A Parallel Approach to the Solution of Sparse Linear Systems

M. V. Ribeiro Jr.[1], J. S. Aude[2]

[1]IPRJ – Rio de Janeiro Polytechnic Institute
Friburgo, RJ, Brazil
{mvicente@uol.com.br}
[2]IM and NCE - Federal University of Rio de Janeiro
PO. Box 2324 – Rio de Janeiro – RJ – 20001-970, Brazil
{salek@nce.ufrj.br}

Abstract—

This paper discusses the parallel implementation of direct methods for the solution of positive definite sparse linear systems. The adopted procedure consists of four phases: ordering with the use of the approximate minimum degree algorithm; symbolic factorization; Choleski numerical factorization; and solution of the final triangular system. It has been implemented on a 6 processor IBM SP2 platform with the use of MPI. Only the Choleski factorization phase has been implemented in parallel. However, different permutation matrices are concurrently processed during the ordering and symbolic factorization phases by the available processors. The result with the lowest number of fill-ins is the one processed by the numerical factorization. The system performance has been evaluated with the use of matrices extracted from the Harwell-Boeing set. Speed-up factors above 4 have been achieved with 6 processors. In addition, over 30% fill-in reduction has been obtained by the concurrent processing of alternate ordering possibilities.

Keywords— **sparse linear systems, minimum degree ordering algorithm, Choleski factorization, MPI, parallelism**

I. Introduction

As pointed out by Duff [DUF 97], the solution of large sparse linear systems is frequently required in a wide range of engineering and scientific problems, such as: air traffic control, astrophysics, VLSI circuit simulation, oil reservoir simulation, structural analysis, optimization, etc. In addition, recent studies [SAA 98] show that over 70% of the computational time spent on supercomputers by scientific applications is due to work related to solving large scale linear systems. Therefore, a significant improvement on the performance of such applications can be produced by providing efficient computational methods for solving such systems. This observation and the current availability of parallel processing platforms have been the motivation for this work.

In this paper, techniques for the parallel implementation of direct methods for solving sparse positive definite linear systems are analyzed and evaluated through the development of a complete package for solving such systems using MPI [PAC 97] on a 6 processor IBM SP2 platform [AGE 95]. The adopted direct method for solving sparse linear systems consists of four phases: minimum degree ordering, symbolic factorization, Choleski numerical factorization and triangular system solution. The Choleski factorization has been implemented in parallel with the use of different approaches for workload distribution among the processors. The ordering and symbolic factorization phases are concurrently processed by the available processors which explore alternative ordering paths. The performance of the implemented system has been evaluated with the use of 7 matrices extracted from the Harwell-Boeing collection.

Section 2 of this paper briefly describes the four processing phases which are used by direct methods for solving sparse linear systems. Section 3 describes the basic minimum degree ordering algorithm and explains the operation of the adopted approximate minimum degree algorithm. In addition, this section also describes the symbolic factorization phase and the concept of elimination trees. The adopted approach for exploiting the availability of a set of processors to improve the quality of the ordering algorithm is also discussed in Section 3. Section 4 describes the Choleski numerical factorization procedure and presents the adopted technique for its parallel implementation. Section 5 describes and analyses the experimental results obtained on the IBM SP2 – MPI platform. These results have been obtained for a set of matrices considering different approaches for the workload distribution among the processors during the Choleski factorization phase. Finally, Section 6 summarizes the main conclusions and comments on some natural evolutions of this work.

II. Direct Methods for Solving Sparse Linear Systems

Two common approaches for solving linear systems are the direct methods and the iterative methods, such as successive over-relaxation (SOR) and conjugate gradient. This work focuses on the use of direct methods based on the Choleski factorization to solve positive definite sparse linear systems.

Let us consider the linear equation system Ax=b, where *A* is a symmetric positive definite matrix, *b* is a known vector and *x* is the solution vector to be computed. With the use of the Choleski factorization, it is possible to find a lower triangular *L* matrix with diagonal positive elements such that $A=LL^T$. Therefore, the solution vector *x* can be computed through the solution of the following triangular systems: Ly=b and $L^Tx=y$.

If *A* is a sparse matrix, the factorization to compute *L* will often produce non-zero elements in *L* where the original elements of *A* were zero. Such non-zero elements which are created during the factorization are known as *fill-in*. In order to save memory space and to reduce the number of arithmetic operations to be performed, it is desirable to reduce the number of fill-ins to a minimum. It is known that the solution of a linear system Ax=b does not depend on the order in which the rows and/or columns of *A* appear. In addition, as *A* is positive definite, numerical stability is achieved without pivoting. Therefore, the rows and columns of *A* can be reordered to reduce the fill-in generation. So, if *P* is the chosen permutation matrix, the resulting PAP^T matrix is also symmetric positive definite and the solution of the original system is given by: Ly=Pb, $L^Tz=y$, $x=P^Tz$.

Unfortunately, finding *P* in order to minimize the fill-in generation is an NP-complete problem [YAN 81]. Therefore, some sort of heuristics, as the minimum degree [ROS 70, ROS 73, DUF 74, DUF 82, GEO 80a, GEO 80b, GEO 81, GEO 89, LIU 85, AME 96] or the nested dissection [GEO 73, GEO 78] algorithms, has to be adopted to solve this problem.

Once a permutation matrix *P* is found, it is possible to determine the locations of the non-zero elements of *L* and, consequently, to define the structure which will store the elements of *L*. This process of determining such a structure is called symbolic factorization. Following this step, the lower triangular matrix *L* is computed as a factor of PAP^T within the numerical factorization phase, which is the most time consuming step of the overall procedure and the one which offers the most suitable opportunities for the efficient use of parallelism, since, due to the sparsity of the coefficient matrix, the factorization work for some columns can be processed in parallel.

Finally, the fourth and last phase solves the linear triangular system by back substitution. It requires much less computation than the numerical factorization and has only a limited amount of parallelism. The parallel implementation of this phase has been studied by Alvarado et al. [ALV 98] but it is out of the scope of this work due to its limited benefit to the overall procedure performance.

III. Ordering and Symbolic Factorization

In this work, the minimum degree algorithm has been used for solving the ordering problem. The application of this algorithm often leads to good results in relation to fill-in generation. Rose [ROS 70, ROS 73] has developed the model based on graph theory for this algorithm. Advances in relation to the algorithm computational time and memory requirements have been reported in several works by Duff and Reid [DUF 74, DUF 82] , George and Liu [GEO 80a, GEO 80b, GEO 81, GEO 89], Liu[LIU 85] and Amestoy et al. [AME 96].

The structure of non-zero elements of a symmetric n x n matrix can be represented by a graph $G^0 = (V^0, E^0)$ with nodes $V^0 = (1, \dots, n)$ and edges E^0. An edge (i, j) is in E^0 if and only if $A[i, j] \neq 0$ and $i \neq j$. As *A* is symmetric, G^0 is a non-directed graph. The elimination graph G^k describes the structure of the non-zero elements of the sub-matrix to be factorized after the first *k* pivots have been chosen and eliminated. At step *k*, the new graph G^k depends on the previous graph G^{k-1} and on the pivot *p* which is chosen at step *k* from V^{k-1}. Edges are added to E^{k-1} to make the nodes adjacent to *p* in G^{k-1} a clique. The addition of edges corresponds to the generation of fill-ins. Once the pivot node *p* is chosen, this node and its incident edges are removed from the graph G^{k-1} to produce the graph G^k.

The problem with the use of elimination graphs is that, due to the addition of edges which represent the generation of fill-ins, it is not possible to know the storage requirements of the graph structure in advance. In contrast to the elimination graphs, the quotient graphs model the factorization of the *A* matrix using at most the amount of storage needed for the original graph G^0 [GEO 81]. Within a quotient graph, a clique is represented by a list of its members rather than by a list of all the edges in the clique. In the quotient graphs, *elements* are the nodes which have been removed from the corresponding elimination graph and *variables* are the nodes which have not been eliminated yet.

The concept of supervariables can be used to reduce the number of iterations to be performed by the minimum degree algorithm. Two variables *i* and *j* are indistinguishable if they have the same set of adjacent nodes in the quotient graph and, therefore, they have the same degree. So, if variable *i* is selected as the pivot at the algorithm step *k*, variable *j* can be selected as the pivot at step *k + 1* without producing any additional fill-in.

The minimum degree algorithm selects the node *p* as the pivot at step *k* such that the degree of *p* is minimum. Therefore, it is a greedy heuristic which tries to minimize the number of edges (fill-ins) added to the elimination graph at each step. The most time consuming step of the minimum degree algorithm is the evaluation of each graph node degree. In this work, it has been adopted the same approach proposed by Amestoy et al. [AME 96] in the approximate minimum degree algorithm. In this algorithm, quotient graphs are used and just upper bounds for the node degrees are determined. The resulting algorithm is faster than traditional minimum degree algorithms and the number of generated fill-ins is comparable to the best results achieved with those algorithms.

After finding the permutation matrix P in the ordering phase, the next step of the direct method procedure performs the symbolic Choleski factorization of the resulting PAP^T matrix. No floating point operation is performed, but the number of symbolic operations at this phase is the same as the number of floating point operations within the numerical factorization phase. The goal of the symbolic factorization is to set up the data structures needed for storing the triangular matrix which will result from the numerical factorization by predicting the amount and the location of the fill-ins which will be produced.

It can be shown [GEO 81] that the structure of the non-zero elements of a column j in matrix L is given by the union of the strictly lower triangular structure of that column in matrix A with the structure of every column in L which has its first off-diagonal non-zero element in row j. Fig. 1 illustrates the application of this simple idea to find out the structure of the L matrix which will be derived from the factorization of the original A matrix. In this figure, "X" represents the non-zero elements in the original A matrix and "+" represents the *fill-ins* that will be generated in matrix L.

$$A = \left|\begin{array}{cccccccc}
X & & & & X & & & X \\
 & X & & X & X & & & \\
 & & X & X & & & & X \\
 & X & X & X & & & X & \\
X & X & & & X & X & & \\
 & & & & X & X & & X \\
 & & & X & & & X & \\
X & & X & & & X & & X
\end{array}\right|$$

$$L = \left|\begin{array}{cccccccc}
X & & & & & & & \\
 & X & & & & & & \\
 & & X & & & & & \\
 & X & X & X & & & & \\
X & X & & + & X & & & \\
 & & & & X & X & & \\
 & & & X & + & + & X & \\
X & & X & + & + & X & + & X
\end{array}\right|$$

Fig. 1 Original Matrix A and Structure of Matrix L After Symbolic Factorization

Once the structure of the non-zero elements of the resulting L matrix is determined, the data dependencies among columns regarding the numerical factorization step can be analyzed. This is very important for an efficient parallel implementation of the numerical factorization procedure. The elimination tree is the structure used for this analysis. In this tree, each node represents an L matrix column and the descendants of a node k are the nodes associated with the columns which have to be factorized before column k can be. The node k parent in the elimination tree is the smallest j node such that $j > k$ and $L[j, k] \neq 0$. Fig. 2 illustrates the elimination tree derived from the L matrix shown in Fig. 1.

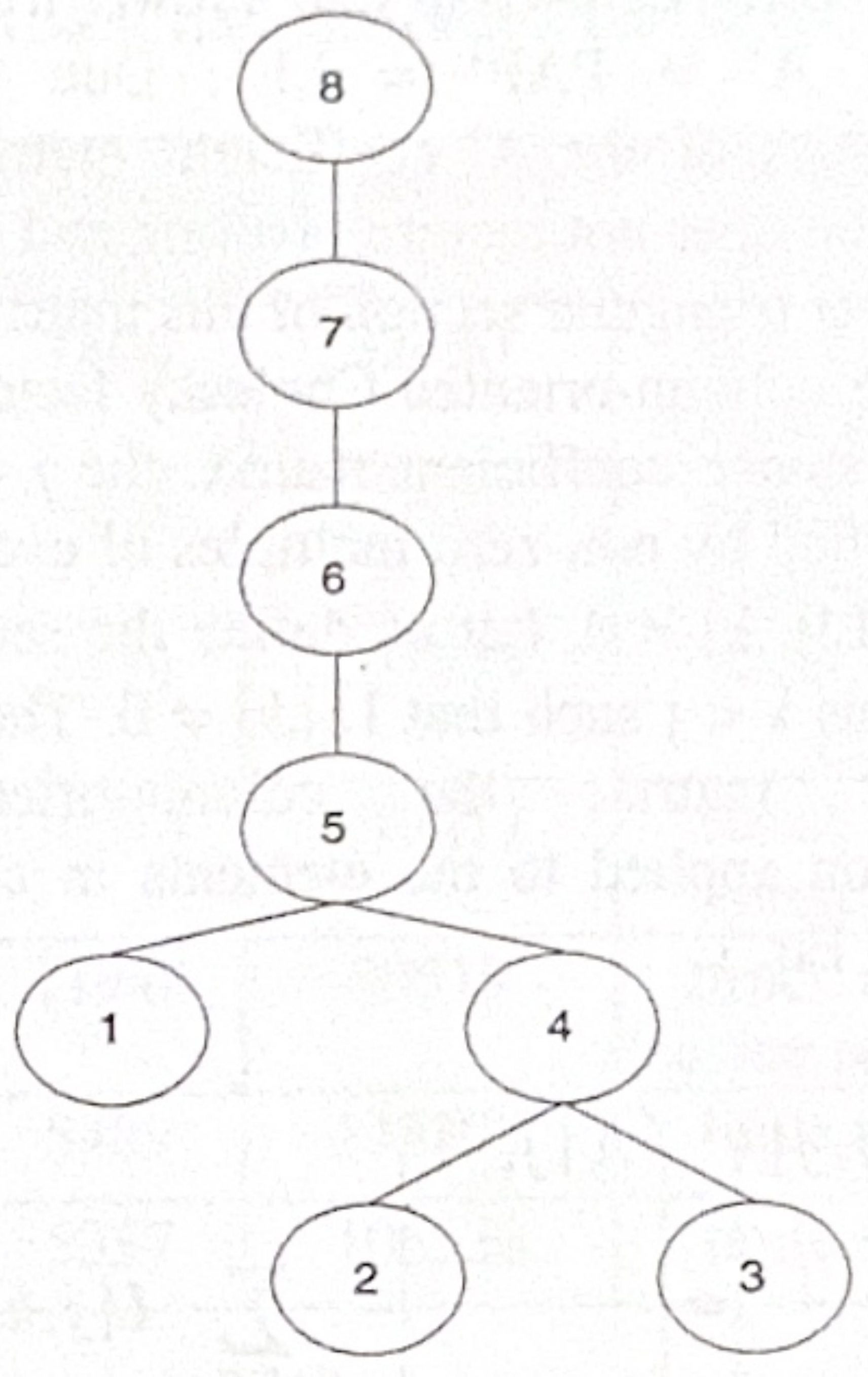

Fig. 2 The Elimination Tree

Both the minimum degree ordering algorithm and the symbolic factorization procedure have very efficient sequential implementations. Therefore, little or no benefit is achieved with the parallel implementation of these processing phases. Nevertheless, the result produced by the ordering algorithm has great impact on the efficiency of the numerical factorization phase parallel implementation, since it defines the number of fill-ins and the elimination tree shape. It has also been observed [KUM 94] that the sparsity pattern of the resulting L matrix is sensitive to tie-breaking between candidate pivots during the application of the minimum-degree ordering algorithm. Different permutations of a matrix satisfying the minimum-degree criterion may produce different degrees of fill-in and parallelism during the factorization phase.

Therefore, the availability of processors on the IBM SP2 platform has been used in this work to concurrently exploit different matrix permutation alternatives due to the existence of more than one candidate pivot at different steps of the minimum degree ordering procedure. These different matrix permutations are processed concurrently throughout the ordering and symbolic factorization phases, and the one which produces the best result in relation to fill-ins is chosen to be processed within the parallel implementation of the Choleski numerical factorization phase.

IV. Numerical Factorization

General systems numerical factorization is based on the Gaussian elimination, which solves the following equation within a triple loop by varying the values of *i*, *j* and *k*: A[i, j] = A[i, j]- (A[i, k]*A[k, j])/A[k, k].

For symmetric positive definite systems, the Choleski factorization can be used to find a lower triangular matrix *L* such that A' = PAP^T = LL^T. Due to the specific characteristics of the *A'* coefficient matrix, the Choleski factorization does not require pivoting and can operate only on the lower triangular section of this matrix.

For the column-oriented Cholesky factorization applied to the *A'* sparse coefficient matrix, the *j* column of *A'* is only modified by non-zero multiples of every column k < j such that L[j, k] ≠ 0. Let us define the set S[j] containing the columns k < j such that L[j,k] ≠ 0. Then, since *A'* is a symmteric matrix, the column-oriented Choleski factorization applied to the elements in column *j* can be defined by Eq. 1.

$$\begin{pmatrix} L[j,j] \\ . \\ L[n,j] \end{pmatrix} = \begin{pmatrix} A'[j,j] \\ . \\ . \\ A'[n,j] \end{pmatrix} - \sum_{K \in S \mid K<j} L[j,k] \begin{pmatrix} L[j,k] \\ . \\ . \\ L[n,k] \end{pmatrix}$$

$$\begin{pmatrix} L[j,j] \\ . \\ . \\ L[n,j] \end{pmatrix} = \frac{1}{\sqrt{L[j,j]}} \begin{pmatrix} L[j,j] \\ . \\ . \\ L[n,j] \end{pmatrix}$$

Eq. 1: Choleski Factorization

These equations assume that the factorization process is applied from left to right along the columns of the *A'* coefficient matrix. So, when the *j* column is under factorization, all the *k* columns to the left of j (k < j) have already been factorized and, therefore, the lower triangular matrix elements for those columns are already known (L[*, k]).

The numerical factorization is typically the most time consuming phase of the overall procedure. In order to speed up its processing, the work related to columns belonging to disjoint sub-trees of the elimination tree can be performed in parallel. Under this scheme, communication between any pair of processors *p* and *q* is required whenever processor *p* is associated with at least one elimination tree node which is a child of one of the nodes assigned to processor *q*. In this case, processor *p* has to send to processor *q* the result of the summation shown in Eq. 1.

For the definition of the parallel implementation of the column-oriented Choleski factorization on a message passing system, let us consider that the function *map[k]* returns the processor *p* assigned to the processing of column *k* by some workload distribution policy. Fig. 3 shows the basic codification of the algorithm. The processor identification is given by *my_id* and *my_set* is the set of coefficient matrix columns assigned to the processor with identification *my_id*.

```
begin
  $my\_set = \{ k \mid map[k] = my\_id \}$
  for $j = 1$ to $n$ do
      $S = \{k \mid (k < j \text{ and } L[j,k] \neq 0)\}$
      if ($my\_set \cap S \neq \emptyset$) or ($j \in my\_set$)
      begin
        $u = (0, \ldots, 0)^T$
        for $k \in (my\_set \cap S)$ do
        $u = u + L[j,k]\,(L[j,k], \ldots, L[n,k])^T$
        if $j \notin my\_set$ then
             send vector u to processor p such that
                                 $p = map[j]$
        else
             $L[*,j] = (A'[j,j], \ldots, A'[n,j])^T - u$
             while there are columns to send data to me
               do
                           receive data in vector v
                           $L[*,j] = L[*,j] - v$;
                           $L[*,j] = \frac{L[*,j]}{\sqrt{L[j,j]}}$
end;
```

Fig. 3 Message Passing Codification of the Parallel Cholesky Factorization

It is interesting to point out that in the procedure shown in Fig. 3, vector *u* is obtained from the summation of calculations performed on columns assigned to the local processor. Therefore, this calculation does not require any communication. So, if the workload distribution among processors is done in order to reduce the number of situations where parent and child nodes in the elimination tree are assigned to different processors, the amount of communication in the algorithm implementation can be drastically reduced.

V. Implementation Aspects And Experimental Results

The performance of the parallel Choleski factorization has been evaluated through the full implementation of the overall system to solve sparse positive definite linear systems on an MPI/IBM-SP2 platform. The experiments have been performed with 7 matrices obtained from the Harwell-Boeing set and with the use of different approaches to the workload distribution among the available processors.

The IBM-SP2 system which has been used consists of 6 thin processors running at 66.7 MHz, each one with 256 MB of memory and 2 MB of level 2 cache, interconnected by a 40 Mbytes/s switch and running the IBM C compiler version 3.66 and the AIX 4.2.6 operating system.

As stated previously, within the minimum degree ordering phase, all the available processors are used to explore different permutation alternatives concurrently. So, whenever there is a list of pivot candidates at any step of the algorithm, each processor randomly chooses a pivot candidate to process. Therefore, at the end of the ordering phase, each processor may have generated a different permutation matrix *P*. The number of fill-ins produced by each permutation is evaluated within the symbolic factorization phase which is also independently processed by each processor. At this moment, an MPI collective communication operation, MPI_Allreduce, is used to determine the permutation that produced the lowest fill-in and the corresponding processor. This result is transmitted to all the other processors which cooperatively perform the numerical factorization phase considering the use of this permutation matrix.

Before performing the numerical factorization, the system distributes the work to be done among the *n* available processors. Three different workload distribution approaches have been used. The first one, D1, associates each column *j* with the processor *p* such that $j \bmod n = p$. It is a simple approach which tries to assign the same number of columns to the processors without taking data locality into consideration.

The second distribution approach, D2, searches the elimination tree bottom-up for an intermediate level of the tree consisting of *n* nodes. The processing of the columns associated with each of the corresponding *n* sub-trees is assigned to one of the *n* available processors. From that level up, the remaining work tends to be assigned to a smaller number of processors because the elimination tree converges to a single root node. In this distribution process, however, data locality is always an important issue and additional work associated with one particular node of the tree is usually assigned to one of the processors which is going to process at least one of the descendant sub-trees of that node.

The third workload distribution approach, D3, scans the elimination tree bottom-up and evaluates the number of descendants of each node. Whenever this number gets close to or above a certain threshold value, defined as a function of the total number of tree nodes and the number of available processors *n*, the whole sub-tree that has this node as a root is assigned to one processor. This procedure is repeated until all processors have sub-trees assigned to them and until all the tree nodes are assigned to a processor. Once again, after the initial assignment of work to every processor, data locality is the adopted criterion for the assignment of additional work to a particular processor.

Table I summarizes the main characteristics of the 7 matrices used in the development of the experimental work.

TABLE I
MAIN CHARACTERISTICS OF THE TEST MATRICES

name	size	#non-zero elem.	Application
Bcsstk14	1806	32630	struct. analysis (OmniColiseum)
Bcsstk15	3948	60882	struct. analysis (offshore platform)
Bcsstk16	4884	147631	struct. analysis (dam)
Bcsstk17	10974	219812	struct. analysis (pressure tank)
Bcsstk18	11948	80519	struct. analysis (nuclear power plant)
S1rmq4m1	5489	143300	finite elements
S3rmt3m3	5357	106526	finite elements

For every matrix in Table I, experiments have been carried out to evaluate the performance of the Choleski factorization procedure considering the use of 1, 2, 4, and 6 processors and the application of the three workload distribution approaches previously described. For each experiment, Fig. 4 to Fig. 10 show the achieved speed-up. In addition, Table II shows the number of fill-ins generated in each case by the best permutation matrix found by the set of processors in use.

The results show that the combination of the parallel implementation of the Choleski factorization with the reduction of the number of fill-ins by the concurrent evaluation of alternative permutation matrices in the ordering phase have in general produced good speed up results. Distribution D3 has been a clear winner in most cases. Only for Bcsstk14 matrix a poor speed up (below 2) has been achieved with the use of this distribution and 6 processors. Nevertheless, as shown in Table II, no other distribution approach has succeeded in producing a better result.

It is interesting to observe that for Bcsstk17 matrix, a speedup factor above the number of processors has been achieved with the use of distribution D3 and 4 processors. This kind of result is possible because the speed up factor is not only a result of parallelism but also a result of the reduction of fill-ins produced (over 30%) when 4 processors were in use during the ordering phase, as shown in Table II. Only with Bcsstk15 matrix no reduction at all in the number of fill-ins has been observed for any number of processors in use during the ordering phase.

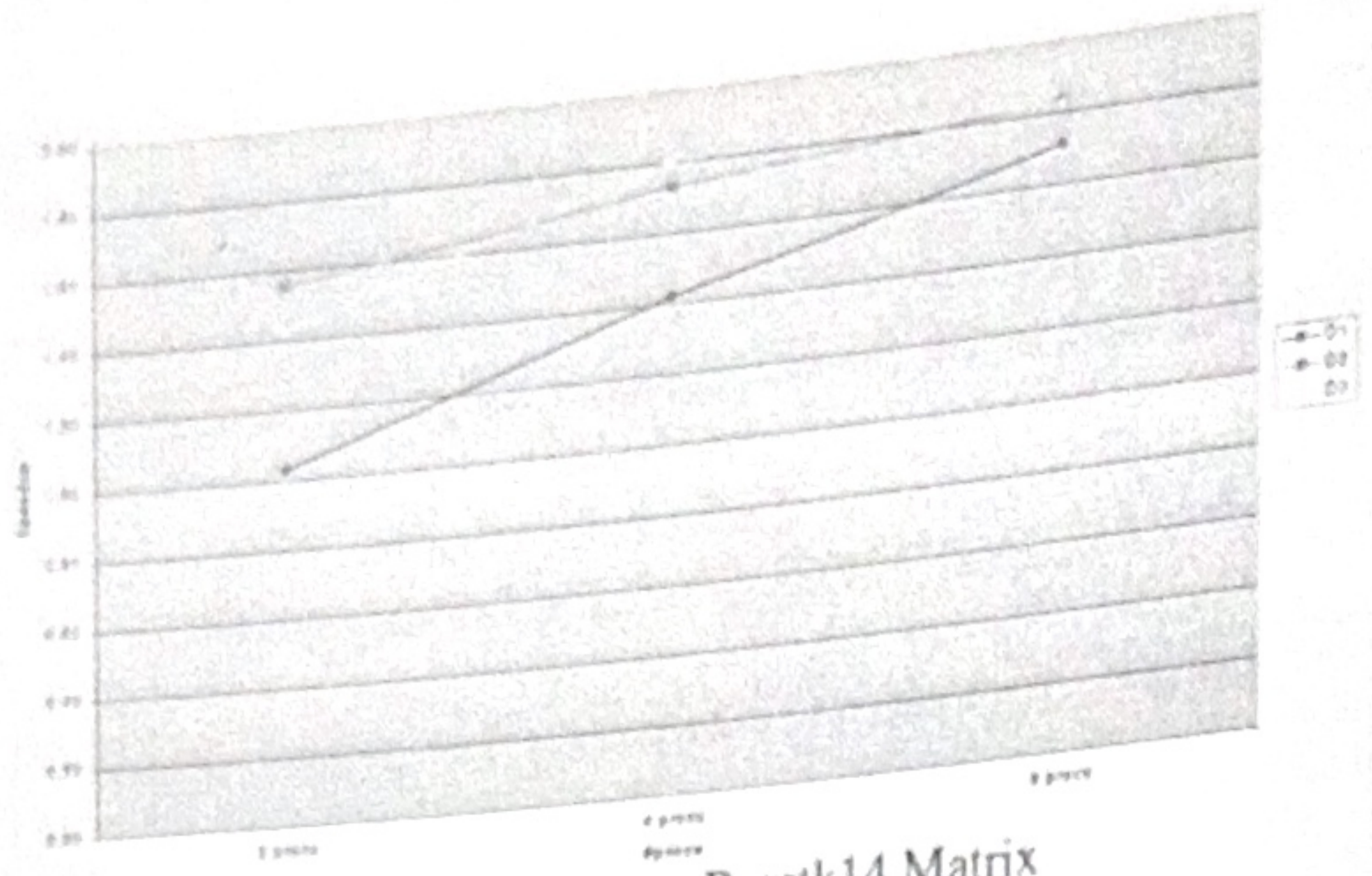

Fig. 4: Speed-up - Bcsstk14 Matrix

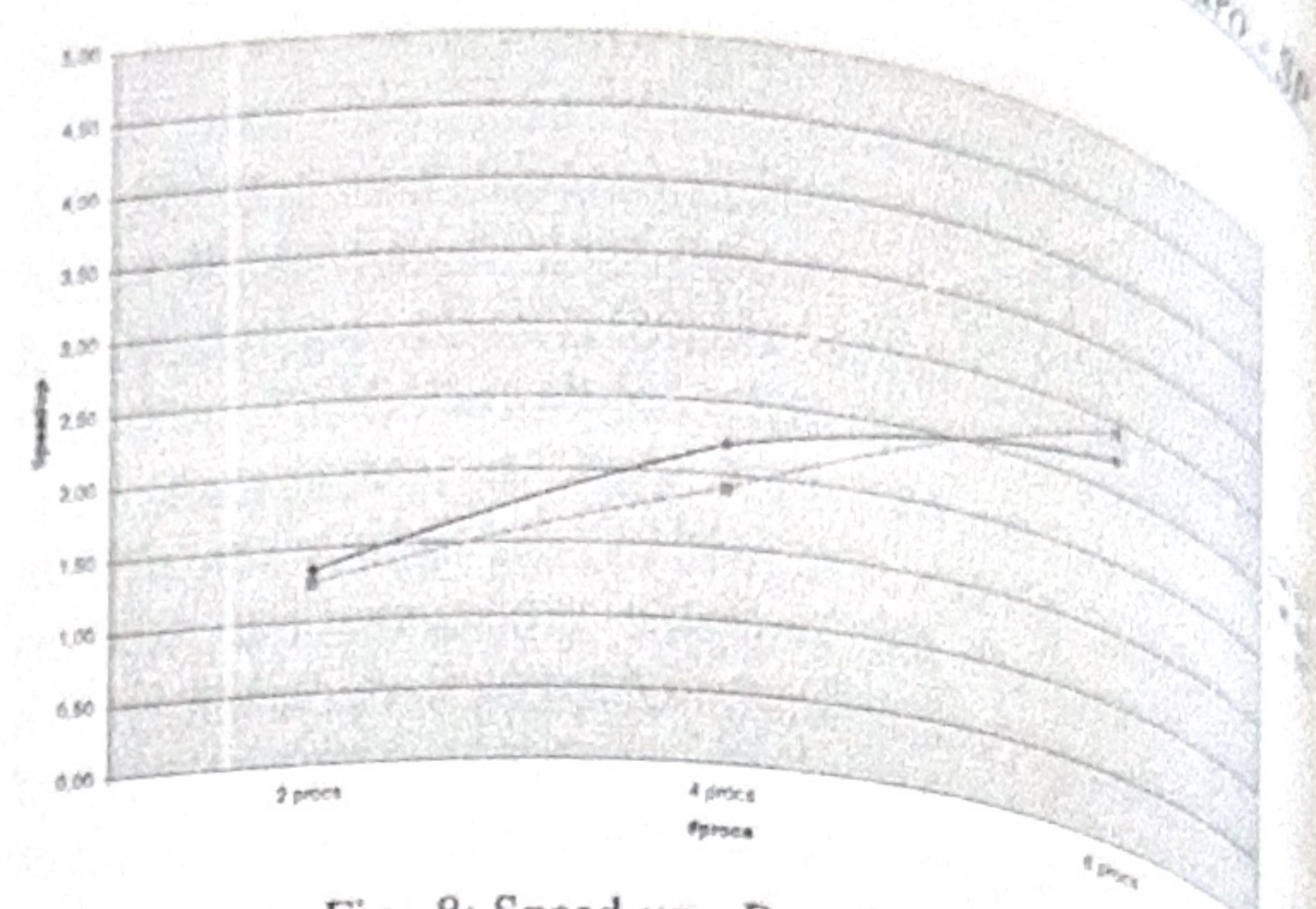

Fig. 8: Speed-up - Bcsstk18 Matrix

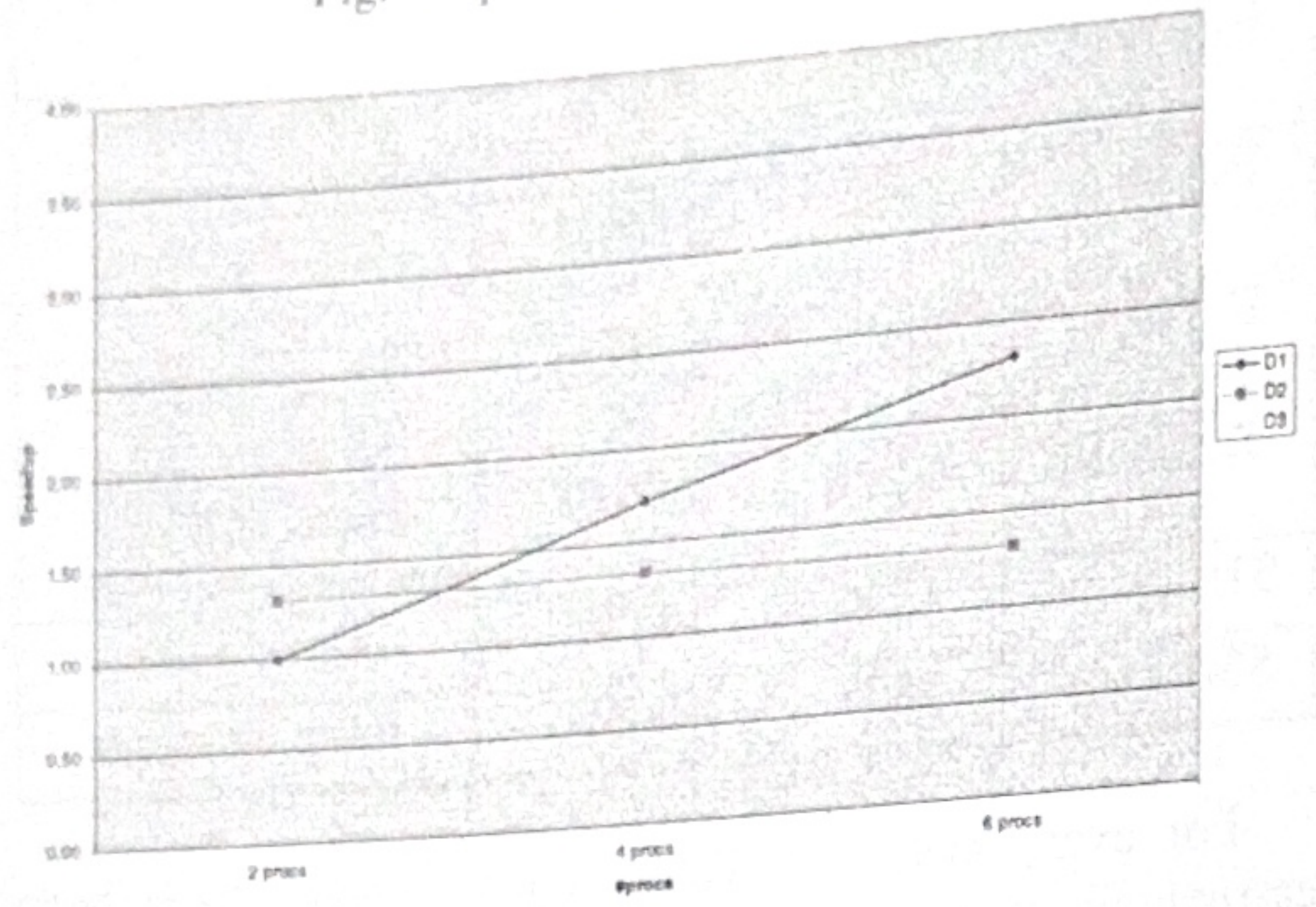

Fig. 5: Speed-up - Bcsstk15 Matrix

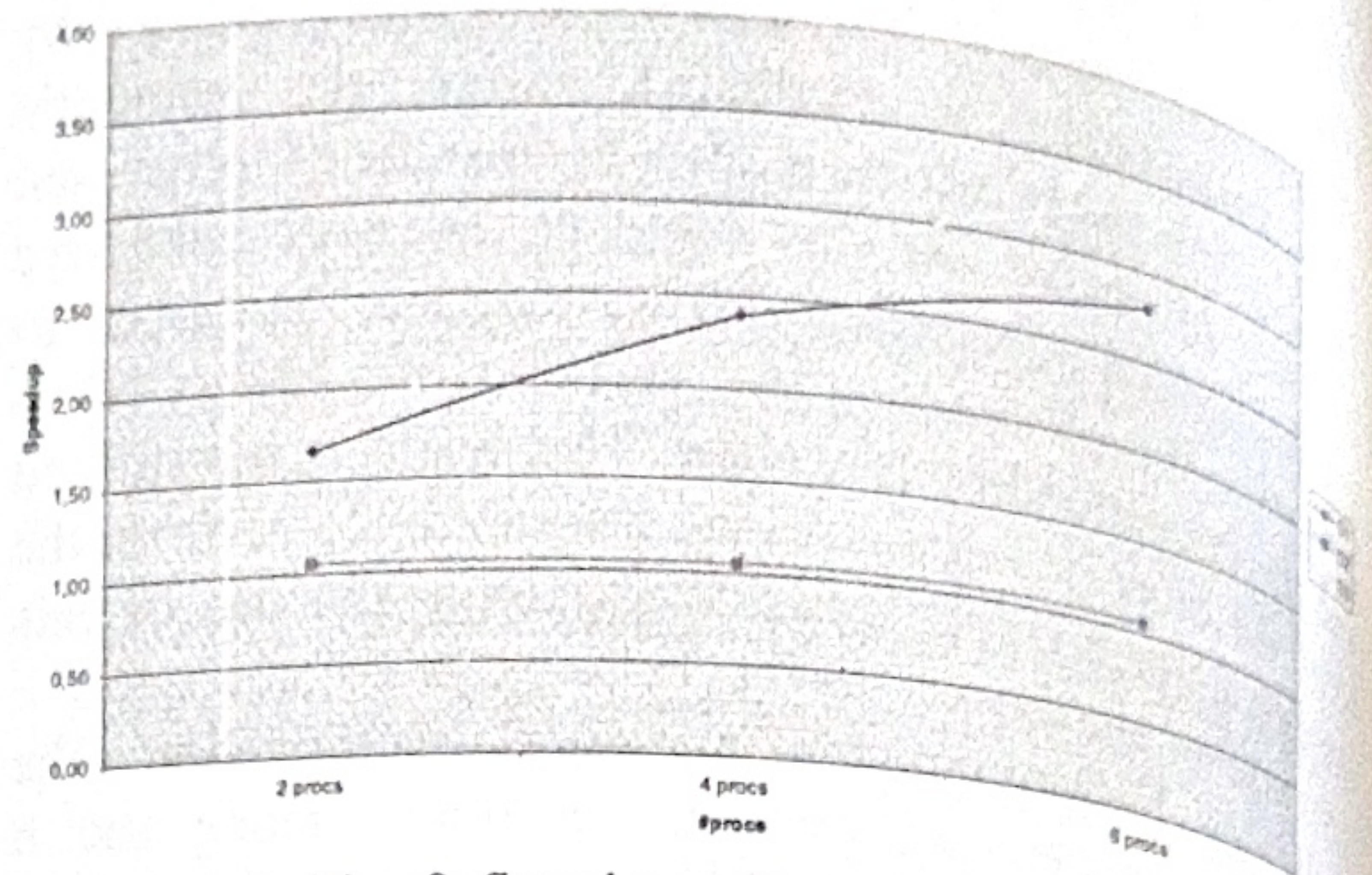

Fig. 9: Speed-up - Slrmq4m1 Matrix

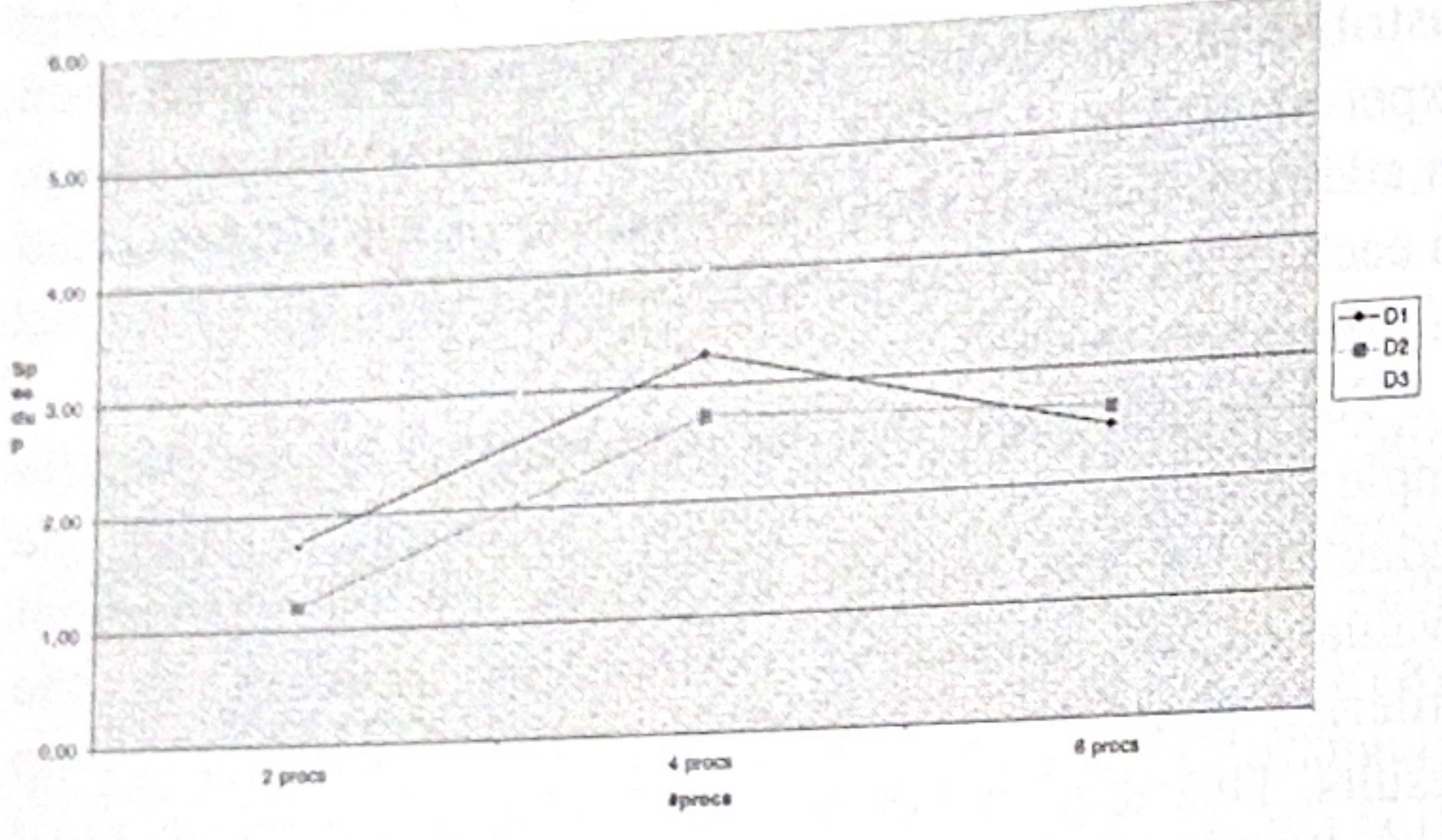

Fig. 6: Speed-up - Bcsstk16 Matrix

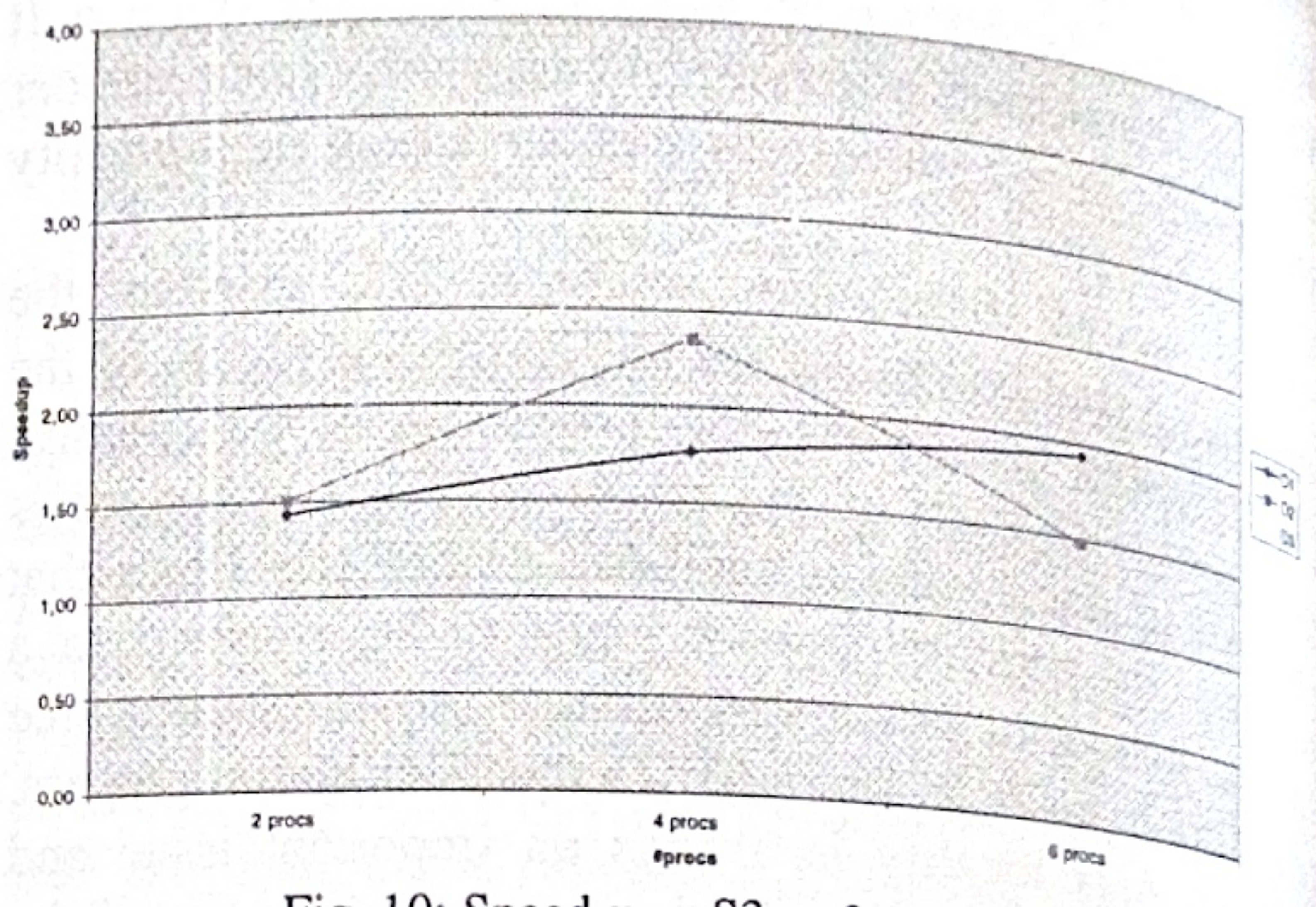

Fig. 10: Speed-up - S3rmt3m3 Matrix

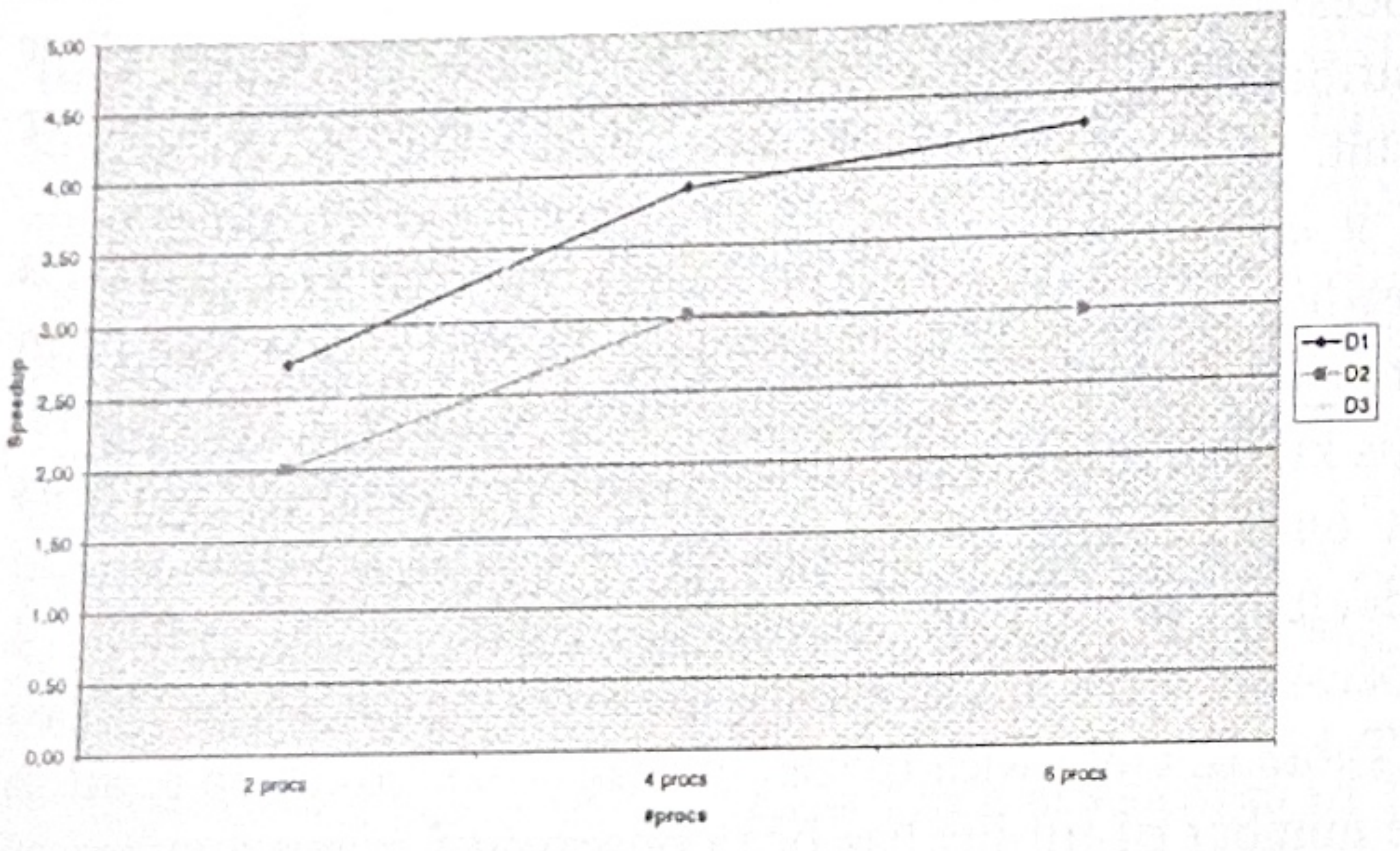

Fig.7 : Speed-up - Bcsstk17 Matrix

TABLE II
FILL-IN GENERATION

	1 proc.	2 proc.	4 proc.	6 proc.
Bcsstk14	111020	102299	98805	98805
Bcsstk15	990000	990000	990000	990000
Bcsstk16	1418065	1418065	1203896	1184408
Bcsstk17	2283651	1693007	1560706	1560706
Bcsstk18	884423	884423	781431	693762
S1rmq4m1	1118358	980271	980271	980271
S3rmt3m3	635400	600375	600375	600375

In relation to the number of communication messages required by the parallel algorithm implementation, Table III clearly shows that distribution D2 is the winner and distribution D1 displays always the worst result. Nevertheless, distribution D2 has often produced the worst results in relation to speed up because it has consistently failed to find the best workload balance among the processors. This problem is aggravated by the typical elimination tree shape produced by minimum degree ordering algorithms, since distributions D2 and D3 would tend to yield similar results if a more balanced tree were generated by the ordering algorithm, as it would probably happen with the nested dissection algorithm.

TABLE III

NUMBER OF COMMUNICATION MESSAGES

	2 processors			4 processors		
	D1	D2	D3	D1	D2	D3
Bcsstk14	1698	288	895	4919	601	1652
Bcsstk15	3323	1339	2937	9575	2096	7973
Bcsstk16	4721	1290	2500	13862	1846	5218
Bcsstk17	9911	819	5183	28718	1810	12116
Bcsstk18	8569	1075	4898	21292	1959	12027
S1rmq4m1	5443	205	1554	16155	226	1957
S3rmt3m3	5307	736	1662	15767	958	1816

	6 processors		
	D1	D2	D3
Bcsstk14	7917	720	2213
Bcsstk15	15402	2370	11868
Bcsstk16	22416	1905	5018
Bcsstk17	46297	1870	14883
Bcsstk18	31220	1058	17160
S1rmq4m1	26601	205	2068
S3rmt3m3	26032	736	2017

It should be pointed out that all workload distribution approaches tried to evenly divide the total number of columns among the processors. However, the amount of actual work to be done for each column is not constant. It is in fact proportional to the number of non-zero elements in each column. Therefore, no distribution approach has been able to produce well balanced workload distribution among the processors for all combinations of the number of processors and matrix types. As an illustration, Fig. 11 and Fig. 12 show the resulting running time distribution among the processors for the processing of the Bcsstk18 matrix with the use of the distributions D2 and D3, respectively. As can be seen, in this particular case, distribution D3 has achieved a very good workload distribution among the processors while distribution D2 hasn't. However, such good result produced by distribution D3 does not hold for some of the other matrices.

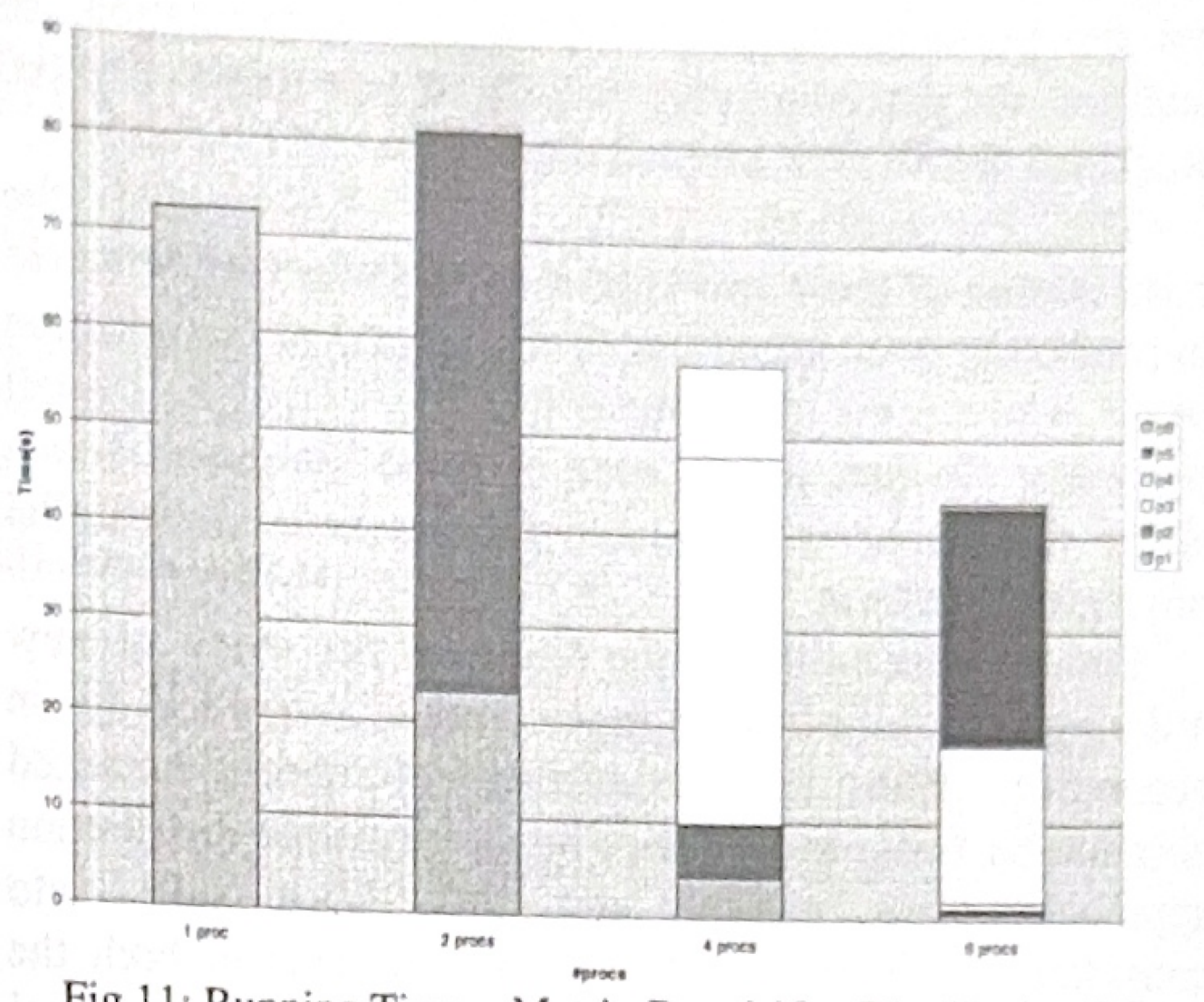

Fig.11: Running Time – Matrix Bcsstk18 – Distribution D2

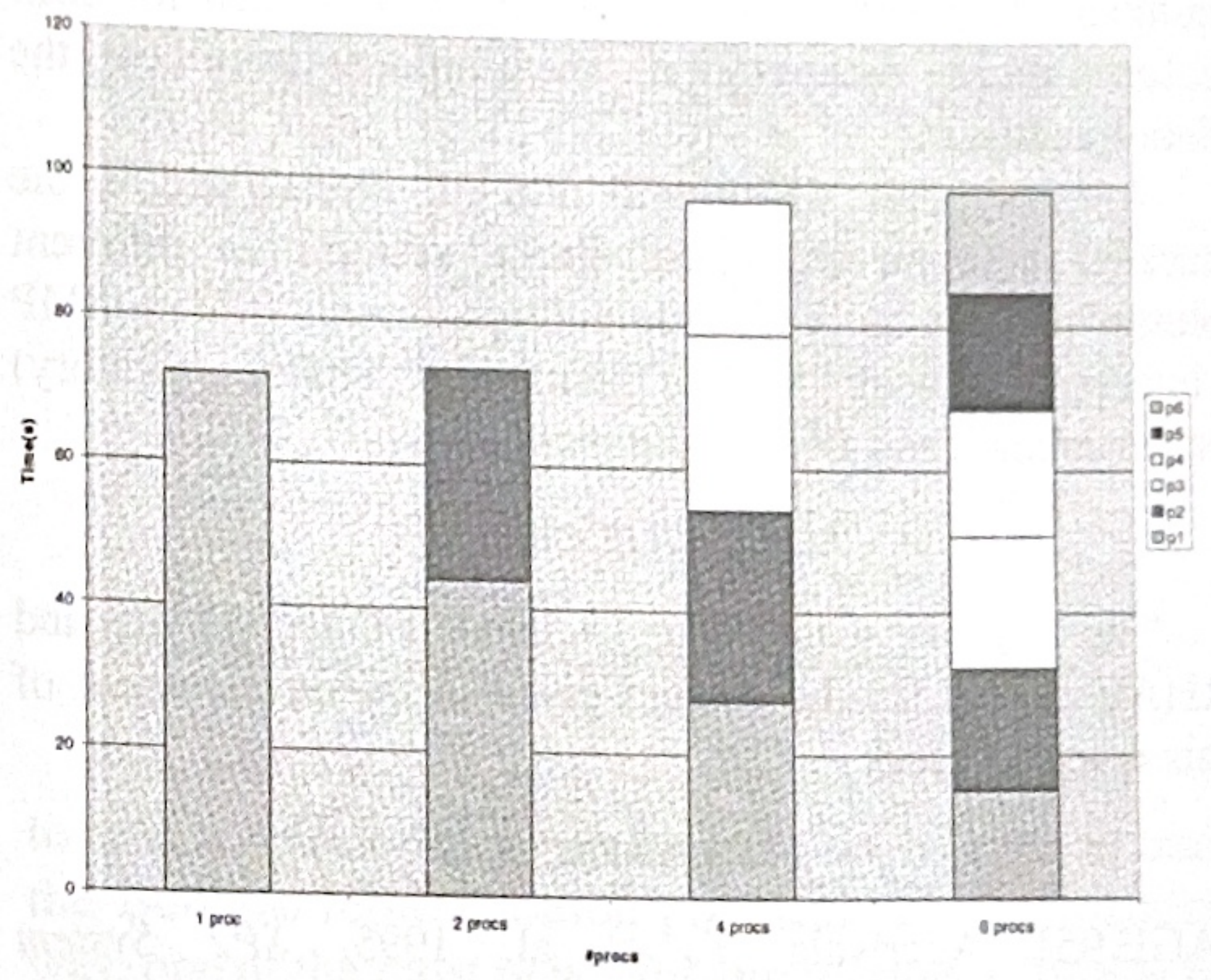

Fig. 12: Running Time – Matrix Bcsstk18 – Distribution D3

VI. CONCLUSIONS AND FUTURE WORK

In this work two different approaches to parallel processing have been adopted within the four processing phases of direct methods applied to the solution of positive definite sparse linear systems. The first approach is applied to the minimum degree ordering and symbolic factorization phases and its goal is to use the available processors to concurrently explore alternative minimum-degree permutations generated at different steps of the ordering process. Each processor performs in fact a totally independent task and the end result is a fill-in reduction at no extra cost.

The second parallel processing approach is adopted within the Choleski numerical factorization phase. In this case, a more cooperative pattern among the processors has been used to perform a column-oriented factorization. The result produced by the ordering phase in relation to both the number of fill-ins and the elimination tree shape are critical for the performance of this parallel implementation. In addition, the procedure performance is very sensitive to the workload distribution adopted approach.

The experimental results have shown that the combination of these two approaches can be very effective in producing good speed-ups in the numerical factorization phase, which is the most time consuming step of the overall procedure. In fact it can even produce speed-up factors above the number of available processors in the parallel processing platform.

Future work will mainly focus on the study of new ordering approaches which may combine a very low fill-in generation with the production of more balanced elimination trees. In addition, better workload distribution approaches are needed and they must take into consideration, in more detail, several aspects of both the problem and the platform, such as: the cache size, the actual number of arithmetic operations to be performed for each column during factorization, the communication cost, the data locality, etc.

Another issue which will also be investigated is the parallel solution of sparse linear systems in different platforms such as symmetric multiprocessors (SMP), SMP clusters, software DSM (distributed shared memory) environments and DSM multiprocessors.

ACKNOWLEDGEMENTS

The authors would like to thank FINEP, CNPq and RHAE, Brazil for the support given to the development of this research work

REFERENCES

[AGE 95] Agerwala T. et al, 1995, *SP2 System Architecture*, IBM Systems Journal, Vol 34, No.2 – Scalable Parallel Computing.

[ALV 98] Alvarado F. L., Pothen A., Schreiber R.,1998, *Highly Parallel Sparse Triangular Solution,* Sparse Matrix Computations : Graph Theory Issues and Algorithms, IMA Volumes in Mathematics and its Applications, Springer-Verlag, Univ. of Minnesota.

[AME 96] Amestoy P. R., Davis T.A., Duff I.S.,1996, *An Approximate Minimum Degree Algorithm*, Siam J. Matrix Anal. Appl.,17-4, pp. 886-905.

[DUF 74] Duff I.S., Reid J. K.,1974, *A Comparison of Sparsity Orderings for Obtaining a Pivotal Sequence in Gaussian Elimination,* J. Inst. Math. Appl.,14, pp. 281-291

[DUF 82] Duff I.S., Reid J. K,1982, *MA27- A Set of Fortran Subroutines for Solving Sparse Symmetric Sets of Linear Equations,* Tech. report, AERE R10533, HMSO, London.

[DUF 97] Duff I.S., Erisman A.M., Reid J.K., 1997,*Direct Methods for Sparse Matrices*, Oxford University Press, Oxford, UK

[GEO 73] George J.A.,1973, *Nested Dissection of a Regular Finite Element Mesh*, Siam J. Numer. Anal.,10, pp. 345-363.

[GEO 78] George J. A., Liu J. W. H.,1978, *An Automatic Nested-Dissection Algorithm for Irregular Finite Elements Problems*, Siam J. of Numer. Anal.,15, pp. 1053-1069.

[GEO 80a] George A., Liu J. W. H.,1980, *A Fast Implementation of the Minimum Degree Algorithm Using Quotient Graphs*, ACM Trans. Math. Software,6, pp. 337-358.

[GEO 80b] George A., Liu J. W. H.,1980, *A Minimal Storage Implementation of the Minimum Degree Algorithm*, Siam J. Numer. Anal.,17, pp. 282-299.

[GEO 81] George A., Liu J. W. H.,1981, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice Hall, Englewood Cliffs, NJ.

[GEO 89] George A., Liu J. W. H.,1989, *The Evolution of the Minimum Degree Ordering Algorithm*, Siam Rev., 31, pp. 1-19.

[KUM 94] Kumar V., Grama A., Gupta A., Karypis, G., 1994, *Introduction to Parallel Computing: Design and Analysis of Algorithms*, The Benjamin/Cummings Publishing Company

[LIU 85] Liu J. W. H., 1985, *Modification of the Minimum Degree Ordering by Multiple Elimination,* ACM Trans. Math. Software,11, pp. 141-153.

[PAC 97] Pacheco P.S., 1997, *Parallel Programming with MPI,* Morgan Kaufmann Publishers, San Francisco, California.

[ROS 70] Rose D. J.,1970, *Symmetric Elimination on Sparse Positive Definite Systems and Potential Flow Network Problem,* Ph.D. Thesis, Harvard University.

[ROS 73] Rose D.J., 1973, *A Graph-theoretic Study of the Numerical Solution of Sparse Positive Definite Systems of Linear Equations,* Graph Theory and Computing, R. C. Read, ed., Academic Press, New York, pp. 183-217.

[SAA 98] Saad Y., December 1998, *Supercomputing Institute – Research Bulletin,* Volume 15 Number 1, University of Minnesota, Minneapolis.

[YAN 81] Yannakakis M.,1981,*Computing the Minimum Fill-In is NP-Complete*, Siam J. Alg. Disc. Math., 2, pp. 77-79.