MPE — Volume 1, pp. 41-57
Reprints available directly from the publisher
Photocopying permitted by license only

A NESTED DECOMPOSITION ALGORITHM FOR PARALLEL COMPUTATIONS OF VERY LARGE SPARSE SYSTEMS*

D. D. ŠILJAK and A. I. ZEČEVIĆ

School of Engineering, Santa Clara University, Santa Clara CA 95053 USA

(Received 15 June 1994)

In this paper we present a generalization of the balanced border block diagonal (BBD) decomposition algorithm, which was developed for the parallel computation of sparse systems of linear equations. The efficiency of the new procedure is substantially higher, and it extends the applicability of the BBD decomposition to extremely large problems. Examples of the decomposition are provided for matrices as large as 250,000 × 250,000, and its performance is compared to other sparse decompositions. Applications to the parallel solution of sparse systems are discussed for a variety of engineering problems.

AMS No.: 65F, 65W
KEYWORDS: Linear equations, sparsity, balanced decompositions, parallel computations

1. INTRODUCTION

Large systems of sparse linear equations arise in a wide variety of engineering problems, ranging from electric circuit simulation to solving partial differential equations by finite element and finite difference methods. Linear equations are also a central issue in many nonlinear problems, such as those that require the Newton iterative method or numerical integration of nonlinear differential equations. The need for solving these equations efficiently is particularly apparent for problems that have reached a level of complexity which cannot be handled adequately by a single processor. In such cases parallel computation becomes a necessity, and this paper describes a decomposition algorithm that enables us to perform the parallelization very effectively.

Two classes of engineering problems are of fundamental interest from the standpoint of parallel computing. One type are extremely large problems such as VLSI circuit simulation or finite element solutions to three dimensional partial differential equations, where it is necessary to solve anywhere between 20,000 and several million equations. Moderately sized problems (involving 5,000–10,000 equations) are also of interest if they need to be solved repeatedly, as is the case in the transient stability analysis of electric

*The authors would like to thank Professor Fernando Alvardo, University of Wisconsin, for providing some of the test matrices. The research reported herein was supported by the National Science Foundation under Grant ECS-9114872.

power systems or in iterative procedures for solving nonlinear equations. For both classes, parallel computing is really the only effective way of producing a solution.

In the following we will present a generalized and improved version of the decomposition algorithm described in ([1]), which removes any restrictions on the matrix size. The main objective of this decomposition is to permute the matrix into a balanced BBD form like the one shown in Fig. 1. Such a structure is characterized by a *balance* in size between the diagonal blocks and the border, as well as a BBD structure *within* the border block. These features allow for the implementation of several levels of parallelism in the LU factorization, and result in good load balancing and low interprocessor communications. Significant computational speedups can be achieved based on this structure, particularly on massively parallel SIMD (Single Instruction Multiple Data) architectures with several thousand processors.

2. THE ORIGINAL BALANCED BBD DECOMPOSITION ALGORITHM

The bordered block diagonal form has long been recognized as a desirable matrix structure in parallel solutions of linear equations (*e.g.*, [2]). The principle advantage of this approach lies in the fact that the BBD structure can be utilized to implement the parallelism *off-line*, thus avoiding the inevitable overhead associated with other methods (such as those based on elimination trees, for example). In addition, a balance in the sizes of the diagonal blocks secures an optimal distribution of the work load across the set of processors, as well as low interprocessor communications.

The main obstacle in applying this type of parallelization has been the decomposition itself. Numerous algorithms have been developed, including node clustering ([5]), graph dissection ([4]), and diakoptics ([5], [6]). Nevertheless, the success of these methods has been mainly limited to matrices with a very regular structure. In the general case, these algorithms did not perform as well; in applications to electric circuits they actually

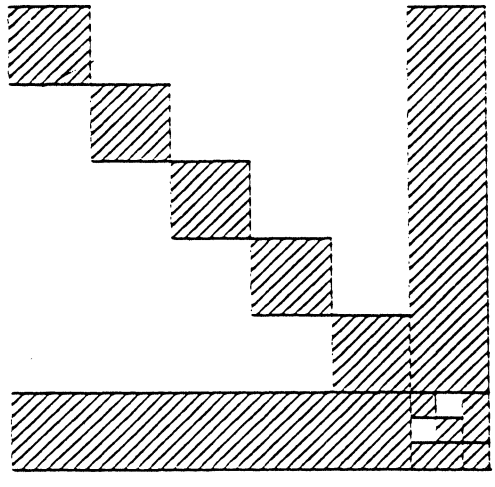

Figure 1 A balanced bordered block diagonal structure.

produced rather disappointing results ([7]; [8]). Part of the problem was that standard BBD decomposition algorithms attempted to identify a *minimal* border which induces a block diagonal structure in the rest of the matrix. Such a strategy frequently resulted in only a few diagonal blocks of varying sizes, which is not a particularly desirable structure for parallel processing. The balanced BBD decomposition algorithm described in ([1]) alleviated these difficulties by allowing a larger border with an internal BBD structure (such as the one shown in Fig. 1.). It was found that any type of sparse matrix can be permuted into this form, regardless of its structural regularity.

The balanced BBD decomposition consists of three stages. In the first stage we initially select a maximal degree Dm and all vertices with degree $\geq$ Dm are placed into the border (this procedure is referred to as Dm decomposition). We then select a maximal allowable block size Nmax, and additional vertices are moved into the border to achieve this size (this procedure is referred to as Nmax decomposition). The Nmax decomposition removes vertices indiscriminately, and typically results in a structure like the one shown in Fig. 2, where the border is much larger than any of the diagonal blocks.

In the second stage of the decomposition, the border is recursively decreased until a balance in size is achieved between the border and the largest diagonal block. In each step of this procedure, the next vertex to be removed from the border is the one whose reconnection results in a minimal increase in the size of the diagonal blocks. Such a "greedy" strategy can be implemented very effectively by associating data structures like the ones shown in Fig. 2. with every border vertex.

The example in Fig. 3 has the following interpretation:

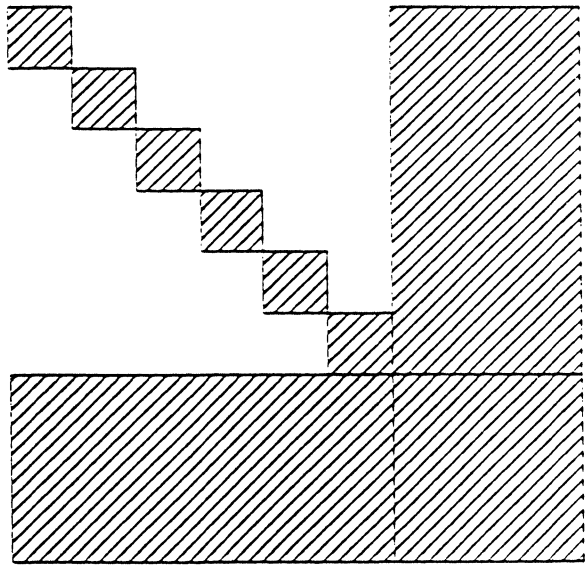

Figure 2 The decomposition after stage 1.

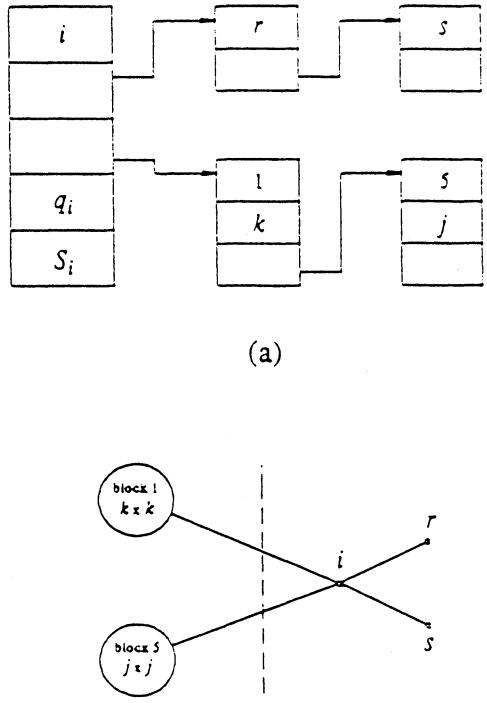

Figure 3 a) Data structures corresponding to border vertex x_i b) Schematic interpretation of (a).

a) *Lower list.* Vertex x_i is connected to blocks 1 and 5, of size k and j, respectively. Reconnecting x_i requires merging blocks 1 and 5, which produces a new block of size $S_i = k + j$.

b) *Upper list.* Vertex x_i is also incident to border vertices r and s, so $q_i = 2$. Reconnecting x_i to block 1 would add this block to the lower list of vertices x_r and x_s (if it is not there already).

In any step of the second stage, the next border vertex to be reconnected will be the one with the minimal S_i. If there is a tie between several vertices, the one with the smallest q_i is chosen.

The final result of the second stage is a BBD structure in which the largest diagonal block and the border have approximately the same size. However, at this point other diagonal blocks may be much smaller, so a balancing procedure is additionally executed, in order to achieve the structure shown in Fig. 4.

The third and final stage of the decomposition amounts to reordering the border vertices in order to secure a BBD structure *within* the border block. In this procedure, symbolic fill-in is added to the border block to include the effects of LU factorization. The border

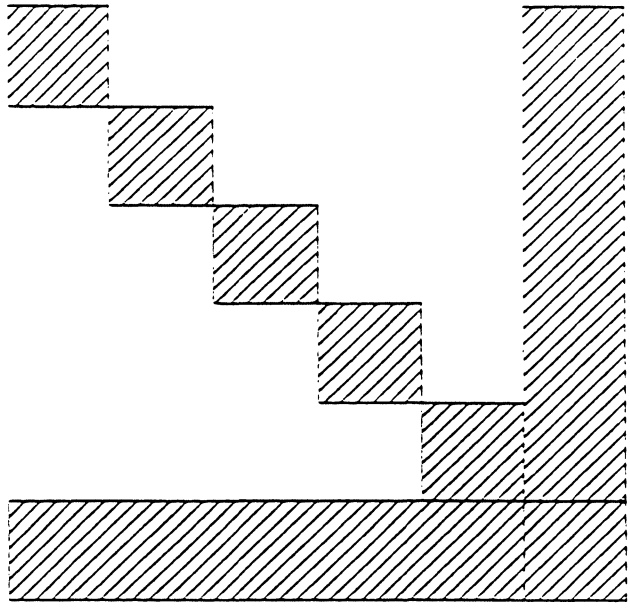

Figure 4 The decomposition after stage 2.

block is then decomposed following the first and second stages, and the resulting matrix has the form indicated in Fig. 1. A flowchart for the overall decomposition is shown in Fig. 5; additional details can be found in ([9]).

3. THE IMPROVED BALANCED BBD DECOMPOSITION

The decomposition algorithm described in the previous section is very effective for sparse matrices of size up to 5,000 × 5,000. Particularly good results were obtained in applications to large electric power systems such as the one shown in Fig. 13, which represents a model of the entire U.S. power network. However, with increasing system size and more nonzero elements, the second and third stages of the decomposition were found to be very time consuming. This was primarily due to the large amount of symbolic fill-in which required an unacceptable amount of memory for the data structures as well as long list searches. We now present a generalized and improved balanced BBD algorithm which can be effectively applied to matrices as large as 250,000 × 250,000 (such an example is shown in Fig. 16).

The strategy of the improved version significantly differs from the earlier one in the second and third stages of the algorithm. The idea of this new decomposition is to utilize a *nested* procedure in which the reconnection would continue until exactly two large blocks remain. This algorithm is initially executed on the entire matrix and is subsequently repeated on individual blocks at each level of the decomposition.

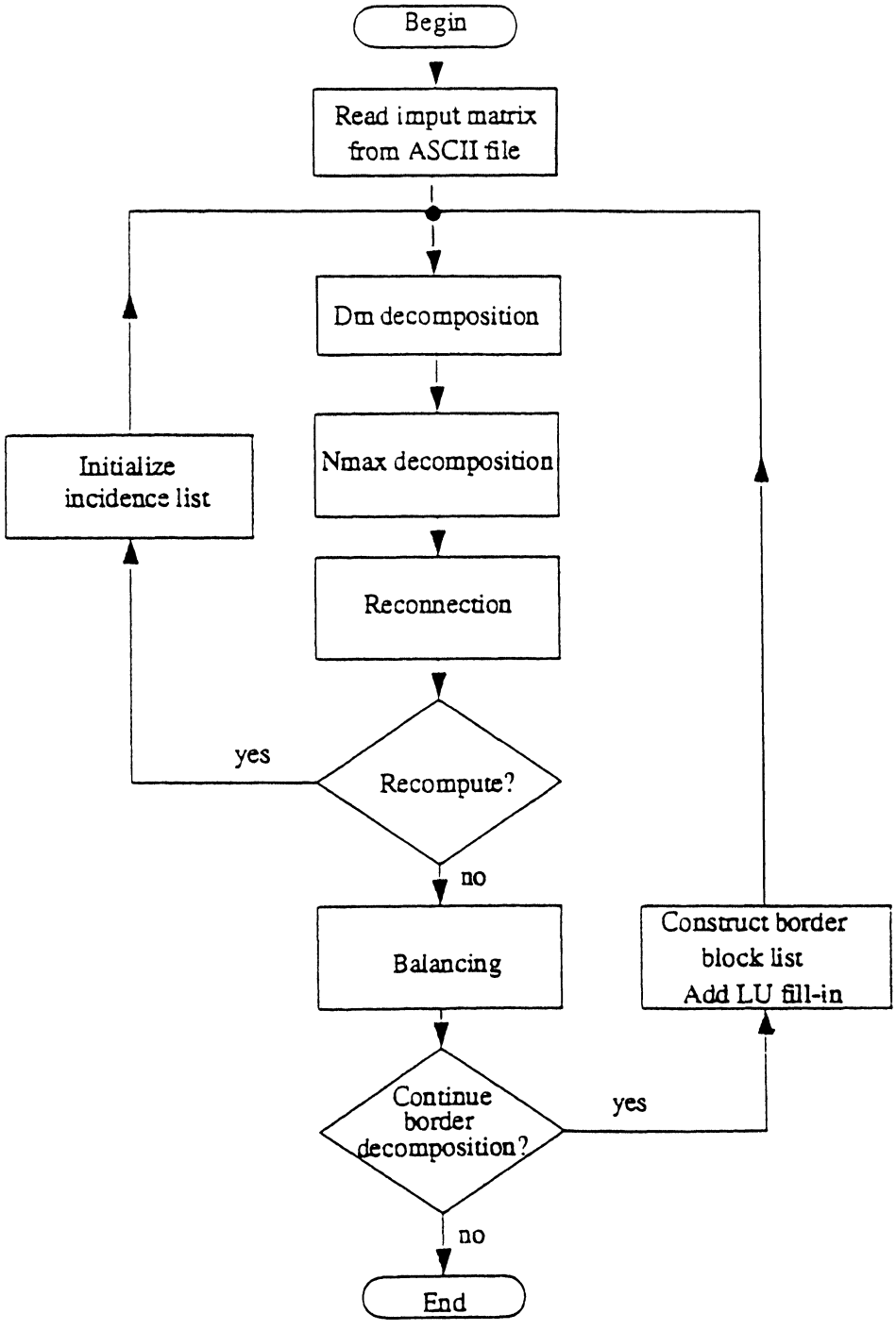

Figure 5 A flowchart for the algorithm.

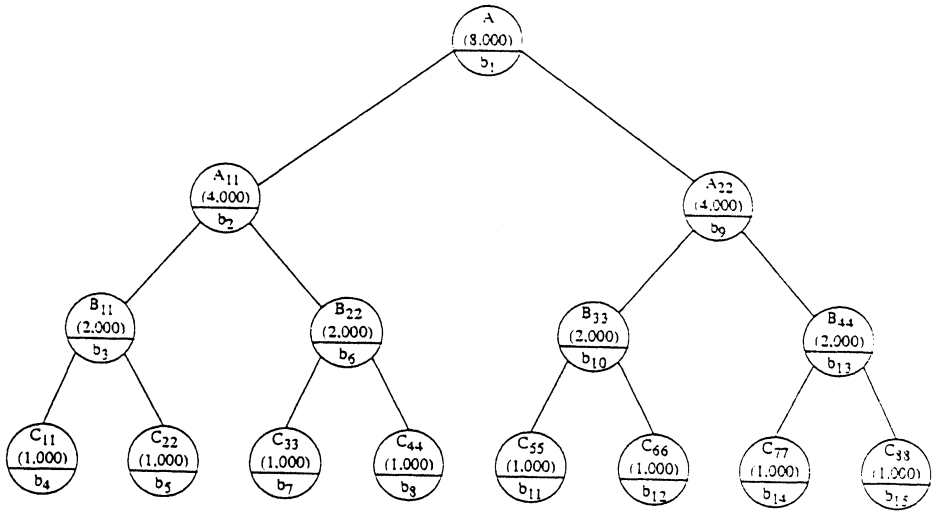

Figure 6 The decomposition graph after three levels of decomposition.

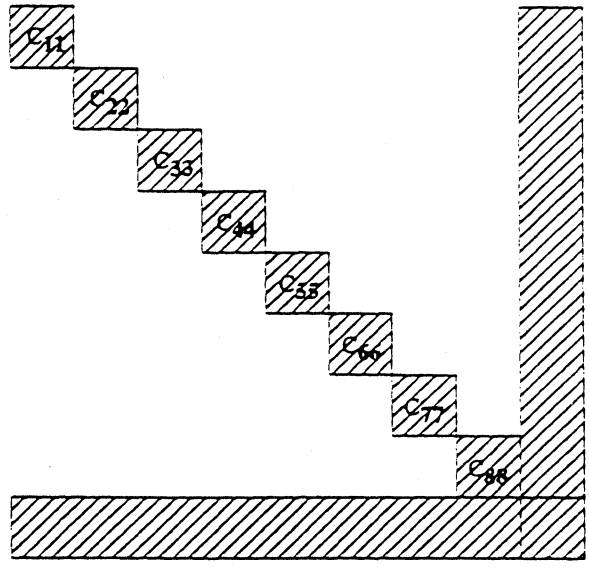

Figure 7 The decomposed matrix after local borders are moved.

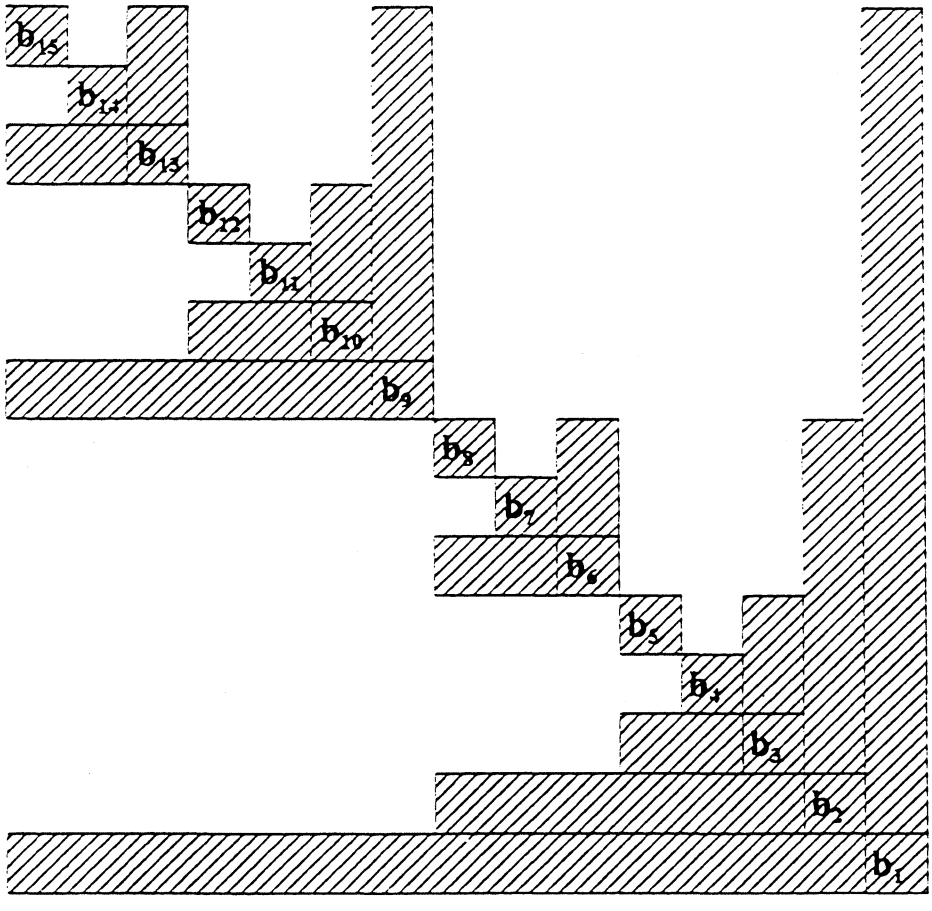

Figure 8 The internal structure of the border block after 3 levels of decomposition.

The overall procedure is controlled by a decomposition graph which contains the necessary information about all border and block sizes. An illustration of this graph after three levels of decomposition is shown in Fig. 6, for an 8,000 × 8,000 matrix A. In Fig. 6 A_{ii}, B_{ii}, and C_{ii} represent diagonal blocks associated with different decomposition levels, and b_j represents corresponding local borders. By removing the local borders and placing them into the *overall* border, we can now obtain a matrix with the generic structure shown in Fig. 7.

The sequence in which individual local borders are removed is defined by a Depth First Search of the decomposition graph. In the graph of Fig. 6, the borders have been enumerated in the order in which they were visited; the corresponding internal structure of the border block is shown in Fig. 8. It should be observed that the border block is now even more structured than in Fig. 1, which results in better sparsity preservation

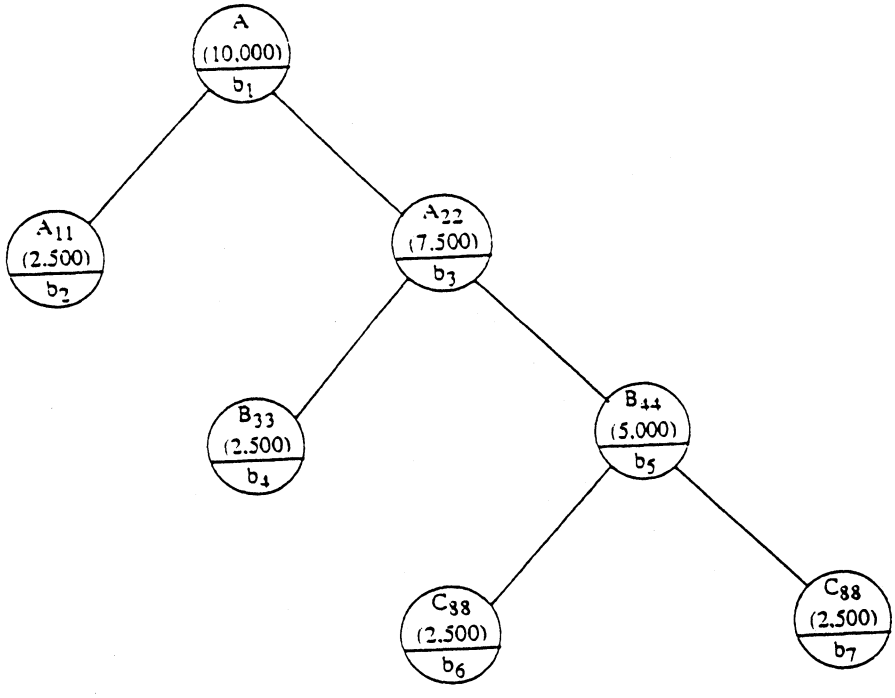

Figure 9 A possible scenario for the decomposed graph.

(particularly for very large matrices). The actual number of decomposition levels in the graph is determined by the sizes of the diagonal blocks and the corresponding borders. In general, different branches of the graph may have different levels, as illustrated in Fig. 9. The decomposition procedure checks the sizes after each level, and continues until the border and maximal block size become approximately equal. It should also be observed that significant portions of the nested BBD decomposition can be performed in parallel. Specifically, each block in a given decomposition level can be decomposed by a different processor. The number of such blocks could roughly double in each level, so the overall effect of parallelization will increase with a larger number of processors.

4. APPLICATIONS OF THE IMPROVED BBD DECOMPOSITION: PARALLEL LU FACTORIZATION OF SPARSE MATRICES

Large systems of sparse linear equations require special solution techniques, which are typically based on LU factorization (*e.g.*, [10]). In all problems of this nature, a key objective is to order the rows and columns in such a way that the L and U factors retain as much of the original sparsity as possible. The critical importance of ordering is illustrated by the following example.

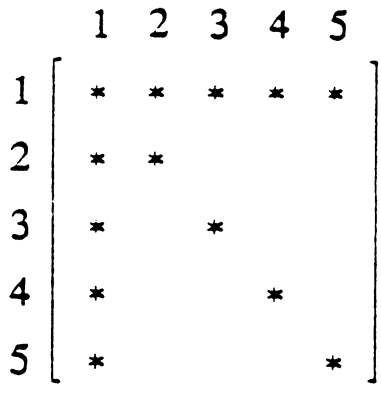

Figure 10 Matrix A.

EXAMPLE Consider the matrix shown in Fig. 10, in which* denotes the nonzero elements. The L and U factors for this matrix are shown in Fig. 11a, where ⊙ denotes additional nonzero elements generated by the factorization; these elements are generally referred to as *fill-in* elements. Clearly, in this case both L and U are dense matrices. On the other hand, permuting matrix A in the manner shown in Fig. 11b will completely eliminate the fill-in.

Fill-in minimization is a fundamental requirement for efficient computation with sparse matrices—it can decrease the number of floating point operations needed for factorization from $O(n^3)$ to as low as $O(n)$ (*e.g*, [10]). Numerous ordering algorithms have been developed for reducing fill-in in the factorization process. Undoubtedly the most general

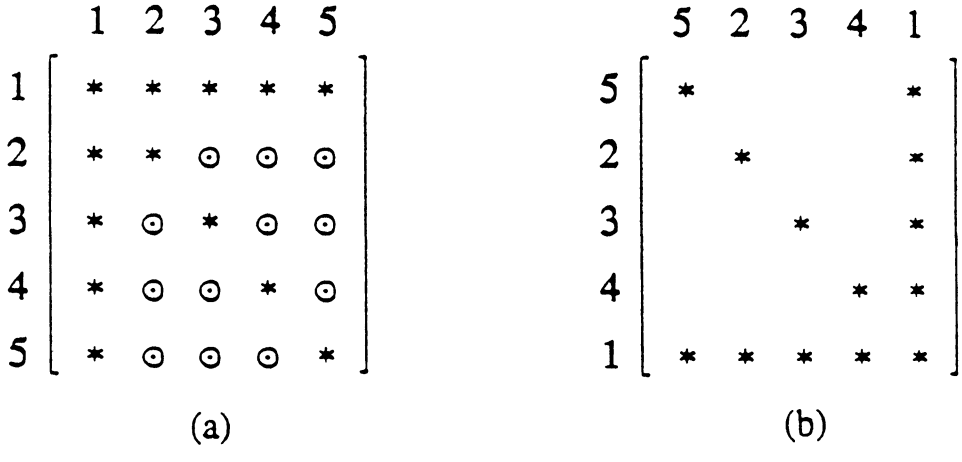

Figure 11 a) L and U factors of A without reordering. b) L and U factors with reordering.

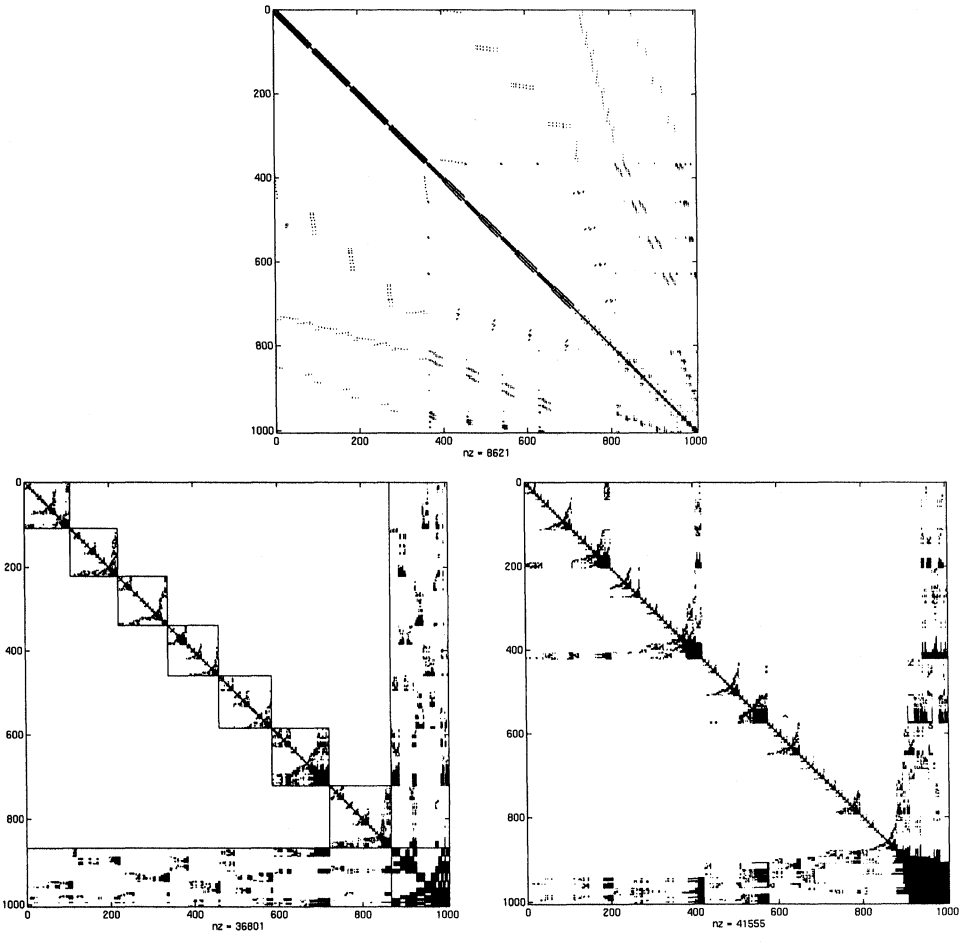

Figure 12 a) A matrix from structural analysis (1,005 × 1,005). b) The LU factors using a balanced BBD decomposition. c) The LU factors using minimal degree.

and effective method to date has been the *minimal degree algorithm* [11]; its relative simplicity and applicability to a wide range of matrix structures have made it a standard part of any sparse matrix software package.

Despite the low level of fill-in, in parallel computations minimal degree and similar orderings are by no means optimal. The main reason for this is a lack of structure in the reordered L and U factors, due to element scattering throughout the matrix. As a result, an effective off-line parallelization is not possible, and techniques such as dynamic load balancing must be applied [2]. This typically results in significant overhead and a suboptimal load distribution across the processors.

In contrast, the balanced BBD decomposition automatically produces a highly parallelizable structure. It is well known that in BBD matrices each diagonal block and corresponding border segment can be independently factorized by a different processor, and only the factorization of the border block requires interprocessor communication.

Table I A fill-in comparison

Matrix Size	*Number of Nonzeros*	*Nonzeros in LU Factors*		
		Min. Deg. (N_m)	*BBD Dec.* (N_b)	N_b/N_m
1,005	8,621	41,555	36,801	0.88
1,993	7,443	13,439	14,259	1.06
2,003	83,883	588,887	568,545	0.96
3,466	23,896	186,348	180,680	0.97
8,738	591,904	6,745,812	7,730,880	1.14

In addition, the internal BBD structure within the border block (like the one shown in Fig. 8 allows for a second level of parallelism.

In order to minimize the fill-in associated with balanced BBD decompositions, the individual diagonal blocks are ordered locally using the minimal degree algorithm. In this way we combine the structural advantages of BBD decompositions with the fill-in optimality of the minimal degree ordering. In Fig. 12 we present an example from structural analysis which demonstrates that the BBD decomposition not only results in a superior structure but also can produce even less fill-in than the minimal degree (in this case, 36,081 nonzeros compared to 41,555 nonzeros).

Numerical experiments on a variety of large sparse matrices have shown that the difference in fill-in between the BBD decomposition and minimal degree is not significant. As demonstrated in Table I, typically it does not exceed 15%. In light of the superior parallelizability of the BBD structure, this is more than acceptable.

It is also interesting to point out that in a number of cases the BBD decomposition was found to be considerably faster than the minimal degree ordering. This is particularly important for very large matrices, where the ordering itself can require an excessive amount of time. For some such matrices the BBD decomposition was performed over four times faster than the minimal degree ordering in Matlab 4.0 (function **symmmd( )**). A summary of experimental results is given in Table II.

The versatility of the balanced BBD decomposition algorithm is illustrated in Figs. 13–16. These examples, taken from very diverse engineering applications, indicate that our decomposition can indeed be successfully applied to sparse matrices of any structure and size.

Table II A comparison of execution times for BBD decompositions and Symmetric minimal degree ordering on HP Apollo 700 workstations

Matrix Size	*Number of Nonzeros*	*Execution Time (seconds)*		
		Min. Deg. (t_m)	*BBD Dec.* (t_b)	t_m/t_b
2,003	83,883	14.47	1.86	7.79
8,738	591,904	101.93	91.79	1.11
28,924	2,043,492	1,118.6	287.15	3.90
35,588	1,181,416	434.02	105.54	4.11
44,609	2,014,701	1,031.9	241.92	4.26

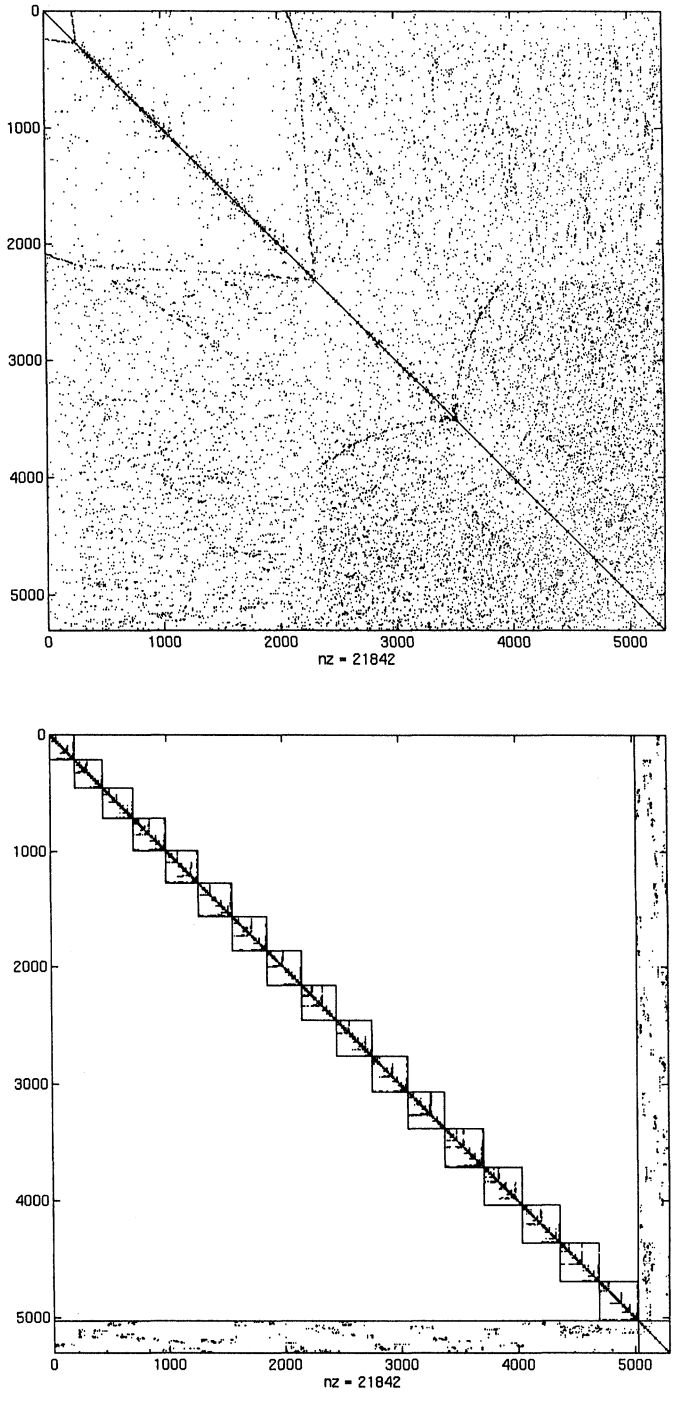

Figure 13 a) A model of the entire U.S. electric power network (5,300 × 5,300). b) The matrix after balanced BBD decomposition.

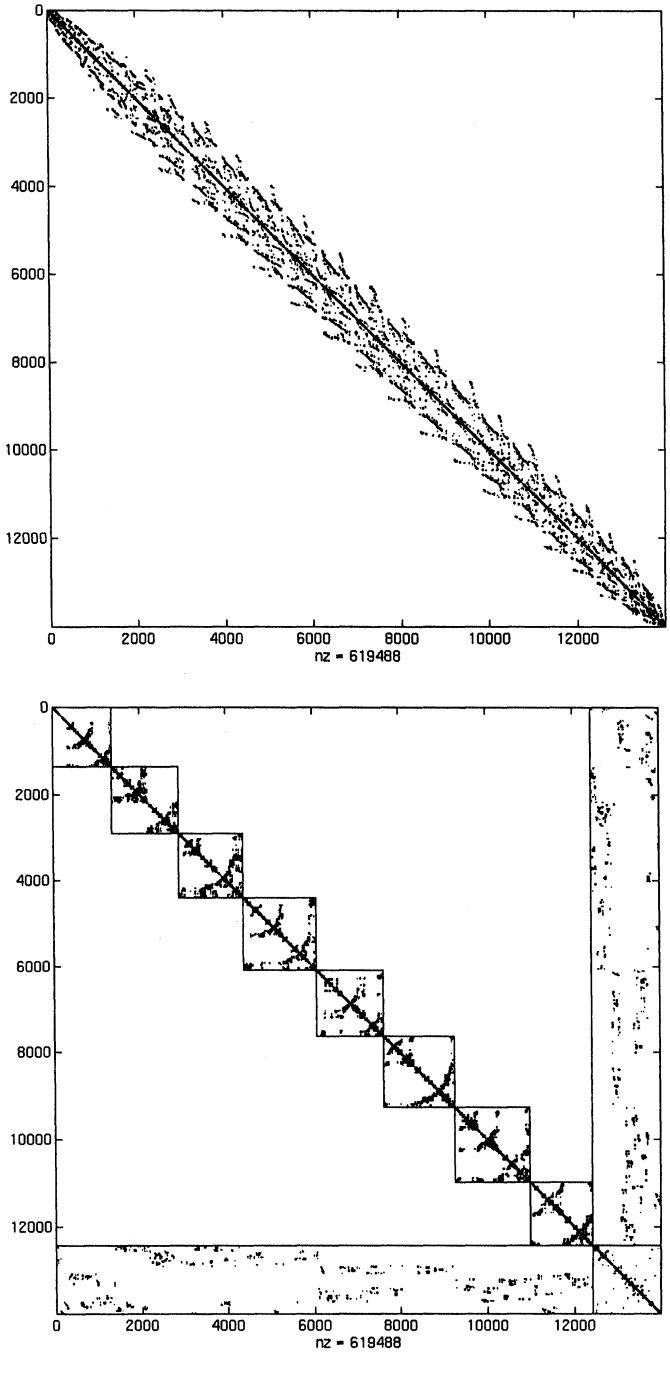

Figure 14 a) A buckling model for a Boeing 767 rear bulkhead (13,992 × 13,992). b) The matrix after balanced BBD decomposition.

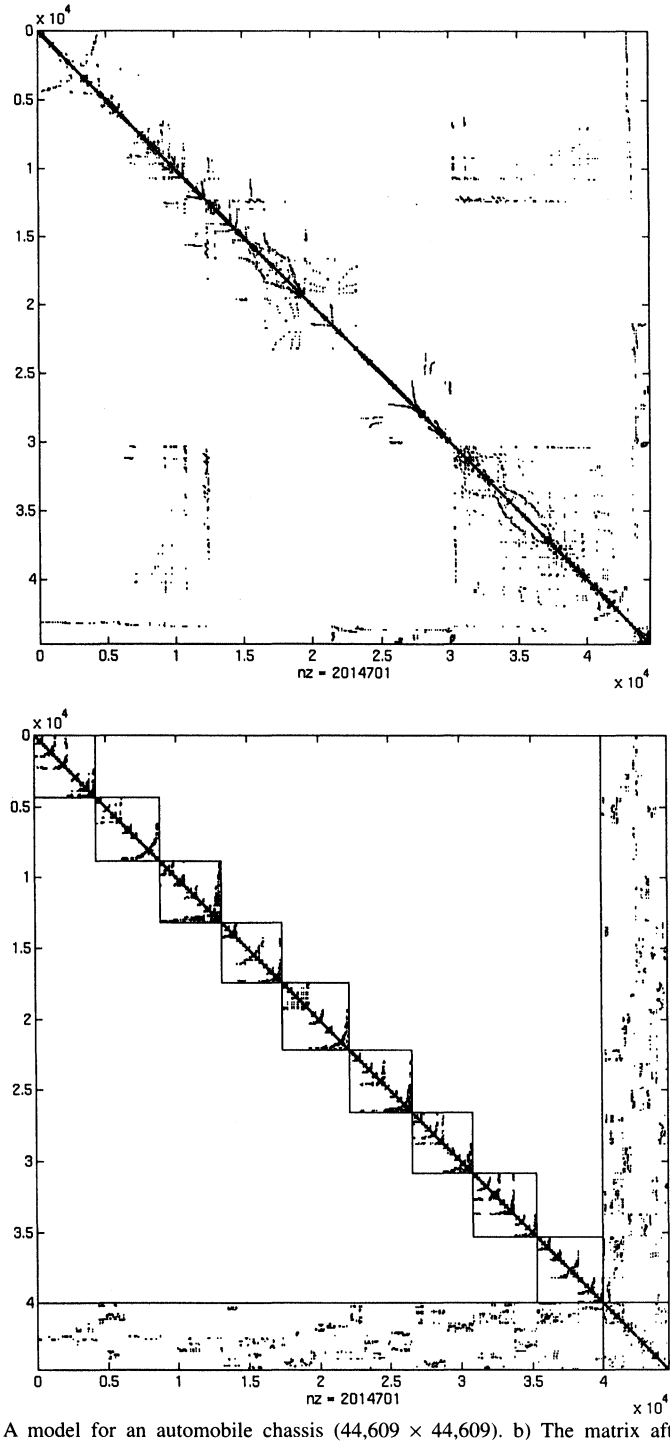

Figure 15 a) A model for an automobile chassis (44,609 × 44,609). b) The matrix after balanced BBD decomposition.

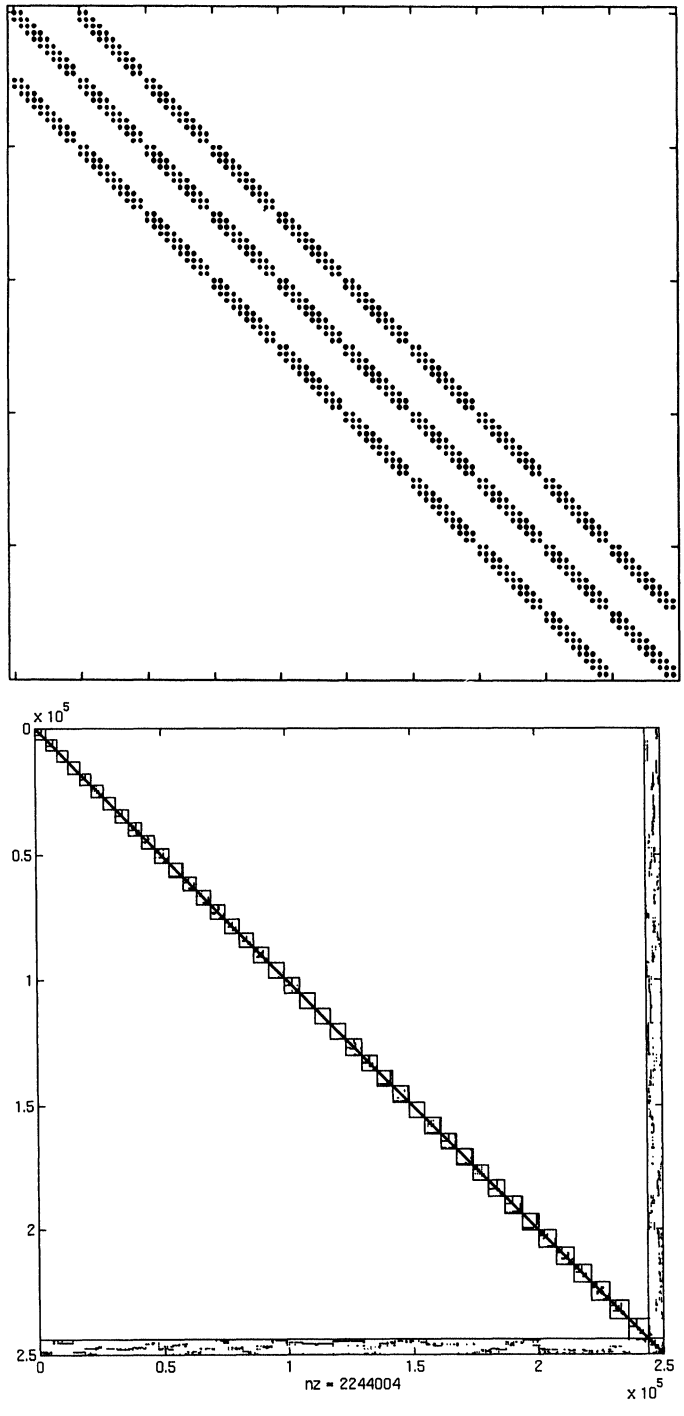

Figure 16 a) Segment of the matrix resulting from a nine-point finite element approximation on a square grid with $n = 500$ (the overall matrix is 250,000 × 250,000). b) The matrix after balanced BBD decomposition.

References

1. A. I. Zečević and D. D. Šiljak, Balanced Decompositions for Multilevel Parallel Processing, *IEEE Trans. Circuits and Systems*, **41**, 220–233 (1994).
2. M. T. Heath, E. Ng, and B. W. Peyton, Parallel Algorithms for Sparse Linear Systems, in K. A. Gallivan *et al.* (Eds.), SIAM, Philadelphia, pp. 83–124, (1990).
3. A. Sangiovanni-Vincentelli, L. K. Chen, and L. O. Chua, An Efficient Heuristic Cluster Algorithm for Tearing Large Scale Networks, *IEEE Trans. Circuits and Systems*, vol. CAS-**24**, 709–717 (1977).
4. A. George, and J. W. H. Liu, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice Hall, Englewood Cliffs, N. J. (1981).
5. G. Kron, *Diakoptics*, McDonald, London, (1963).
6. H. H. Happ, *Diakoptics and Networks*, Academic Press, New York (1971).
7. F. L. Alvarado, Sparsity in Diakoptic Algorithms, *IEEE Trans. Power App. and Syst.*, **96**, 1450–1459 (1977).
8. A. M. Erisman, Sparse Matrix Problems in Electric Power System Analysis, in *Sparse Matrices and Their Uses* (I. S, Duff, Ed.), Academic, New York pp. 31–56 (1981).
9. A. I. Zečević, New Decomposition Methods for Parallel Computations of Large Systems, Ph. D. Thesis, Santa Clara University (1993).
10. I. S. Duff, A. M. Erisman, and J. K. Reid, *Direct Methods for Sparse Matrices*, Oxford: Clarendon Press (1986).
11. W. F. Tinney and J. Walker, Direct Solutions of Sparse Equations by Optimally Ordered Triangular Factorization, *Proc. IEEE*, **55**, 1801–1809 (1967).