PERFORMANCE MODELS FOR MATRIX COMPUTATIONS ON NETWORKS OF HETEROGENEOUS WORKSTATIONS

Panagiotis D. Michailidis, Theodoros Typou, Vasilis Stefanidis and Konstantinos G. Margaritis

Parallel and Distributed Processing Laboratory
Department of Applied Informatics
University of Macedonia
156 Egnatia str., P.O. Box 1591
GR-54006 Thessaloniki, Greece

Abstract
Matrix computations is one of the most important computational tasks in scientific computing. This paper discusses parallel implementations of two fundamental matrix computation kernels such as matrix - vector multiplication and matrix multiplication on a cluster of heterogeneous workstations. These parallel implementations are based on the master - worker paradigm using dynamic block distribution scheme. Furthermore, the parallel implementations are analyzed experimentally using the Message Passing Interface (MPI) library on homogeneous and heterogeneous clusters of workstations. Also, we propose a general analytical prediction model that can be used to predict the performance of two matrix computations for both types of clusters. The theoretical performance model has been checked against experimental results and it is shown that the model is able to predict the parallel performance accurately.

Keywords - Matrix - vector multiplication, matrix multiplication, cluster of workstations, Message Passing Interface (MPI), performance prediction model

1. INTRODUCTION
Matrix computations is the most fundamental group of operations in numerical computing. The efficient implementation of matrix computations on parallel computers is an issue of prime importance when providing such systems with scientific software libraries (Anderson et al., 1992; Choi et al., 1992a; Choi et al., 1992b; Dongarra et al., 1979; Dongarra & Whaley, 1997). Consequently, considerable effort has been devoted in the past to the development of efficient parallel matrix algorithms, and this will remain a task in the future as well. This is mainly due to evolutionary architectures of parallel and distributed computers such as 1D- and 2D-systolic arrays, 2D-meshes, 2-D tori, and hypercubes, as well as the shared memory model and distributed memory model. Many parallel algorithms have been designed, implemented, and tested on different parallel computers or clusters of homogeneous workstations.
for matrix computations (Fox et al., 1988; Grama et al., 2003; Wilkinson & Allen, 2005). It is possible that most of the reported parallel algorithms used in practice are based on parallel multicomputers where the processors are interconnected in a two dimensional mesh. These algorithms may be roughly classified as "broadcast and shift algorithms" initially presented in (Fox et al., 1988) and "align and shift algorithms", initially presented in (Cannon, 1969). Both kinds of algorithms are described in terms of two-dimensional processor grids where each processor holds a large consecutive block of data.

Several distributed implementations BLACS, ScaLAPACK, PUMMA, SUMMA, and LAPACK (Agarwal et al, 1994; Anderson et al., 1992; Choi et al., 1992a; Choi et al., 1994; Choi, 1997; Dongarra et al., 1979; Dongarra & Whaley, 1997; Geijn & Watts, 1997; Li, 2005) have been successfully adapted the broadcast based algorithms to the point-to-point interconnection of two dimensional torus. All of them aim to obtain the best performance for the whole algorithm by implementing broadcast over the point-to-point communication links of distributed memory parallel computers.

Recently, several parallel matrix computations have been proposed and implemented on cluster of heterogeneous workstations (Beaumont et al., 2000; Beaumont et al., 2001a; Beaumont et al., 2002; Kalinov, 2004; Ohtaki et al., 2004; Tinetti et al., 2001). They use different distribution schemes and redesigned several parallel matrix algorithms designed for traditional parallel computers to heterogeneous clusters. Finally, the general efficient algorithms for the master-worker paradigm on heterogeneous clusters have been widely developed in (Beaumont et al., 2001b).

In this paper we study implementations for two matrix computation kernels such as matrix - vector multiplication and matrix multiplication on a cluster of workstations and develop a performance prediction model for these implementations. This work differs from previous ones in two aspects: First, we develop four parallel implementations for two matrix computations. These parallel implementations are based on row block decomposition scheme using dynamic master - worker model. Further, these implementations were executed using the Message Passing Interface (MPI) (Pacheco, 1997; Snir et al., 1996) library on two kinds of high performance cluster of workstations: homogeneous and heterogeneous. Second, we develop a simple heterogeneous performance model for four implementations that is general enough to cover performance evaluation of both homogeneous and heterogeneous computations in a dedicated cluster of workstations. So, we consider the homogeneous computing as a special case of heterogeneous computing.

The rest of this paper is organized as follows: Section 2 briefly presents heterogeneous computing model and the metrics. Section 3 presents a performance prediction model for estimating the performance of the four matrix implementations on a clus-
ter of workstations. Section 4 discusses the validation of the performance prediction model with the experimental results. Finally, Section 5 contains our conclusions.

2. HETEROGENEOUS COMPUTING MODEL

A heterogeneous network (HN) can be abstracted as a connected graph HN(M,C), where

- \( M = \{M_1, M_2, \ldots, M_p\} \) is set of heterogeneous workstations (\( p \) is the number of workstations). The computation capacity of each workstation is determined by the power of its CPU, I/O and memory access speed.

- \( C \) is standard interconnection network for workstations, such as Fast Ethernet or an ATM network, where the communication links between any pair of the workstations have the same bandwidth.

Based on the above definition, if a cluster consists of a set of identical workstations, the cluster is homogeneous.

2.1 Metrics

Metrics help to compare and characterize parallel computer systems. Metrics cited in this section are defined and published in previous paper (Yan et al., 1996). They can be roughly divided into characterization metrics and performance metrics.

2.1.1 Characterization metrics

To compute the power weight among workstations an intuitive metric is defined as follows:

\[
W_i(A) = \frac{\min_{j=1}^{p}\{T(A, M_j)\}}{T(A, M_i)}
\]  

(1)

where \( A \) is an application and \( T(A, M_i) \) is the execution time for computing \( A \) on workstation \( M_i \). Formula 1 indicates that the power weight of a workstation refers to its computing speed relative to the fastest workstation in the network. The value of the power weight is less than or equal to 1. However, if the cluster of workstations is homogeneous then the values of the power weights are equal to 1.

To calculate the execution time of a computational segment, the speed, denoted by \( S_f \) of the fastest workstation executing basic operations of an application is measured by the following equation:

\[
S_f = \frac{\Theta(c)}{t_c}
\]  

(2)

where \( c \) is a computational segment, \( \Theta(c) \) is a complexity function which gives the number of basic operations in a computational segment and \( t_c \) is the execution time of \( c \) on the fastest workstation in the network.
Using the speed of the fastest workstation, $S_f$, we can calculate the speeds of the other workstations in the system, denoted by $S_i$ ($i = 1, \ldots, p$), using the computing power weight as follows:

$$S_i = S_f \times W_i, \ i = 1, \ldots, p, \ \text{and} \ i \neq f \ (3)$$

where $W_i$ is the computing power weight of $M_i$. So, by equation 3, the execution time of a segment $c$ across the heterogeneous network $HN$, denoted by $T_{cpu}(c, HN)$, can be represented as

$$T_{cpu}(c, HN) = \frac{\Theta(c)}{\sum_{i=1}^{p} S_i} \ (4)$$

where $\sum_{i=1}^{p} S_i$ is the computational capacity used which is obtained by summing the individual speeds of the workstations. Here, $T_{cpu}$ is considered the required CPU time for the segment. Furthermore, substituting $S_i = 1$ in above equation for dedicated cluster of homogeneous workstations, the execution time of a segment $c$ returns to the conventional form:

$$T_{cpu}(c, HN) = \frac{\Theta(c)}{p} \ (5)$$

### 2.1.2 Performance metrics

Speedup is used to quantify the performance gain from a parallel computation of an application $A$ over its computation on a single machine on a heterogeneous network system. The speedup of a heterogeneous computation is given by:

$$SP(A) = \min_{j=1}^{p} \{T(A, M_j)\} / T(A, HN) \ (6)$$

where $T(A, HN)$ is the total parallel execution time for application $A$ on $HN$, and $T(A, M_j)$ is the execution time for $A$ on workstation $M_j$, $j = 1, \ldots, p$.

Efficiency or utilization is a measure of the time percentage for which a machine is usefully employed in parallel computing. Therefore, the utilization of parallel computing of application $A$ on a dedicated heterogeneous network is defined as follows:

$$E = \frac{SP(A)}{\sum_{j=1}^{p} W_j} \ (7)$$

The previous formula indicates that if the speedup is larger than $\sum_{j=1}^{p} W_j$, the system computing power, the computation presents a superlinear speedup in a dedicated heterogeneous network. Further, substituting $W_j = 1$ in above equation for dedicated cluster of homogeneous workstations, the utilization returns to the traditional form:

$$E = \frac{SP(A)}{p} \ (8)$$
3. PERFORMANCE MODELS

In this section, we develop an analytical performance model to describe the computational behaviour of four parallel matrix multiplication implementations of both kinds cluster. We must note that the matrix multiplication is a generalized problem of the matrix - vector multiplication. First of all, we consider the matrix multiplication product \( C = A \times B \) where the three matrices \( A, B \) and \( C \) are dense of size \( n \times n \). In case of matrix - vector multiplication the \( A \) is matrix of size \( 1 \times n \) instead of \( n \times n \), whereas the \( C \) is matrix of size \( 1 \times n \). The number of workstations in the cluster is denoted by \( p \) and we assume that \( p \) is power of 2. The performance modeling of the four implementations is presented in next subsections.

3.1 Dynamic allocation of columns of the matrix \( B \) (MM1, MV1)

For the analysis, we assume that the entire matrix \( B \) stored in the local disk of the master workstation. The basic idea of this implementation is as follows: The master workstation broadcasts the matrix \( A \) by rows to all workers. It partitions the matrix \( B \) into blocks of columns and each block is distributed dynamically to a worker. Each worker executes a sequential matrix - vector multiplication algorithm between the matrix \( A \) and the corresponding block of \( b \) columns. Finally, each worker sends back a block of size \( b \) columns of the matrix \( C \). The execution time of the dynamic implementation that is called MM1 (for matrix multiplication) or MV1 (for matrix - vector multiplication), can be broken up into five terms:

- \( T_a \): It includes the communication time for broadcasting of the matrix \( A \) by rows to all workers involved in processing of the matrix multiplication. We used the function \( MPI\_Bcast \) to broadcast information to all workers. The size of each row of the matrix \( A \) is \( n \times \text{sizeof}(\text{int}) \) bytes. Therefore, the total time \( T_a \) for \( n \) rows is given by:

\[
T_a = \frac{n^2 \times \text{sizeof}(\text{int})}{S_{\text{comm}}}
\]  

where \( S_{\text{comm}} \) is the communication speed. If the \( A \) is matrix of size \( 1 \times n \), then the time \( T_a \) is as follows: \( T_a = \frac{n \times \text{sizeof}(\text{int})}{S_{\text{comm}}} \).

- \( T_b \): It is the total I/O time to read the columns of the matrix \( B \) into several blocks of size \( b \times n \times \text{sizeof}(\text{int}) \) bytes from the local disk of the master workstation. The \( b \) is the number of columns. Therefore, the master reads \( n^2 \times \text{sizeof}(\text{int}) \) bytes totally of the matrix. Then, the time \( T_b \) is given by:

\[
T_b = \frac{n^2 \times \text{sizeof}(\text{int})}{(S_{I/O})_{\text{master}}}
\]  

where \((S_{I/O})_{\text{master}}\) is the I/O capacity of the master workstation.
• $T_c$: It is the total communication time to send all blocks of the matrix $B$ to all workers. The size of each block is $b \times n \times \text{sizeof}(\text{int})$ bytes. Therefore, the master sends $n^2 \times \text{sizeof}(\text{int})$ bytes totally. Then, the time $T_c$ is given by:

$$T_c = \frac{n^2 \times \text{sizeof}(\text{int})}{S_{\text{comm}}}$$  \hfill (11)

where $S_{\text{comm}}$ is the communication speed.

• $T_d$: It is the average computation time across the cluster. Each worker performs a matrix - vector multiplication between the matrix $A$ and the block of the matrix $B$ with size $b \times n \times \text{sizeof}(\text{int})$ bytes. It requires $b \times n^2$ steps. Then, the time $T_d$ is given by:

$$T_d = \frac{(n^2 - p)(b \times n^2 \times \text{sizeof}(\text{int}))}{\sum_{j=1}^{p}(S_{\text{comp}})_j} + \max_{j=1}^{p}\left\{\frac{b \times n^2 \times \text{sizeof}(\text{int})}{(S_{\text{comp}})_j}\right\}$$  \hfill (12)

where $\sum_{j=1}^{p}(S_{\text{comp}})_j$ is the computation capacity of the cluster (homogeneous or heterogeneous) when $p$ workstations are used. We include the second $\max$ term in the equation 12 which defines the worse load imbalance at the end of the execution when there are not enough blocks of the matrix left to keep all the workstations busy. If the $A$ is matrix of size $1 \times n$, then the time $T_d$ is as follows: $T_d = \frac{(\frac{n^2}{b} - p)(b \times n \times \text{sizeof}(\text{int}))}{\sum_{j=1}^{p}(S_{\text{comp}})_j} + \max_{j=1}^{p}\left\{\frac{b \times n \times \text{sizeof}(\text{int})}{(S_{\text{comp}})_j}\right\}$.

• $T_e$: It includes the communication time to receive $\frac{n^2}{b}$ results from all workers. Each worker sends back $b \times n \times \text{sizeof}(\text{int})$ bytes. Therefore, the master will receive $n^2 \times \text{sizeof}(\text{int})$ bytes totally. Therefore, the time $T_e$ is given by:

$$T_e = \frac{n^2 \times \text{sizeof}(\text{int})}{S_{\text{comm}}}$$  \hfill (13)

where $S_{\text{comm}}$ is the communication speed. If the $A$ is matrix of size $1 \times n$, then the time $T_e$ is as follows: $T_e = \frac{n \times \text{sizeof}(\text{int})}{S_{\text{comm}}}$.

Further, we note that this dynamic implementation in practice there is parallel communication and computation and this reason we take the maximum value between the communication time i.e. $T_c + T_e$ and the computation time i.e. $T_d$. Therefore, the total execution time of our dynamic implementation, $T_p$, using $p$ workstations, is given by:

$$T_p = T_a + T_b + \max\{T_c + T_e, T_d\}$$  \hfill (14)

3.2 Dynamic allocation of matrix pointers (MM2, MV2)

For the analysis, we assume that the matrix $B$ stored in the local disk of the master workstation. Further, in this implementation we assume that the master workstation
has a pointer that shows the current column in the matrix $B$. The basic idea of the second implementation is as follows: The master workstation broadcasts the matrix $A$ by rows to all workers. It also sends pointers for the matrix $B$ dynamically to workers. Each worker reads a block of size $b$ columns of the matrix $B$ from the local disk of the master starting from the pointer that receives. Also, each worker executes a sequential matrix-vector multiplication algorithm between the matrix $A$ and the corresponding block of $b$ columns. Finally, each worker sends back a block of size $b$ columns of the matrix $C$. The execution time of the dynamic implementation with the matrix pointers that is called MM2 or MV2, can be broken up into five terms:

- $T_a$: It includes the communication time for broadcasting of the matrix $A$ by rows to all workers involved. The amount of this time is similar to the $T_a$ of the previous dynamic implementation.

- $T_b$: It is same with the time $T_c$ of the previous implementation but this term includes the total communication time to send $\frac{n}{b}$ matrix pointers instead of blocks of columns of the matrix $B$ to all workers. Therefore, the time $T_b$ is given by:

$$T_b = \frac{n}{bS_{comm}}$$

(15)

where $S_{comm}$ is the communication speed.

- $T_c$: It is the total I/O time to read the matrix $B$ into several blocks of size $b \times n \times sizeof(int)$ bytes from the local disk of the master workstation. We note that each worker reads a block of the matrix $B$ from the local disk of the master starting from the pointer that receives. The amount of this time is similar to the $T_b$ of the previous implementation.

- $T_d$: It includes the average computation time across the cluster. The amount of this time is similar to the time $T_d$ of the previous implementation.

- $T_e$: It includes the communication time to receive the results of the matrix-vector multiplication from all workers. The amount of this time is same with the time $T_e$ of the previous implementation.

We take the maximum value between the communication time i.e. $T_b + T_e$ and the computation time i.e. $T_c + T_d$, since in this implementation there is parallel communication and computation. Therefore, the total execution time of our dynamic implementation, $T_p$, using $p$ workstations, is given by:

$$T_p = T_a + max\{T_b + T_e, T_c + T_d\}$$

(16)
3.3 Dynamic allocation of matrix pointers (MM3, MV3)

For the analysis, we assume that the entire matrix $B$ stored in the local disk of worker workstations. The basic idea of the third implementation is as follows: The master workstation broadcasts the matrix $A$ by rows to all workers. It sends matrix pointers for the matrix $B$ dynamically to workers. Each worker reads a block of size $b$ columns of the matrix $B$ from the local disk of the worker starting from the pointer that receives. Also, each worker executes a sequential matrix - vector multiplication algorithm between the matrix $A$ and the corresponding block of $b$ columns. Finally, each worker sends back a block of size $b$ columns of the matrix $C$. The execution time of the dynamic implementation with the matrix pointers that is called MM3 or MV3, can be broken up into five terms:

- $T_a$: It includes the communication time for broadcasting of the matrix $A$ by rows to all workers involved. The amount of this time is similar to the $T_a$ of the previous dynamic implementation.

- $T_b$: It includes the communication time to send all matrix pointers to all workers. The amount of this time is same with the time $T_b$ of the previous implementation.

- $T_c$: It is the average I/O time to read the matrix $B$ into several blocks of size $b \times n \times sizeof(int)$ bytes from the local disks of the worker workstations. We note that each worker reads from the local disk $b \times n \times sizeof(int)$ bytes of the matrix $B$ starting from the pointer that receives. Then, the time $T_c$ is given by:

$$T_c = \frac{(n^2 - p)(b \times n \times sizeof(int))}{\sum_{j=1}^{p}(S_{I/O})_j} + \max_{j=1}^{p}\left\{\frac{b \times n \times sizeof(int)}{(S_{I/O})_j}\right\}$$

(17)

where $\sum_{j=1}^{p}(S_{I/O})_j$ is the I/O capacity of the cluster (homogeneous or heterogeneous) when $p$ workstations are used. We include the second $\max$ term in the equation 17 which defines the worse load imbalance at the end of the execution when there are not enough blocks of the matrix left to keep all the workstations busy.

- $T_d$: It includes the average computation time across the cluster. The amount of this time is similar to the time $T_d$ of the previous implementation.

- $T_e$: It includes the communication time to receive the results of the matrix - vector multiplication from all workers. The amount of this time is same with the time $T_e$ of the previous implementation.
We take the maximum value between the communication time i.e. \( T_b + T_e \) and the computation time i.e. \( T_c + T_d \), since in this implementation there is parallel communication and computation. Therefore, the total execution time of our dynamic implementation, \( T_p \), using \( p \) workstations, is given by:

\[
T_p = T_a + \max\{T_b + T_e, T_c + T_d\}
\]

(18)

3.4 \textbf{Dynamic allocation of matrix pointers (MM4, MV4)}

For the analysis, we assume that the matrices \( A \) and \( B \) stored in the local disk of worker workstations. The basic idea of the fourth implementation is as follows: The master sends matrix pointers for the matrix \( B \) dynamically to workers. Each worker reads the entire matrix \( A \) from the local disk. Also, each worker reads a block of the matrix \( B \) from the local disk starting from the pointer that receives. Further, each worker executes a sequential matrix-vector multiplication algorithm between the matrix \( A \) and the corresponding block of \( b \) columns. Finally, each worker sends back a block of size \( b \) columns of the matrix \( C \). The execution time of the dynamic implementation with the matrix pointers that is called MM4 or MV4, can be broken up into five terms:

- **\( T_a \):** It includes the I/O time to read the matrix \( A \) from the local disk of the worker workstations. Therefore, the time \( T_a \) is given by:

\[
T_a = \frac{n^2 \times \text{sizeof}(\text{int})}{\sum_{j=1}^{p}(S_{I/O})_j}
\]

(19)

where \( \sum_{j=1}^{p}(S_{I/O})_j \) is the I/O capacity of the cluster (homogeneous or heterogeneous) when \( p \) workstations are used. If the \( A \) is matrix of size \( 1 \times n \), then the time \( T_a \) is as follows: \( T_a = \frac{nn\text{sizeof}(\text{int})}{\sum_{j=1}^{p}(S_{I/O})_j} \).

- **\( T_b \):** It includes the communication time to send all matrix pointers to all workers. The amount of this time is same with the time \( T_b \) of the previous implementation.

- **\( T_c \):** It is the average I/O time to read the matrix \( B \) into several blocks of size \( b \times n \times \text{sizeof}(\text{int}) \) bytes from the local disks of the worker workstations. The amount of this time is similar to the time \( T_c \) of the previous implementation.

- **\( T_d \):** It includes the average computation time across the cluster. The amount of this time is similar to the time \( T_d \) of the previous implementation.

- **\( T_e \):** It includes the communication time to receive the results of the matrix-vector multiplication from all workers. The amount of this time is same with the time \( T_e \) of the previous implementation.
We take the maximum value between the communication time i.e. \( T_b + T_e \) and the computation time i.e. \( T_a + T_c + T_d \), since in this implementation there is parallel communication and computation. Therefore, the total execution time of our dynamic implementation, \( T_p \), using \( p \) workstations, is given by:

\[
T_p = \max\{T_b + T_e, T_a + T_c + T_d\}
\]  

(20)

4. EXPERIMENTAL AND ANALYTICAL RESULTS

In this section, we compare the analytical results with the experimental results for four parallel implementations of two matrix operations like matrix - vector multiplication and matrix multiplication. These parallel algorithms are implemented in ANSI C programming language using the MPI library (Pacheco, 1997; Snir et al., 1996).

4.1 Experimental environment

The target platform for our experimental study are two kinds clusters of workstations connected with 100 Mb/s Fast Ethernet network. The homogeneous cluster consists of 32 Pentium workstations based on 200 MHz with 64 MB RAM. The heterogeneous cluster consists of 10 Pentium 233 MHz with 64 MB RAM, 12 Pentium 200 MHz with 64 MB RAM, 5 Pentium 166 MHz with 64 MB RAM and 5 Pentium 100 MHz with 64 MB RAM. The configuration of the heterogeneous cluster was as follows: 12 Pentium 200 MHz, 10 Pentium 233 MHz, 5 Pentium 166 MHz and 5 Pentium 100 MHz. The middleware of the clusters is ROCKS of NPACI (Papadopoulos et al., 2001) with RedHat 7.1. The MPI implementation used on the network is MPICH version 1.2. During all experiments, two clusters were dedicated. Finally, to get reliable performance results 5 - 10 executions occurred for each experiment and the reported values are the average ones.

4.2 Computing power weight and speed

The performance estimated results for four implementations of two matrix computations were obtained by the equations 14, 16, 18 and 20, respectively. In order to get these estimated results, we must to determine the values of the power weights and the values of the speeds \( S_{I/O} \) and \( S_{comp} \) of the fastest workstation. The computing power weight provides an average performance reference by including the heterogeneity of network systems. The power weight is a combination measurement of a program and a cluster system. We consider the effects of memory size on the execution of two matrix computations, which is a major architecture performance factor.

In our experiments, the execution timing results of matrix computations on the network system were measured using different matrix sizes. The performance is di-
Table 1: Speeds (in ints/sec) for I/O and computation of the fastest workstation for two matrix computations

<table>
<thead>
<tr>
<th>Application</th>
<th>$S_{I/O}$</th>
<th>$S_{comp}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>MV</td>
<td>1454905,38</td>
<td>25718490,9</td>
</tr>
<tr>
<td>MM</td>
<td>147,12526</td>
<td>9116288,837</td>
</tr>
</tbody>
</table>

Table 2: Speeds (in ints/sec) for communication for two matrix computations

<table>
<thead>
<tr>
<th>Application</th>
<th>$S_{comm}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>MV</td>
<td>52739,99</td>
</tr>
<tr>
<td>MM</td>
<td>47972,78</td>
</tr>
</tbody>
</table>

rectly related to the memory size in each workstation. The average computing power weights of the four types of workstations for four implementations (Pentium 233 MHz, Pentium 200 MHz, Pentium 166 MHz and Pentium 100 MHz) are 1, 0.583, 0.494 and 0.134 respectively. These weights based on formula 1 were measured when the matrix size does not exceed the memory bound of any machine in the system in order to keep the power weights constant.

Finally, the average speeds $S_{I/O}$ and $S_{comp}$ of the fastest workstation for all block sizes ($b = 1, 2, 4$ and $8$), executing the two matrix computations are critical parameters for predicting the CPU demand times of computational segments on other workstations. The speeds were measured for different matrix sizes and averaged by formula 2 as presented in Table 1 for four implementations. Finally, the communication speed was measured for different matrix sizes and block sizes as presented in Table 2.

4.3 Comparison between the experimental and theoretical results

Figures 1, 2, 3 and 4 present for some values $n$, $b$ and $p$ the speedups obtained by experiments and those by the equations 14, 16, 18 and 20 for the MV1, MV2, MV3 and MV4 implementations respectively on a cluster of homogeneous workstations. Similarly, Figures 5, 6, 7 and 8 present the speedups obtained by experiments and those by the equations 14, 16, 18 and 20 for the MV1, MV2, MV3 and MV4 implementations respectively on a cluster of heterogeneous workstations.

Figures 9, 10 and 11 present for some values $n$, $b$ and $p$ the speedups obtained by experiments and those by the equations 16, 18 and 20 for the MM2, MM3 and MM4 implementations respectively on a cluster of homogeneous workstations. It must be noted that the MM1 implementation is not executed on cluster of workstations because it gives poor results and the performance behavior is similar to the
Figure 1: The experimental (left) and theoretical (right) speedup as a function of the number of workstations on a homogeneous cluster for the MV1 implementation.
Figure 2: The experimental (left) and theoretical (right) speedup as a function of the number of workstations on a homogeneous cluster for the MV2 implementation.
Figure 3: The experimental (left) and theoretical (right) speedup as a function of the number of workstations on a homogeneous cluster for the MV3 implementation.
Figure 4: The experimental (left) and theoretical (right) speedup as a function of the number of workstations on a homogeneous cluster for the MV4 implementation.
Figure 5: The experimental (left) and theoretical (right) speedup as a function of the number of workstations on a heterogeneous cluster for the MV1 implementation
Figure 6: The experimental (left) and theoretical (right) speedup as a function of the number of workstations on a heterogeneous cluster for the MV2 implementation.
Figure 7: The experimental (left) and theoretical (right) speedup as a function of the number of workstations on a heterogeneous cluster for the MV3 implementation
Figure 8: The experimental (left) and theoretical (right) speedup as a function of the number of workstations on a heterogeneous cluster for the MV4 implementation.
Figure 9: The experimental (left) and theoretical (right) speedup as a function of the number of workstations on a homogeneous cluster for the MM2 implementation.
Figure 10: The experimental (left) and theoretical (right) speedup as a function of the number of workstations on a homogeneous cluster for the MM3 implementation.
Figure 11: The experimental (left) and theoretical (right) speedup as a function of the number of workstations on a homogeneous cluster for the MM4 implementation.
performance of the MV1 implementation according to (Typou et al, 2004b). Moreover, Figures 9 and 10 are not displayed the speedup curves for the block sizes 1 and 2 because they occur too low performance. Finally, the implementations of matrix multiplication were not executed on a cluster of heterogeneous workstations because the performance can be easily predicted according to the results of Figures 5, 6, 7 and 8.

We can see that the estimated results of the four implementations for two matrix computations confirm well the experimental measurements. However, there are small differences between the experimental and theoretical results in the MV2 implementation of both clusters. This occurs because our performance model of the MV2 implementation did not take into account the time that the worker reads a block from the disk of the master accurately.

Therefore, our performance model of the equations 14, 18 and 20 validate well the computational behaviour of the experimental results. Finally, we observe that the maximal differences between theoretical and experimental values are less than 10% and most of them are less than 5%. Consequently, the model for the MV1, MV3, MV4, MM2, MM3 and MM4 implementations is accuracy since the predictions need not be quantitatively exact, prediction errors of 10-20% are acceptable.

4.4 General remarks

As can be seen from Figures that as the number of workstations increases, the speedups of the MV1, MV2 and MM2 implementations appear to deviate from the ideal ones. However, the speedups of the MV3, MV4, MM3 and MM4 implementations are increased linearly as the number of workstations is increased. Further, from the experimental results we observe that the speedup curves improve as the block size is increased. Therefore, we conclude that the block size is an important parameter which can affect the overall performance. More specifically, this parameter is directly related to the I/O and communication factors. So, the low communication and I/O cost is obtained for large values of block size. However from experimental Figures there is worst performance for very small values of block size (i.e. $b = 1, 2$) because produce a poorly balanced load.

Another parameter which can affect the performance of parallel implementations is the matrix size. From Figures, we can see that the speedup curves of parallel matrix implementations increase slightly as the matrix size is increased. We note that the speedup curves of the MV3, MV4, MM3 and MM4 implementations are very close to the ideal system computing power, $\Sigma_{j=1}^p W_j$ for all number of workstations. This occurs because when the matrix size is large enough, very much matrix computations
are performed in parallel at each workstation. However, the communication overhead between the master workstation and worker workstations is not very significant. Therefore, the ratio of the computation time to communication time is high enough.

Finally, we can note that the parallel implementations like MV1, MV2, MM1 and MM2 occur lower speedups than in case of using the other implementations according to (Typou et al, 2004a; Typou et al, 2004b). From extensive experimental study has been shown that there are two basic reasons for the performance degradation of the MV1, MV2, MM1 and MM2 implementations (Michailidis et al, 2005). The primary factor affecting the performance of the implementations are high communication requirements between master and workers. Another reason for the performance degradation is due to the high cost of reading of data from the local disk of the master workstation for small values of block size. The I/O time could be improved when the matrix is read and partitioned in chunks of large size.

5. CONCLUSIONS

The four parallel implementations of matrix computations are presented and implemented on a cluster platform. These implementations are based on row block decomposition scheme and the cluster platform considered in this paper employ a master - worker model. Moreover, we presented the experimental results of the proposed implementations in the form of performance graphs. From our results, we found that the implementations with low communication requirements such as MV3, MV4, MM3 and MM4 are more suitable for heterogeneous clusters.

Further, we have introduced an effective performance prediction model and methodology, which can help to users to predict the execution time and similar performance metrics of the proposed matrix implementations for two kinds of cluster (homogeneous and heterogeneous). It has been shown that the performance model is able to predict the parallel execution time of parallel matrix implementations accurately.

REFERENCES


