PARALLEL SPARSE MATRIX SOLVERS FOR EQUATION-BASED PROCESS FLOWSHEETING

Alan B. Coon and Mark A. Stadtherr*

We examine two orderings for the large, sparse matrices typically occurring in EB flowsheeting applications and present a new parallel, direct method with which these orderings are used. Some previously considered methods for the parallel solution these matrices are based on a strategy which exploits their inherent bordered block-diagonal structure. These methods are susceptible to processor load imbalances and, consequently, low parallel efficiencies. The methods under study are designed to maintain a uniform processor load distribution throughout the linear solving step while still exploiting large task granularity to reduce communication overhead. Results for the reordering strategies are given for several test problems.

INTRODUCTION

The much faster computational rates that can be achieved through the use of parallel computing hold several potential benefits for process simulation and design (flowsheeting). These increases in computing power not only make possible the use of more realistic unit models involving complex phenomena and the tractability of larger simulation problems, but they will also result in the increased productivity of the design engineer (Stadtherr and Vegeais (1)). Although many parallel computers have extremely fast clock speeds, their ability to achieve high computational rates is also the result of the extensive use of various forms of parallel architecture within each of those computers. If a program which is implemented on one of these machines is to execute at such high computational speeds, it must use algorithms which effectively exploit that parallel architecture. The efficient solution of large, sparse sets of linear equations is a critical requirement in equation-based (EB) process flowsheeting; thus, the parallel implementation of a sparse linear solver is crucial.

We will confine our attention to the identification and exploitation of large grain parallel tasks. Although there are many uses of the term granularity in the literature (*e.g.*, see Kruskal and Smith (3)), we use it here to indicate the amount of time spent calculating relative to the time spent communicating (with global memory or other processors). Our objectives in this approach are not only to reduce interprocessor communication, but to reduce processor synchronization delays and load imbalances. These goals can be achieved if the algorithm we use can be written in a high level language with independent procedure calls of approximately equal duration. One should note that this does not preclude the use of a low level parallel strategy to execute the individual tasks (by low level parallelism we mean concurrency exploited on the DO LOOP level). On multivector processors, this may

* Department of Chemical Engineering, University of Illinois, Urbana, IL 61801, U.S.A.

be achieved by assigning each task to a different processor, and exploiting the vector architecture of the processor in the execution of the task. On multiprocessors, one might assign more than one processor to each task if there are idle processors and the tasks exhibit some fine grain parallelism. Many of today's vectorizing and parallelizing compilers can identify much of this low level parallelism, and the next generation of such compilers should be even more effective in detecting low level parallelism. Such use of fine grain parallelism within the tasks generated by the course grain approach allows a multiple level concurrency that significantly enhances the overall parallel performance of the algorithm (*e.g.*, see Coon and Stadtherr (3)).

The methods we propose below are extensions of large grain parallel strategies for matrices whose associated graphs have a predetermined structure (*e.g.*, those arising from finite difference applications) (3). With such graphs, good separators and bisections can be readily identified and a parallel ordering based on a nested dissection strategy can be used. Because of the more general structure of EB flowsheeting matrices, good separators and bisections are not so easily identified, and the parallel performance of these strategies is critically dependent on finding good separators.

The use of heuristic algorithms for finding separators in graphs of general sparse matrices was originally considered in the context of reducing the sequential complexity of the LU factorization by Lipton *et al.* (4), and has been investigated more recently as a means of producing an ordering for parallel LU factorization of symmetric matrices by Leiserson and Lewis (5). This latter approach, in which the relative size of the connected components can be bounded, is equivalent to a parallel implementation of nested dissection. This idea can be adapted to generate a parallel ordering for EB flowsheeting matrices that will not result in a severe processor load imbalance. This adaption to the case of the nonsymmetric EB flowsheeting matrix entails using the occurrence graph of $A+A^T$. The orderings that we can generate for the occurrence matrix of the symmetric part can then be used for the original matrix (if two pivots can be processed concurrently in $A+A^T$, they can be processed concurrently in A as well). Unfortunately, using the symmetric part of the matrix to generate a parallel ordering can often preclude the identification of parallelism that exists in the nonsymmetric structure of the matrix.

We examine here an alternate approach in which a heuristic bisection algorithm is used to identify a block tridiagonal ordering for EB flowsheeting matrices. A graph theoretical model that includes a description of the asymmetry of the matrix is used, so that this method is not subject to the same pitfalls as the method mentioned above. The reordered matrix can be solved using a parallel block tridiagonal scheme that is a generalization of a method for solving narrow banded matrices. We present the results of these ordering strategies as applied to typical EB flowsheeting matrices.

BLOCK TRIDIAGONAL APPROACH

Generalization of Narrow-Banded Solver

This approach is motivated by an algorithm for the parallel solution of narrow banded linear systems presented by Dongarra and Sameh (6). In this algorithm, the narrow banded system is partitioned into block-tridiagonal form. For P processors and a system of order N with semi-bandwidth b, the diagonal blocks of this partition will have order N/P. (Here, we assume that N is a multiple of P; otherwise, the first P-1 diagonal blocks will have order $\lceil N/P \rceil$.) Each of the off-diagonal blocks is also of order N/P, and contains all zeros except for either an upper or lower triangular matrix of order b (Figure 1a). The

diagonal blocks can be factored concurrently, with each factorization assigned to a different processor. Left multiplication of the system by the inverse of the block diagonal part creates fill-in only above the lower triangular submatrix or below the upper triangular submatrix within each off-diagonal block (Figure 1b). (In the actual implementation, one does not compute the inverse of the block diagonal part and premultiply with it, rather one computes the fill-in columns by forward elimination and back substitution using the factorization of the diagonal block and the appropriate off-diagonal block column as the right-hand side.) Again, these computations can be done concurrently by assigning one block row to each processor. We can include partial or threshold pivoting within the factorization of each diagonal block, and we assume that this is enough to maintain numerical stability.

At this point, an independent system of order 2b(P-1) remains, as depicted by the solid outline in Figure 1b. This reduced system comprises the b equations above and the b equations below each block row. It can be solved independently and the rest of the overall solution obtained by substitution (which can be executed concurrently), or the overall system can be repartitioned into block-triangular form with diagonal blocks of order 2N/P. In this latter case (in which we assume $P = 2^k$ for some integer k) the factorization, premultiplication, and repartitioning are applied recursively until a reduced system of order 2b remains, at which point this reduced system is solved and rest of the overall solution is determined by concurrent substitution. One should note that for this recursive procedure, the diagonal blocks of the repartitioned matrix are not dense and can be factored as indicated in Figure 1.

The general structure of EB flowsheeting matrices is nearly block-banded, with some nonzero blocks occurring outside of the block bands (*e.g.*, Vegeais and Stadtherr (7), Vegeais (8)) feedforward and recycle streams are typically represented by these off-band blocks. One might consider reordering the EB flowsheeting matrix with a bandwidth reduction algorithm (using the structure of the symmetric part) and applying the narrow banded strategy to the result. Investigations of bandwidth and profile reduction algorithms for EB flowsheeting matrices in (8) indicate that the resulting matrices have fairly large bandwidths with sparse bands. Since the size of the reduced system is proportional to the bandwidth, this would not appear to be a good strategy. For EB flowsheeting matrices, however, most equations have very few nonzero coefficients. This indicates that the matrix could be reordered so that it has areas of small local bandwidth, *i.e.*, subsets of equations with small bandwidths. The key observation is that small bisections correspond to small local bandwidths.

We now consider an adaptation of a bisection algorithm to find a more general block-tridiagonal ordering to which we can apply the above strategy. Thus, the above strategy will not be limited to narrow banded systems. For the graph $G = (X,E)$ (of the occurrence matrix of $A + A^T$) of order N, and a given partitioning of X into two disjoint subsets, X_1 and X_2, of orders $\lceil N/2 \rceil$ and $\lfloor N/2 \rfloor$, respectively, we define the <u>bisection</u>, $B \subset E$, to be the set of edges with one vertex in X_1 and the other vertex X_2. Given a bisection, B, of a graph and the vertex sets X_1 and X_2 into which the vertices are partitioned, a block-tridiagonal form with two diagonal blocks will result if we order the vertices as follows:

(a) all the vertices in X_1 are mapped into integers less than those associated with the vertices in X_2, and

(b) the vertices incident to B in X_1 are mapped into integers greater than those integers into which the rest of X_1 is mapped, while the vertices incident to B in X_2 are mapped into integers less than those integers associated with the rest of X_2.

(The latter condition is equivalent to requiring that vertices incident to B be numbered consecutively.) We will let b_1 denote the number of vertices in X_1 that are incident to B, and b_2 denote the number of such vertices in X_2. In the resulting matrix, the upper right off-diagonal block has zero entries everywhere except in the $b_1 x b_2$ subblock closest to the diagonal. The lower left off-diagonal block has zero entries everywhere except in its $b_2 x b_1$ subblock which is closest to the diagonal. If this system is left multiplied by the inverse of the block-diagonal portion of the matrix, fill-in will occur only above the $b_1 x b_2$ subblock and below the $b_2 x b_1$ subblock, analogous to the narrow banded case. A reduced system of b_1+b_2 independent equations results. We consider this to constitute the first level of the ordering.

This ordering can be applied recursively to the subgraphs induced by the vertex sets of the partition, in a manner similar to the recursive application of the nested dissection ordering described in (3). One should note, however, that the algorithm which finds the bisection must be modified slightly. This modification should prevent the algorithm from choosing a bisection with any incident vertices that are also incident to a bisection found at a previous level (since those vertices have already been made to satisfy the ordering criteria at that level). The recursive application of the ordering stops after the kth level, where the number of processors is 2^k. The resulting matrix will be block-tridiagonal with 2^k diagonal blocks, each of order $N/2^k$ (for simplification, we assume N to be a multiple of P). Figure 2 shows an example for two levels of this ordering. The size of the nonzero subblocks in the off-diagonal blocks will be determined by the number of vertices incident to the corresponding bisection. If the matrix is premultiplied by the inverse of the diagonal blocks (which can be factored concurrently), the fill-in is limited to the nonzero columns of the off-diagonal blocks. The resulting matrix contains a reduced system of independent equations whose order is equal to the number of vertices which are incident to all of the bisections found in the ordering stage.

As in the narrow-banded case, the reduced system can be solved and the entire solution then can found via substitution, or the system can be repartitioned to a block-tridiagonal form with 2^{k-1} diagonal blocks (Figure 2b). The factorization of the diagonal blocks, the premultiplication by the inverse (both of which can be executed concurrently), and the repartitioning can all be applied recursively until the remaining system has only two diagonal blocks. (Diagonal blocks of each repartitioned matrix have a sparse structure similar to that shown in Figure 1b and can be factored as indicated there.) At this point, the order of the reduced system of independent equations will be equal to the number of vertices incident to the bisection found at the first level. This reduced system can be solved and the rest of the solution is computed by substitution.

One should note that the order of the reduced system is determined by the number of vertices incident to the first level bisection. In the former case (that the reduced system is solved without recursive repartitioning), the order is determined by the number of vertices incident to all bisections. In the narrow banded case, the order of the reduced system was determined by the bandwidth. The number of vertices incident to a bisection can be thought of as a local bandwidth which is always less than or equal to the overall bandwidth (*e.g.*, Figure 2a).

Block Tridiagonal Reordering

Although we have sketched the overall idea of the reordering and numerical phases of this approach, we have not yet discussed how we account for the asymmetry of the EB flowsheeting matrix. In this section, we describe the graph model and the bisection algorithm we actually use to generate the ordering.

The ordering algorithm uses a bipartite graph of row vertices and column vertices to describe the nonzero structure of the matrix. A row vertex and a column vertex are adjacent if the corresponding matrix coefficient is nonzero. Thus, we can adequately describe a nonsymmetric matrix with this graph. Our algorithm assumes the input bipartite graph has a complete matching, or equivalently, the matrix has a full transversal. For each nonzero diagonal element, we say the corresponding row and column vertices are the match of one another. This model of the matrix structure is an extension of the net model used Schweikert and Kernighan (9) to find partitions of electrical circuits. In our model, each net corresponds to a column and the nonzeros elements in that column. The fundamental step in the algorithm is to find a partitioning of the vertices into two sets, such that each set has an equal number of row and column vertices and approximately the same number of total vertices as the other set, and so that the number of nets cut by the partition is minimized. (A net can be thought of as a generalization of an edge, and in that sense, this is essentially a bisection algorithm). This step is applied recursively (with a few further restrictions) to the resulting sets of vertices to obtain a partitioning into several sets. Information from such a partitioning can be used to order the matrix in a block tridiagonal form in which the off-diagonal blocks have very few nonzero columns (with each such nonzero column associated with a net that was cut by a partition). Once such an ordering has been determined, each diagonal block can be factored concurrently and the nonzero columns that correspond to a particular diagonal block are multiplied by the inverse of that diagonal block. We can extract from the resulting matrix a reduced linear system whose order is equal to the total number of nets that were cut by partitions.

We assume that input to the reordering algorithm is a matrix with a full transversal (*i.e.*, a complete matching in the bipartite graph) and tolerance of variation in the partition sizes. The algorithm generates a balanced two-way partition of the input matrix, corresponding to a two-way partition of the row and column vertices into X_1 and X_2 (where each subset contains as many column vertices as it does row vertices). The kernel of the algorithm involves the determination of which row or column vertex to move so that the number of nets cut by the partition (*i.e.*, the number of nets with elements in each partition) is the smallest. We will refer to this reduction in the number of nets cut as the gain of a vertex (as is done in (9)). Vertices are always moved in pairs, that is, every row vertex that is move from X_i to X_j, is followed by a row vertex move from X_j to X_i, or a column vertex move from X_i to X_j. A similar remark holds for column vertex moves. This requirement assures us that each subset will have as many rows as columns. Furthermore, if the second move is a row vertex from X_j to X_i, that row vertex must be adjacent to the match of the former row vertex and the former row vertex must be adjacent to the current match. Likewise, if the second move is a column vertex from X_i to X_j, it must be adjacent to the former row vertex and its match must be adjacent to the match of the former row vertex. This requirement preserves the full transversal. In general, the graph induced by the row and column vertices adjacent to any vertices moved as a pair must be a complete bipartite graph (specifically, $K_{1,1}$, an edge in the matching, or $K_{2,2}$). We choose the vertices with the largest gains from the set of rows and column vertices that satisfy these criteria. Figure 3 shows an example matrix and its graph both before and after the ordering.

The implementation we describe above allows the consideration of row exchanges without exchanging the corresponding columns, which is tantamount to considering alternate transversals. Hence, the overall reordering will not be constrained by the nonexistent data dependencies that reduce potential parallelism when the symmetric part is used. This model attempts to minimize the number of nets cut and does not weight the nets according to the number of vertices on a net. Hence, a net with many vertices is no more expensive to cut than a net with few vertices. This means that the nonzero areas of the off diagonal blocks can include any rows in the nonzero columns (not just those within the tridiagonal blocks). Because we apply the bisection algorithm recursively to each subset of the partition, this could potentially destroy the block tridiagonal. To maintain block tridiagonal structure during the recursive calls, we must further require that a partition be rejected if it cuts a net that was previously cut by another partition. This will result the ordering and reduced system depicted in Figure 4.

Nested Block Tridiagonal Ordering

If we allow a net to be cut by more than one partition, then there is the possibility that the columns corresponding to variables in the reduced system will suffer fill-in across partition boundaries. As mentioned above, this destroys the block tridiagonal structure, but the resulting structure, which we refer to as nested block tridiagonal, can still be exploited in a similar manner. The nested block tridiagonal structure and the resulting reduced system are shown in Figure 5. The modification to the numerical phase requires that the factored diagonal blocks be solved for the additional nonzero columns in each partition. As with the computation of the previous fill-in columns, these operations can be done concurrently, with each block row assigned to a different processor. The reduced systems resulting from this ordering are more dense than those resulting from the former ordering, but because this ordering is less restrictive in choosing candidate rows and columns for reordering, one expects that it will result in greater parallelism (more block rows) or a smaller reduced system. We show the results of these two orderings for several EB flowsheeting matrices in Table 2, with a description of the matrices in Table 1. The parameters which control the tolerance of relative partition size and relative number of nets cut by the partition were selected with a preference for a highly balanced four-way partition matrix (suitable for execution on a four processor machine like the Cray 2).

TABLE 1 - Descriptions of Test Problems (Problems 3-10 are from the Harwell/Boeing Computer Services Sparse Matrix Test Collection)

Problem	Order of Matrix	Nonzeros	Description
1	155	630	ammonia synthesis problem
2	350	1634	light hydrocarbon recovery
3	137	411	ethylene plant
4	207	572	heat exchanger network
5	225	1308	hydrocarbon separations
6	425	1339	nitric acid plant
7	156	371	simple chemical plant model
8	167	507	rigorous model of chemical stage
9	655	2854	16 stage column, some simplified
10	989	3537	7 stage column, all rigorous

TABLE 2 - Results for Block Tridiagonal and Nested Block Tridiagonal Reorderings

		Block Tridiagonal			Nested Block Tridiagonal		
Problem	Size	No. of Diagonal Blocks	Largest Diagonal Block	Size of Reduced System	No. of Diagonal Blocks	Largest Diagonal Block	Size of Reduced System
1	155	4	43	62	4	45	49
2	350	4	105	147	4	105	99
3	137	4	37	51	4	40	46
4	207	4	59	31	4	59	35
5	225	4	59	58	4	58	57
6	425	4	125	61	4	118	70
7	156	4	45	67	4	44	46
8	167	4	67	41	4	48	41
9	655	4	364	87	4	194	207
10	989	4	290	224	4	529	86

CONCLUSIONS

As indicated in Table 2, the algorithms are capable of producing fairly balanced partitions for the graphs of most of these problems. Because the tolerance parameters were selected to give a tightly balanced four-way partition, the reduced systems that result from the indicated orderings can be larger than the diagonal blocks for some problems. If such a large reduced system were costly to solve (*e.g.*, if interprocessor communication costs are high enough to preclude the use of a fine grained dense solver), then its size could be reduced by relaxing the partition and/or cut net tolerances.

For the problems in which the load balancing of the two orderings is similar (*i.e.*, the largest diagonal blocks are about the same size), the nested block tridiagonal ordering does tend to give smaller (albeit, more dense) reduced systems. In the test problems, however, this trend is not as pronounced as one might suspect (although we conjecture that it will be much more pronounced for larger, sparser problems). The circumstances under which one should relax the load balancing criterion to obtain a smaller reduced system, the optimal choice of tolerance parameters, and the choice of ordering are issues that are dependent on the parallel architecture under consideration, as well as on the problem being solved. We are hopeful that our current studies of the parallel solution time for these methods will partially resolve these issues.

SYMBOLS USED

A^T = the transpose of matrix A

B = a bisection, subset of an edge set

b_i = number of vertices in vertex subset X_i incident to bisection B

G(X,E)= a graph on X vertices and E edges

$K_{i,j}$ = the complete bipartite graph on sets of i and j vertices

N = the order of matrix A

P = number of parallel processors used

X_i = a subset of vertex set X

$\lceil z \rceil$ = the least integer greater than or equal to z

$\lfloor z \rfloor$ = the greatest integer less than or equal to z

ACKNOWLEDGEMENTS

This work was supported in part by the National Science Foundation under grant DMC-8520663 and uses the Cray 2 system at the National Center for Supercomputing Applications at the University of Illinois, and the Alliant FX-8 at the Center for Supercomputing Research and Development at the University of Illinois.

REFERENCES

1. Stadtherr, M. A. and J. A. Vegeais, "Advantages of Supercomputers for Engineering Applications," *CEP*, **21**, 21-24 (1985).

2. Kruskal, C. P. and C. H. Smith, "On the Notion of Granularity," *J. Supercomputing*, **1**, 395-408 (1988).

3. Coon, A. B. and M. A. Stadtherr, "Parallel Implementation of Sparse LU Decomposition for Chemical Engineering Applications," *Comput. & Chem. Eng.*, accepted for publication (1989).

4. Lipton, R. J., D. J. Rose and R. E. Tarjan, "Generalized Nested Dissection," *SIAM J. Numer. Anal.*, **16**, 346-358 (1979).

5. Leiserson, C. E. and J. G. Lewis, "Orderings for Parallel Sparse Symmetric Factorization," Third SIAM Conference on Parallel Processing for Scientific Computing, Los Angeles (1987).

6. Dongarra, J. J. and A. H. Sameh, "On Some Parallel Banded System Solvers," *Parallel Computing*, **1**, 223-235 (1984).

7. Vegeais, J. A. and M. A. Stadtherr, "Sparse Matrix Strategies for Equation-Based Chemical Process Flowsheeting on Advanced Computer Architectures: I. Vector Processing," submitted for publication (1989).

8. Vegeais, J. A., "Vector and Parallel Strategies for Sparse Matrix Problems in Equation-Based Chemical Process Flowsheeting," Ph.D. Thesis, Univ. of Illinois, Urbana, Illinois (1987).

9. Schweikert, D. G. and B. W. Kernighan, "A Proper Model for the Partitioning of Electrical Circuits," Proc. of the ACM-IEEE 9th Design Automation Workshop, Dallas (1972).

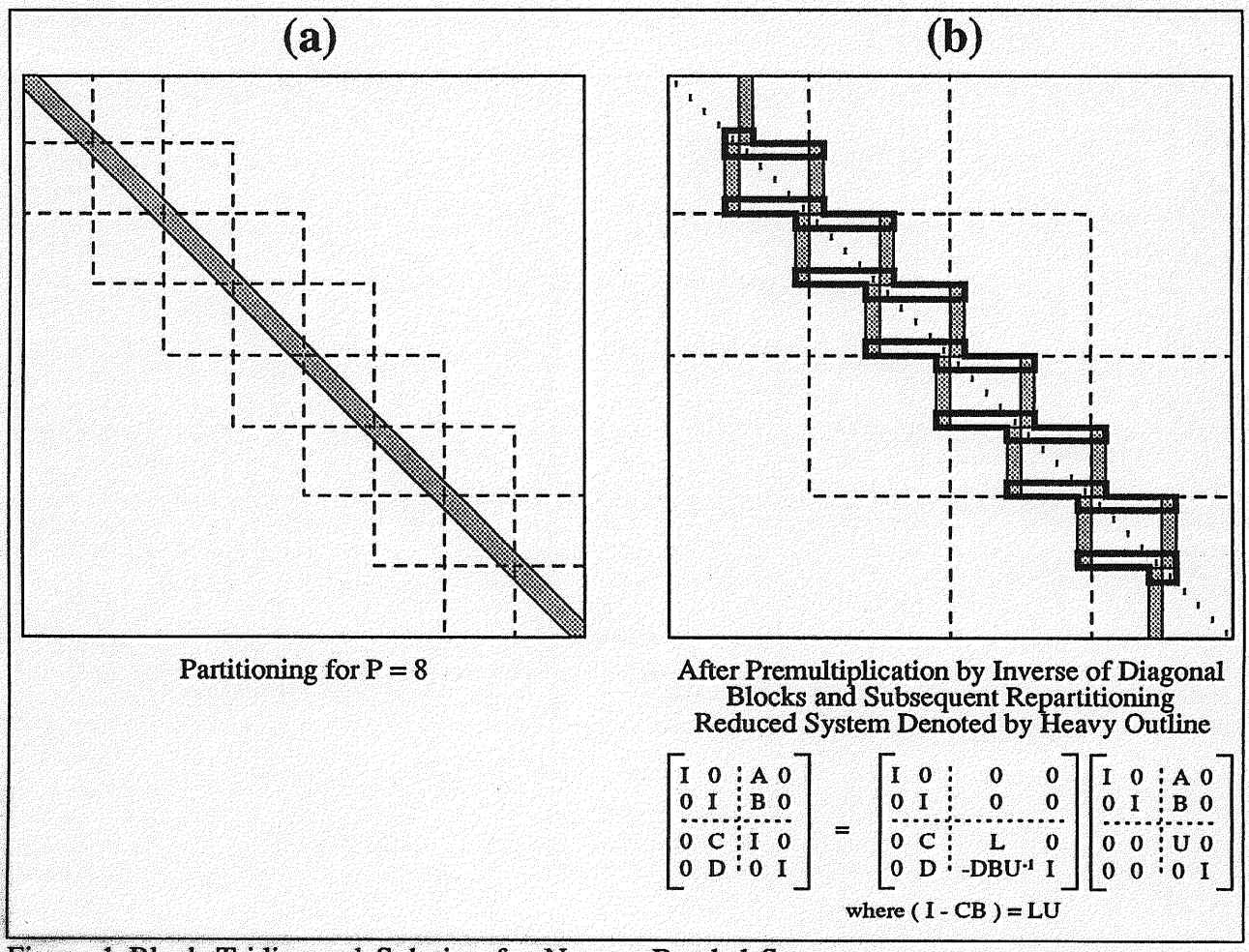

Figure 1 Block Tridiagonal Solution for Narrow Banded System

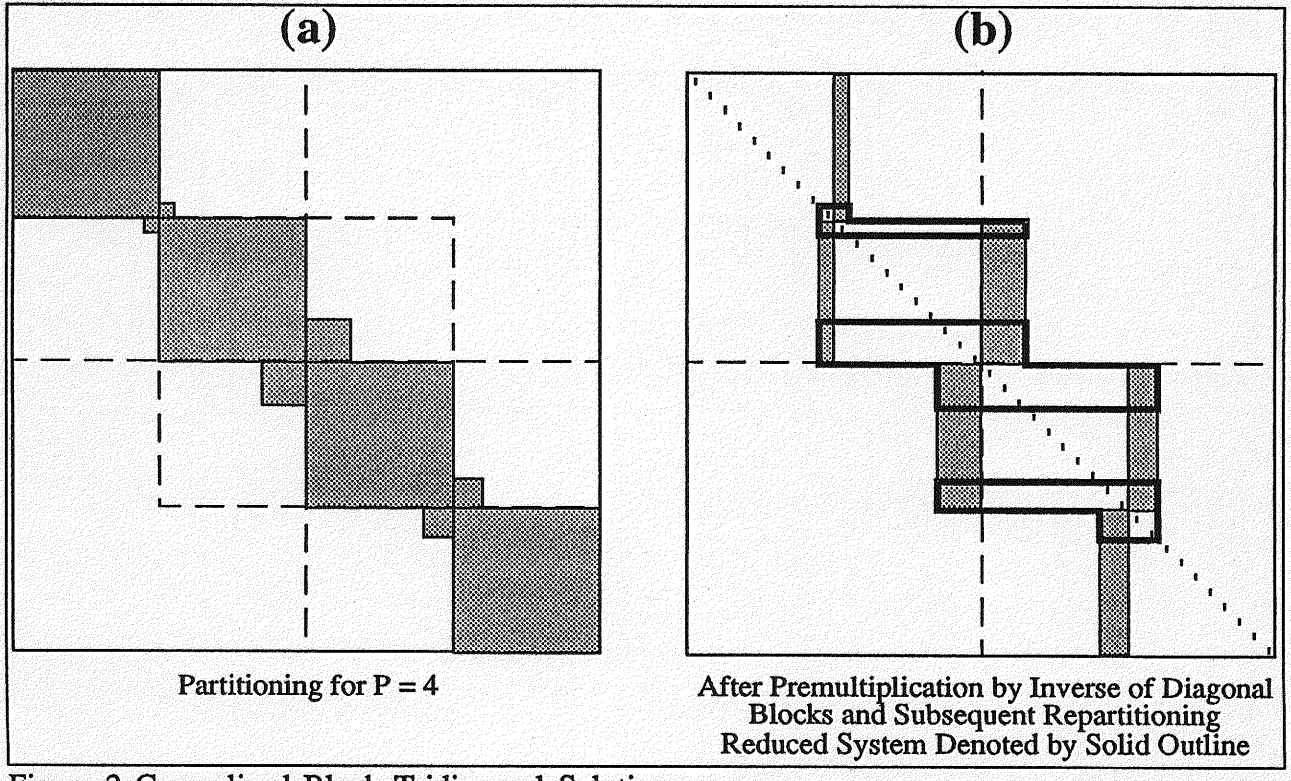

Figure 2 Generalized Block Tridiagonal Solution

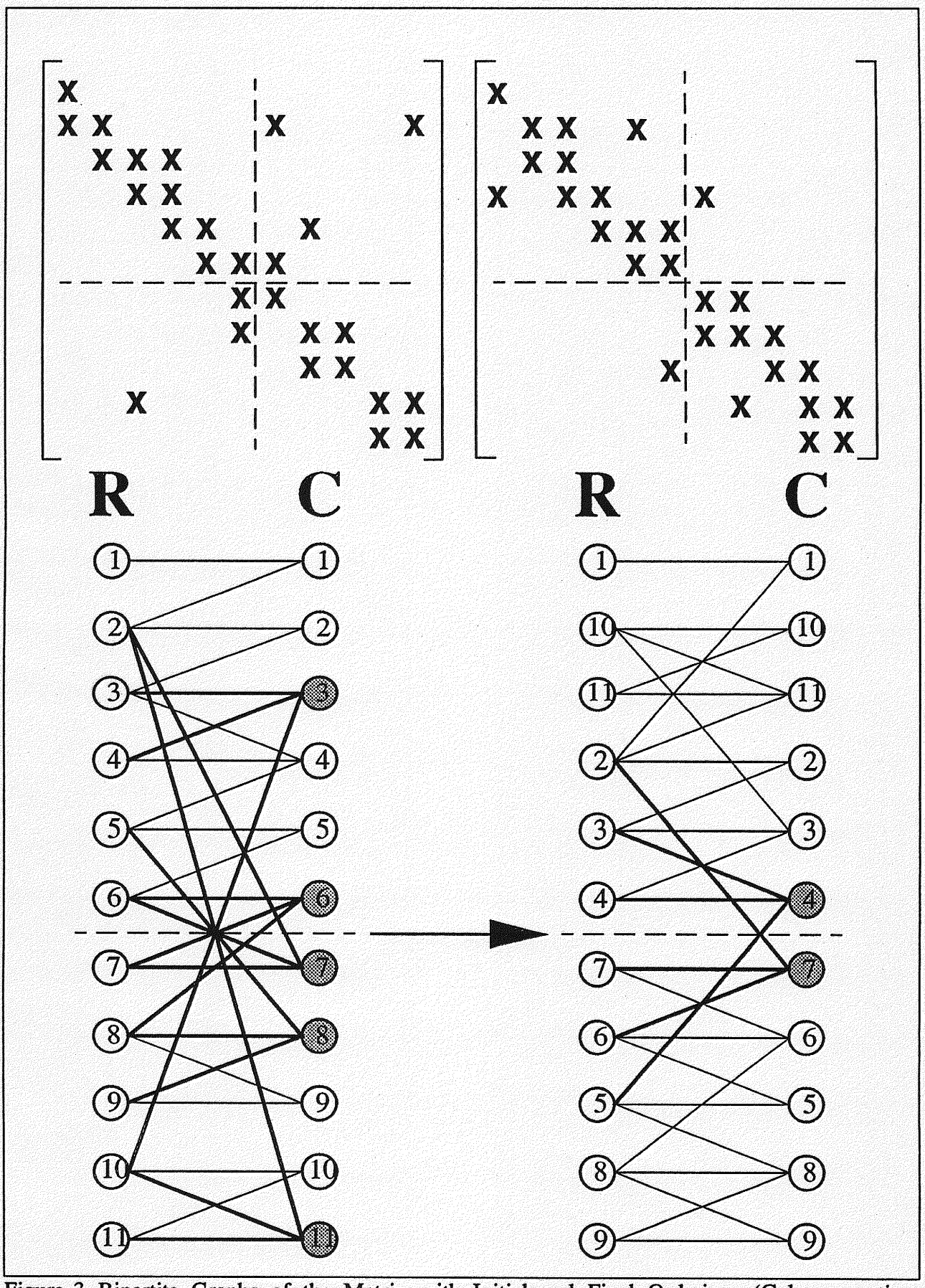

Figure 3 Bipartite Graphs of the Matrix with Initial and Final Orderings (Column vertices corresponding to cut nets are shaded)

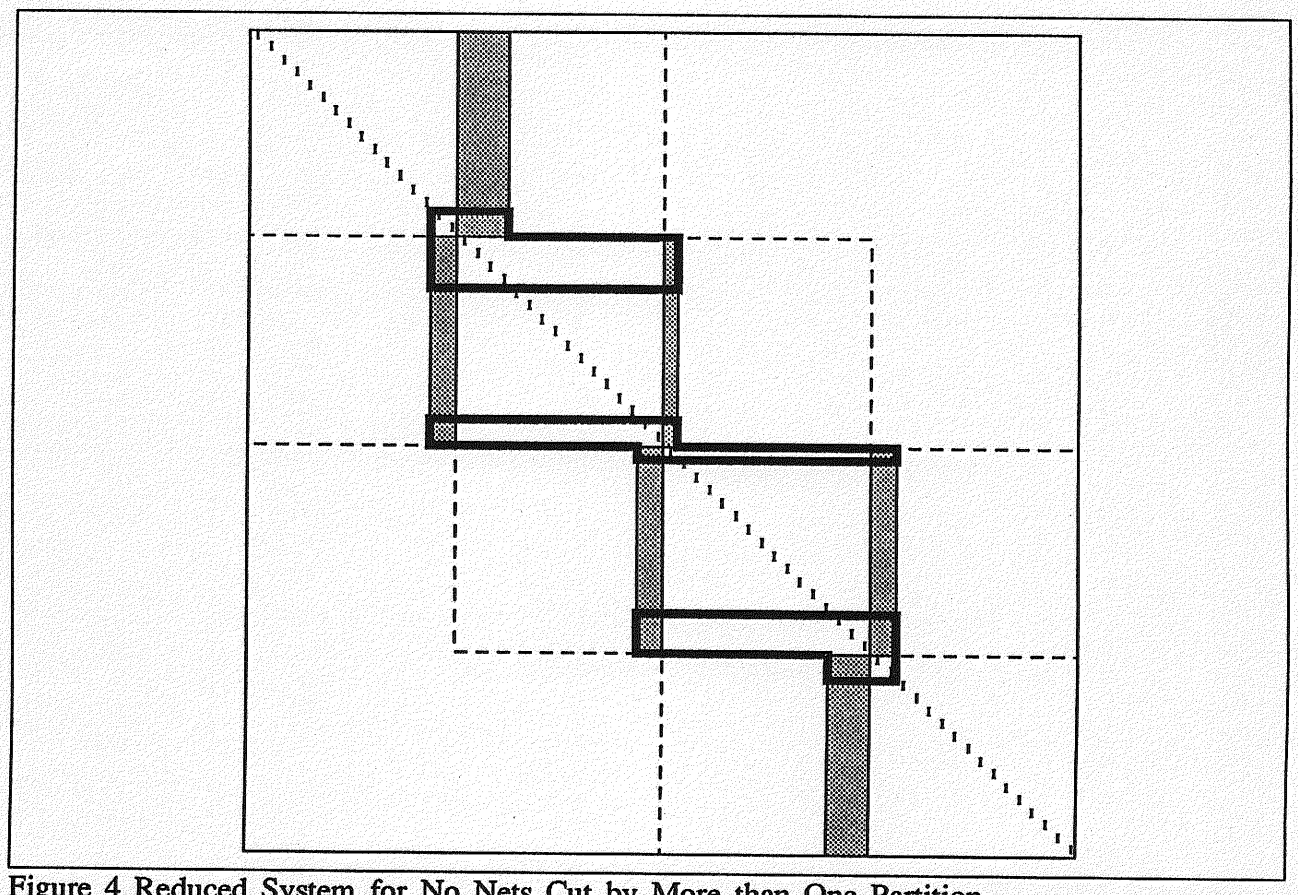

Figure 4 Reduced System for No Nets Cut by More than One Partition

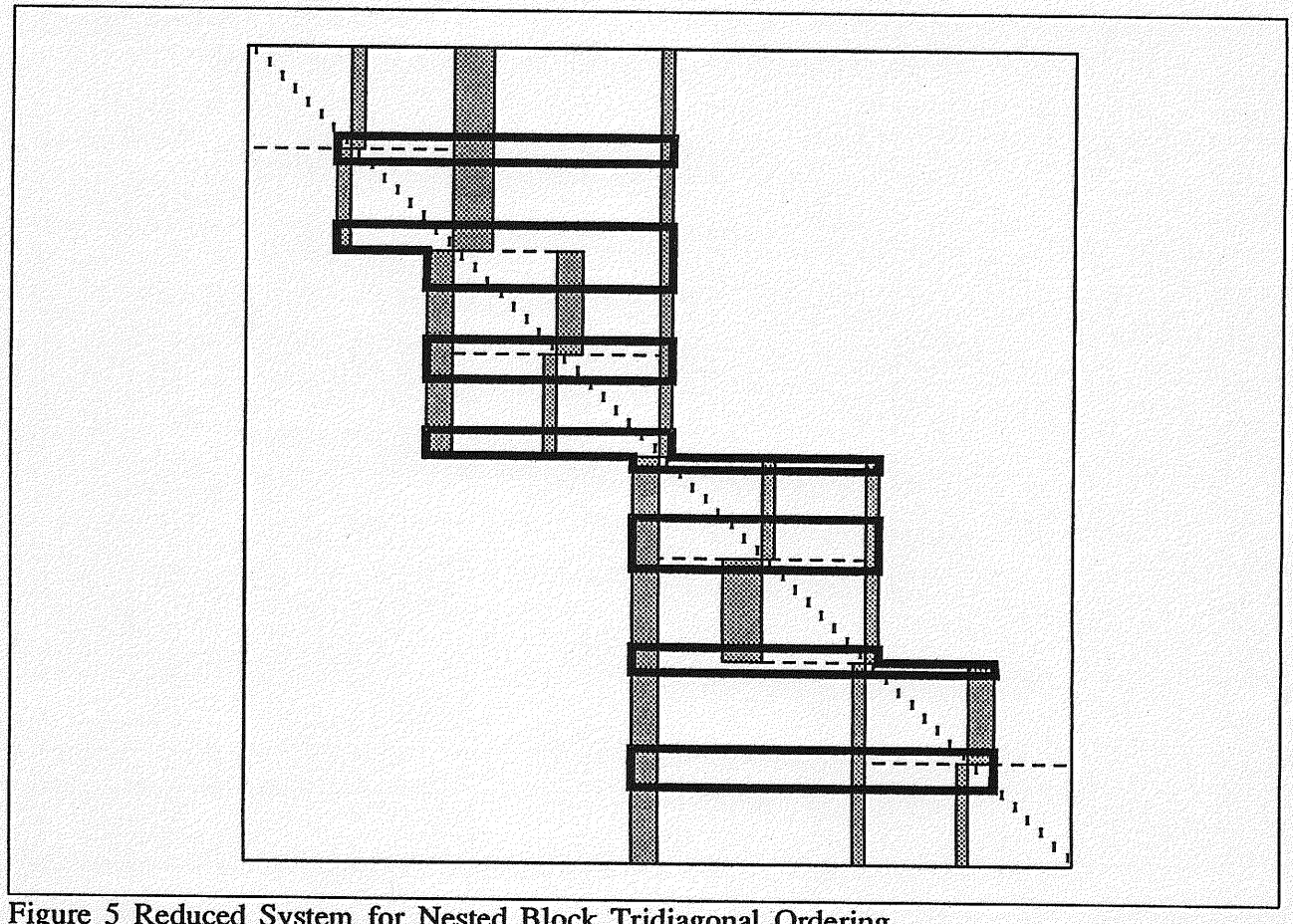

Figure 5 Reduced System for Nested Block Tridiagonal Ordering