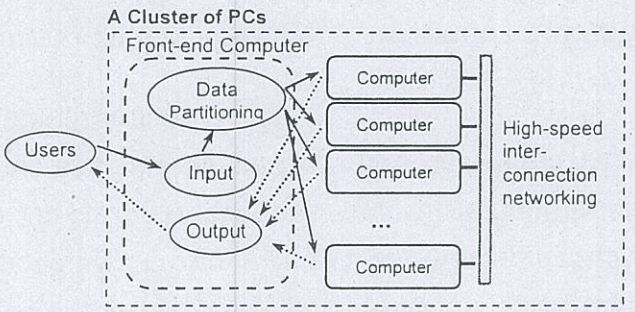


Fig.1 A cluster of PCs environment.

Fig.2 and Fig.3 showed the response time of the rbmm method, where p = 1, 2, 3, 4 for nxn = 300x300, 800x800, and 1000x1000, respectively. When the number of processors (p) were increased up to 4, the respond time (Fig.2) were improved up to 50%, 63%, and 69% when p = 2, 3, 4, respectively. However, when the matrix size (nxn) was less than 300x300, the response time of the rbmm method did not improved much when the number of processors were increased.

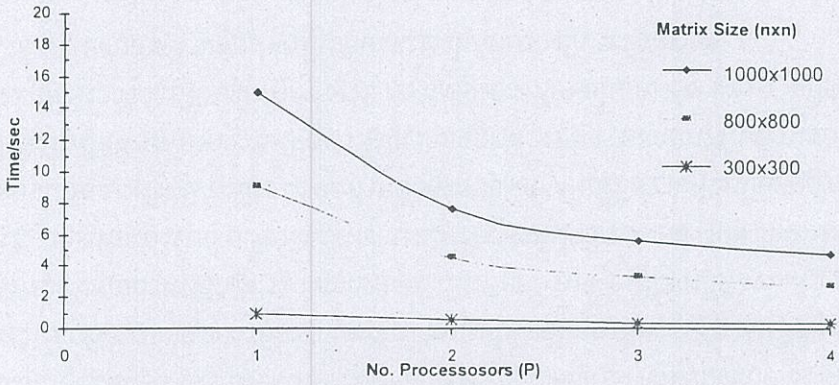


Fig. 2. Response time of the rbmm method (p = 1, 2, 3, 4).

When we investigate the effect of increasing the matrix sizes (nxn) on p = 4 processors (Fig. 3), where n = 500, 600, 800, 1000, 1200, and 1400, for all of these matrix sizes, both (rbmm and cbb) methods improved the response time over the sequential matrix multiplication 60 - 73%. Between two parallel methods, the rbmm and cbb methods performed the comparable response time.

Fig.4 illustrated the speedup of the rbmm method, where p = 2, 3, 4 and n = 300, 800, and 1000. When setting p = 2, the speedup of all matrix sizes were closed to 2 (an ideal case). For $n \geq 800$, the speedup S = 2.7 and 3.2, when setting p = 3 and 4, respectively.

Compared between two (rbmm and cbb) methods (Fig. 5), we investigate the effect of increasing the matrix sizes (nxn) on p = 4 processors, where n = 500, 600, ..., 1300, and 1400. For n = 500, 600, ..., 900, the cbb method yielded the better speedup up than those of the rbmm but when n > 900, the rbmm yielded the better speedup than those of the cbb. For n > 1200, the speedup of both methods was not improved.

However, one limitation of the cbb approach is the number of processors (p) must be arranged in square only (i.e., 2x2, 3x3, 4x4, ...) or p = 4, 9, 16, ...

3. MODIFIED PARALLEL MATRIX MULTIPLICATION FOR ALL-PAIR-SHORTEST PATHS

A simple and practical approach of computing parallel all-pair shortest paths (of n nodes) can be obtained from applying the parallel matrix multiplication (D^n), where the size of D (distance) or W (weight) is nxn. The matrix multiplication D^n is obtained from

Step 1: $D \times D \rightarrow D^2$,
Step 2: $D2 \times D2 \rightarrow D^4$,
Step 3: $D4 \times D4 \rightarrow D^8$,
...
Step $\log_2 n$: $D^{n/2} \times D^{n/2} \rightarrow D^n$.

Clearly, this process can be performed in only $\log_2 n$ steps. Note that for each pair of matrix multiplication, the necessary modification is to replacing the operators x (of $a_{ik} \times b_{kj}$) and + (of $\sum a_{ik} \times b_{kj}$; k = 1, 2, ... , n) in the original matrix multiplication with operators + (i.e., $a_{ik} + b_{kj}$) and min (i.e., $\min(a_{ik} + b_{kj}$; k = 1, 2, ... , n)), respectively.

```
Algorithm
ParAllPairShortestPaths(W, D)
for all i&j pardo
    dij(1)=w(i,j)
end for
for k = 2 to log n do
    MatrixMultiplication(D, D, D, +, Min)
 end for: end
```

Time complexity of the above sequential all-pair shortest paths is $O(n^3 \log n)$, whereas time complexity of the parallel matrix multiplication using p processors (p = n) is $O(n^2 \log n)$.

4. PERFORMANCE EVALUATION

In the system performance evaluation, we implement our parallel matrix multiplication methods (rbmm and cbb) on a cluster of PCs. In our cluster environment, the system consists of 10 CPUs residing in 5 computer units (2 CPUs/unit). One computer, called a front-end computer, works as a master and the rest of computers work as workers. All computers in this cluster system are homogeneous, each of which is 2.4 GHz Xeon (1 GB RAM) and connected via a high speed 1000 Mbps LAN (Fig.1). The master process at the front-end computer is responsible to communicate to the user and perform data partitioning and broadcast them to all workers and collect the final results from the workers. This computing model is known as MPMD (Multiple Programs Multiple Data).

A number of experiments were performed to investigate and compare these two parallel matrix multiplication methods (rbmm, cbb) and the sequential matrix multiplication on such a cluster system. During experiments, many parameters (i.e., nxn (no. of elements), p (no. of processors)) were varied to investigate response time (T), speedup ($S = T_1/T_p$), and efficiency ($E = S/N$), where T_1 (s) is the execution time of running sequential program on one processor and T_p (s) is the execution time of running parallel program on p processors.

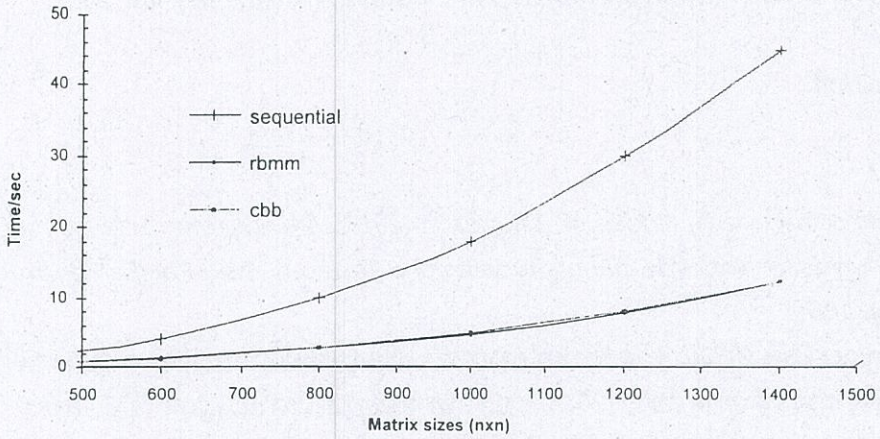


Fig. 3. Response time of the rbmm and cbb methods (p = 4).

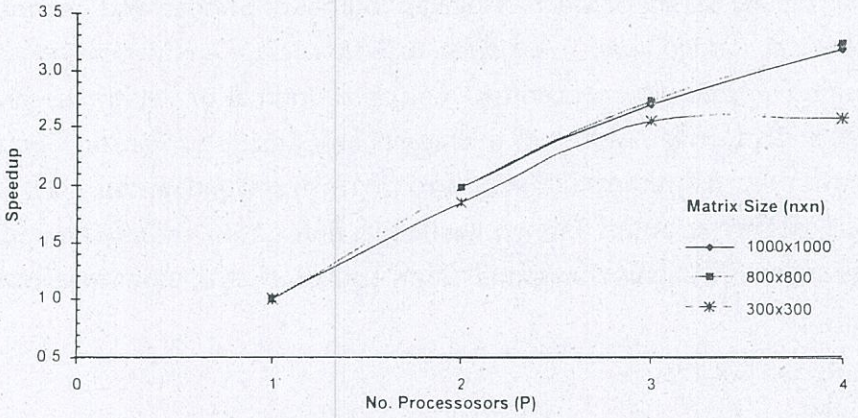


Fig. 4. Speedup of the rbmm method (p = 1, 2, 3, 4).

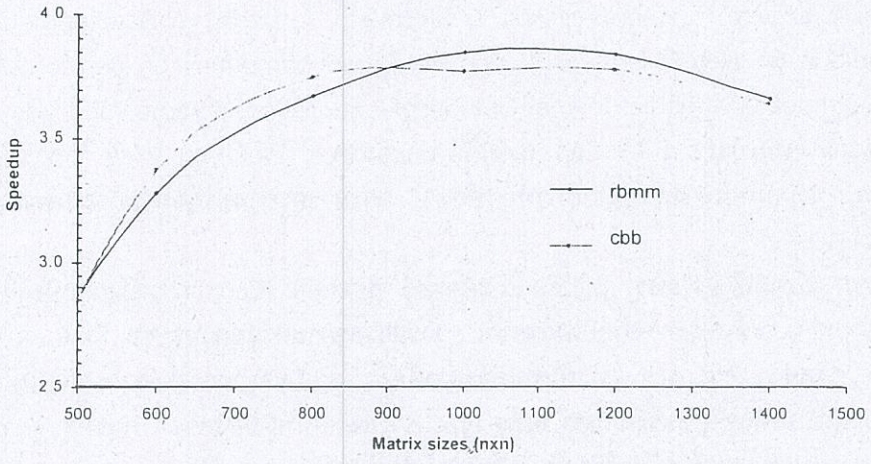


Fig. 5. Speedup of the rbmm and cbb methods (p = 4).

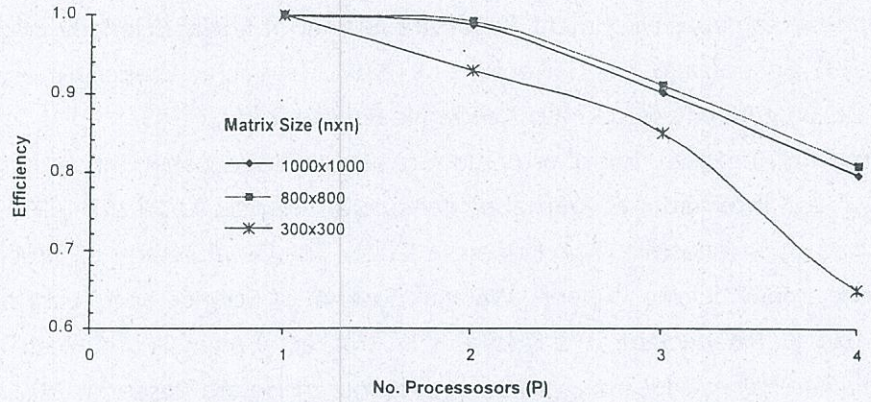


Fig. 6. Efficiency of the rbmm method (p = 1, 2, 3, 4).

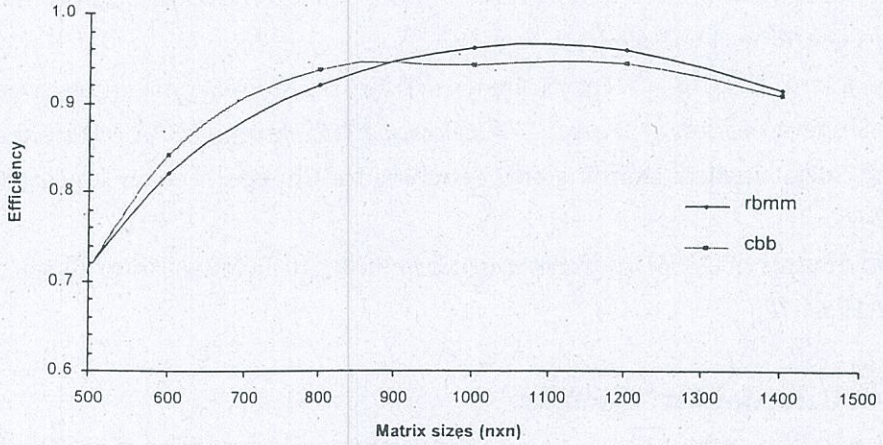


Fig. 7. Efficiency of the rbmm and cbb methods (p = 4).

Fig.6 illustrated the efficiency of the rbmm method, where p = 2, 3, 4 and n = 300, 800, 1000. When setting p = 2, the efficiency of the last two matrix sizes were closed to 1. Fig. 7 showed the similar compared results between two (rbmm and cbb) methods (as illustrated in Fig. 5 for speedup), where p = 4, n = 500, 600, ..., 1400.

5. CONCLUSIONS

In this study, we introduce two parallel matrix multiplication (rbmm and cbb) on a coarse-grained cluster of PCs. In the experimental results, when p (no. of processors) was set to 2, both proposed methods improved response time over the sequential method up to 50% and yielded speedup close to 2 and efficiency close to 1. In our future study, we try to perform efficient-space (by reducing all redundant data on matrix A and matrix B) for both parallel matrix multiplication.

6. ACKNOWLEDGEMENTS

Our work was implemented on the KMITL's cluster of PCs. This cluster system was supported by the computer research and service center at King Mongkut's Institute of Technology Ladkrabang (KMITL), directed by Assoc. Prof. Dr. Manas Sungwarasil and his research assistance. Nopparat Pantsaena

REFERENCES

[1] Alonso Sanches, C.A. and Song, S.W.**1994** SIMD Algorithms for Matrix Multiplication on the Hypercube. *The 8th Int'l Proceedings on Parallel Processing Symposium*, 490-496.

[2] Browne, S., Dongarra, J. and London, K. **1997** Review of Performance Analysis Tools for MPI Parallel rograms, http://www.cs.utk.edu/~browne/ perftools -review/.

[3] Beaumont, O., Boudet, V., Rastello, F. and Robert, Y. **2001** Matrix Multiplication on Heterogeneous Platforms, *IEEE Trans. on Parallel and Distributed Systems*, v.12 (10),1033-1051.

[4] Choi, J. **1997** A Fast Scalable Universal Matrix Multiplication Algorithm on Distributed-Memory Concurrent Computers, *IEEE Trans. on Parallel and Distributed Systems3,310-314.*

[5] Choi, J. **1997** A New Parallel Matrix Multiplication Algorithm on Distributed-Memory Concurrent Computers, *HPC Asia '97 High Performance Computing on the Information Superhighway*, 224-229.

[6] Li, K. **2000** Scalable Parallel Matrix Multiplication on Distributed Memory Parallel Computers, *Parallel and Distributed Processing Symposium IPDPS 2000. Proceedings. 14th Internationa*,307-314.

[7] Sengupta, A. and Raghavendra, C.S. **1998** All-To-All Broadcast and Matrix Multiplication in Faulty SIMD Hypercubes, *IEEE Trans. on Parallel and Distributed Systems*, v.9(6),550-560.

[8] Tasic, J.F., Zajc, M. and Kosir, A. **1996** Comparison of Some Parallel Matrix Multiplication Algorithms, *Electrotechnical Conference MELECON '96., 8th Mediterranean*, v.1,155-158.

[9] Typou, T., Stefanidis, V., Michailidis P. and Margaritis, K. **2004** Implementing Matrix Multiplication on an MPI Cluster of Workstations. *The 1st Int'l Conference from Scientific Computing to Computational Engineering (IC-SCCE)*, Athens.