N74-14588

REVIEW OF NASTRAN DEVELOPMENT RELATIVE TO EFFICIENCY OF EXECUTION

By Caleb W. McCormick
Director of Engineering Analysis
The MacNeal-Schwendler Corporation
Los Angeles, California

SUMMARY

This paper reviews the development of NASTRAN relative to the efficiency of execution, with particular emphasis on those items which have changed significantly since the original release of NASTRAN. Features discussed include main and secondary storage utilization, matrix packing, matrix assembly, matrix multiplication, matrix decomposition and equation solution. Also a brief look into the future discusses the questions of faster arithmetic units and more effective storage utilization. In some cases the improvements in NASTRAN efficiency have resulted from taking advantage of hardware developments, while in other cases increased efficiency has resulted from improvements in the state of the art for data processing or matrix operations. The modular design of NASTRAN has made it possible to improve the efficiency in many parts of NASTRAN without changing the basic design of the program.

INTRODUCTION

The main goal in the original design specifications for NASTRAN was the solution of large problems in both statics and dynamics. Although efficiency has always been an important consideration, the primary emphasis in the beginning was on the wide range of problem types of large size. However, it was recognized that in order to solve large problems, it would be mandatory to take advantage of sparse matrix techniques. Consequently, all of the original matrix operations were designed to utilize sparse matrix techniques. Since the original release of NASTRAN, a number of improvements have been made in the efficiency of the matrix routines by taking advantage of hardware developments and improvements in the state of the art for data processing and matrix operations.

Hardware and software limitations required that the early versions of NASTRAN use only sequential secondary storage. The current versions of NASTRAN use direct access devices for secondary storage. Level 16 will include the use of random access coding in the matrix assembly and equation solution operations. The use of multiple types of secondary storage devices will be an important development for future releases of NASTRAN.

Most of the original code for NASTRAN was written in FØRTRAN IV in order to reduce both the development costs and maintenance costs. Machine language was used only in those placed where it was necessary to interface with the resident operating systems. However, efforts were made to improve the efficiency of the compiled code by care in the use of FØRTRAN. For example, in the case of nested DØ loops it was found that, for some compilers, the speed of the inner loops for multiply-add operations could be improved by a factor of two by simply writing the inner loop as a separate subroutine. Substantial improvements have been made in Level 15 and Level 16 through the use of machine language in many of the more important matrix operations. Even so, except for the transfer of information between main storage and secondary storage and the matrix packing routines, the use of machine language has been restricted to the inner loops of the matrix operations.

The original emphasis on the solution of large problems has caused NASTRAN to be relatively inefficient for small- and medium-size problems. Many of the Level 16 improvements will substantially improve the efficiency of NASTRAN for the smaller problems. The main improvements in this area are associated with the more effective use of main storage and additional options for the matrix packing routines.

MAIN STORAGE UTILIZATION

The original design of NASTRAN recognized the importance of preserving as much as possible of the main storage for matrix operations. The overlay structure was carefully designed to minimize the amount of code that had to be resident in main storage, particularly during the operation of important matrix routines. Also, in order to preserve the maximum amount of main storage for each matrix operation, all results were transferred to secondary storage prior to the start of a new major operation. This transfer of information to and from secondary storage placed a heavy burden on the matrix packing routines and the associated read and write routines. These routines have contributed heavily to the relative inefficiency of NASTRAN.

The original design of the overlay structure for NASTRAN is still in use and there appears to be no need for major changes in this area. The inefficiencies resulting from the transfer of information between main and secondary storage has been a source of some concern and has received extensive revision in both Level 15 and Level 16.

In order to reduce the requirement for transfers between main storage and secondary storage, Level 16 will provide an option to use a portion of main storage for the retention of data which would otherwise be transferred to secondary storage. This option is under the control of the individual functional modules and has been implemented by allowing the use of multiple buffers in main memory by the I/Ø routines. This option will be particularly effective when relatively small amounts of frequently used information can be retained in main memory and thereby avoid excessive transfers from secondary storage. A strong candidate for using this option is the numerical integration in transient response problems, where currently the triangular factors of the dynamic matrix must be read from secondary storage at each time step. If the problem is not too large, it may be convenient to allocate sufficient main memory to hold the triangular factors in main storage. This option should also be useful for nonlinear problems where the problem sizes tend to be small and repetitive operations form an important part of the total solution time.

SECONDARY STORAGE UTILIZATION

All of the transfers of information between secondary storage and main storage in the early releases of NASTRAN used sequential access methods. Much of the original code was also written in FØRTRAN, which further contributed to the inefficiencies. The original dynamic use of files as supervised by the NASTRAN Executive System has stood the test of time and remains today essentially as in the original design.

Sequential procedures are still used for all write operations. However, read operations may be performed either by the use of sequential procedures or random procedures. Two important uses of the random access procedures in Level 16 are for matrix assembly and the back substitution part of the equation solution routines.

MATRIX PACKING

Efficient operation with large sparse matrices requires an effective packing scheme in order to minimize main memory requirements and the time required to transfer the nonzero elements from secondary storage devices to the working space in main memory. The current matrix packing logic is similar to that used in the original design of NASTRAN. However, substantial improvements in the efficiency were made in Level 15, including the use of machine language on the IBM versions. Level 16 will include further improvements in efficiency, including the use of assembly language on the Univac version. The Level 16 matrix packing routines will use machine language on all machines and will be 1-1/2 to 2 times faster than Level 15.

The matrices in NASTRAN are stored by columns, and each column constitutes a logical record. The first nonzero term in the column is described by an integer indicating its row position and the floating point number describing its value. If the following term is also nonzero, only its value is stored, and in general the position of only the first term in the series of nonzero terms is stored. In order to improve the efficiency when working with strings of nonzero terms, the number of nonzero terms in the string is stored along with the row number of the first nonzero term in the string. An option is also provided to include the row number of the last nonzero term in the string along with the number of nonzero terms in the string. This option makes it possible to perform the backward substitution operation for equation solution in a more efficient manner.

An important addition to the packing routines in Level 16 is a new nontransmit option. In the case of a read operation, each call for this option results in the return of the row number of the first nonzero term in the next string and the number of terms in the string. In the case of a write operation, each call returns the location of the next available space in the NASTRAN I/Ø buffer and the number of spaces remaining in the current buffer. The using routines can then operate directly in the I/Ø buffers, and only the time associated with the initial call for each string is required for the packing operation. The time to access a term or store a term can be absorbed in the using routine, as this operation is required, even when operating outside the buffer.

For strings of reasonable length, the time per term for the nontransmit operation is very small and may well be ten or twenty times less than the transmitting pack options. This option is particularly effective when each term in the buffer is used for a single operation, such as for direct transient response or eigenvalue extraction using the inverse power method, where the running times are dominated by equation solutions with single right hand sides. If the transmitting pack options are used in these cases, the running time is dominated by the packing times, which in turn may be several times the associated arithmetic times. If the nontransmit option is used, the inner loops will run at arithmetic speed, and the speed of operation with single right hand sides will be several times faster.

MATRIX ASSEMBLY

The comparison of the matrix assembly times for various assembly procedures is indicated in Figure 1. The initial straight portion of the solid line indicates a linear growth of matrix assembly time with problem size when the complete stiffness matrix can be held in main memory. The curved portion of the solid line indicates a rapid growth in matrix assembly time with problem size when the stiffness matrix is assembled from element stiffness matrices that are stored on a sequential secondary storage device. The rapid growth in matrix assembly time for large problems was unacceptable for NASTRAN.

Since the large matrices in NASTRAN require the use of secondary storage for assembly, and since only sequential access procedures were available during the initial development period, the regeneration procedure indicated by the long-dash--short-dash line in Figure 1 was used. In this procedure the required partitions of the element stiffness matrices are regenerated at each grid point as needed. Consequently, the slope of this line is proportional to the number of grid points connected to each finite element. As indicated in Figure 1, this procedure will be superior to the storage of element matrices on sequential access devices for large problems. The location of the cross-over point will depend on the ratio of the time to generate an element stiffness matrix to the time required to retrieve the same information from a sequential storage device. A further degradation in running time occurs if all of the element generation routines cannot be held in main memory at the same time. This latter problem was not severe in the earlier releases of NASTRAN because the element library was small and consisted of relatively simple elements.

The dashed line in Figure 1 indicates a linear growth in matrix assembly time when the element stiffness matrices are retrieved from a direct access secondary storage device. The slope of this line is proportional to the time to generate the stiffness matrix for a single element plus the time required to retrieve the element stiffness from a random access device. A new matrix assembly module has been completed for Level 16 in which the element matrices are generated for each type of element and stored on a random access device. The complete matrix is generated by assembling as many columns of the matrix as possible in packed form in main memory, using random access methods to retrieve the element matrices as needed. Since only a single element routine is needed in main memory during the formation of the element matrices, there is no penalty for having a large finite element library.

The new matrix assembly module also provides for taking advantage of identical finite elements in the model. In the case of identical elements, the element generation routine generates only one matrix for each group of identical elements. With random access assembly procedures, it is a routine operation to point to the single element matrix each time it is required for matrix assembly. This procedure substantially reduces the matrix generation time when there are large numbers of identical elements.

Test runs indicate that the central processor time for the actual matrix assembly is about the same in Level 16 as in Level 15, even though the element matrices are retrieved from a secondary storage device in Level 16. In other words, the overhead for the matrix assembly operations in Level 16 is very small. The removal of the requirement to regenerate the element matrices at each connection will reduce the matrix generation time in proportion to the number of connected grid points. The new matrix assembler is particularly important for the new, higher order elements where the number of connections are often greater, and the generation times for the element matrices are substantially greater than for the elements in Level 15.

MATRIX MULTIPLY-ADD

The use of sparse matrix multiply-add routines was part of the original design of NASTRAN. Major improvements in efficiency were made in Level 15 with the use of machine language inner loops and improved logic for Method 2. Level 16 includes a new Method 3 and improved logic for the transfer of the packed matrix terms directly into the working area for Method 2. The details of the multiply-add operations are given in the NASTRAN Theoretical Manual.

The summary of multiply-add operations in Table 1 presents the overall picture of multiply-add efficiency in NASTRAN. Various combinations of densities of the [A] and [B] matrices are presented for both the nontranspose and the transpose multiply-add options. In each case the most efficient multiply-add method is given for the particular combination of densities. The efficiency of the inner loop for multiply-add is always proportional to the length of the strings of the second operand. The total arithmetic time is always proportional to the density of the matrix containing the first operand, except for the nontranspose option of Method2, for which the arithmetic time is proportional to the product of the densities of the two matrices. Although the matrix packing operations contribute to the total execution time, these packing times are not of primary consideration in the relative efficiency of the multiply-add methods.

For either the transpose or the nontranspose multiply-add, the summary in Table 1 indicates that, when [B] is dense, Method 1 will be selected regardless of the density of [A]. In these cases, the arithmetic times are proportional to the density of [A]. Since [B] is assumed full in Method 1, the multiply-add loop operates at maximum efficiency. However, unless [B] is very dense, a large number of unnecessary zero operations will be performed.

If [B] is sparse, Method 2 will always be selected in the nontranspose case, regardless of the density of [A]. In this case the arithmetic time will be proportional to the product of the densities of the [A] and [B] matrices. The efficiency of the multiply-add loop will be proportional to the lengths of the strings of the [A] matrix. The internal selection procedures assume that the average length of the strings in the [A] matrix is proportional to the density of the [A] matrix.

For the transpose case with [B] sparse and [A] sparse, Method 2 will usually be selected. Alt ough the efficiency of the multiply-add loop will be low due to the short strings (low density) in [A], the number of operations will be proportional to the density of [A], and the total arithmetic time will be relatively short. If there is sufficient main memory to perform Method 1 in a single pass, the total time will be less than for Method 2 because of the higher efficiency of the multiply-add loop. In neither case is any advantage taken of the sparsity of the [B] matrix.

For the transpose case with [B] sparse and [A] dense, a new Method 3 is the most efficient. In this method the [A] matrix is assumed full and the multiply-add loop operates at maximum efficiency. However, unless [A] is very dense, a large number of zero operations will be performed. The execution time for Method 3 is proportional to the density of the [B] matrix, with no advantage being taken of the density of the [A] matrix. A nontranspose option is not provided for Method 3, as Method 2 handles all cases of interest more efficiently.

MATRIX DECOMPOSITION

The storage and indexing procedures used in NASTRAN for the new symmetric decomposition routine will be discussed with reference to the matrix in Figure 2. Initial nonzero terms are indicated by X's with the O's indicating nonzero terms created as the decomposition proceeds. The shaded terms indicate the relative locations for nonzero contributions to the upper triangular factor when the first row of the matrix is the pivotal row. If there is sufficient main storage to hold all of the shaded terms, the decomposition may proceed without the need for writing intermediate results on secondary storage. The shaded terms in Figure 3 indicate the relative locations for nonzero contributions to the upper triangular factor when the second row is the pivotal row. In this case not only are there more active columns, and therefore more main storage is required, but one of the new active columns (column 8) is inserted in an intermediate location.

The management of the working storage for triangular decomposition is indicated in Figure 4. The pivotal row and the associated active column vector are stored in a separate space. The active column vector contains the column number for each nonzero term in the pivotal row. The lower portion of the main working storage is always used and the amount is proportional to the number of active columns at each stage of the decomposition. The amount of storage required for each of the first six pivotal rows for the matrix shown in Figures 2 and 3 is indicated on Figure 4. The shaded area indicates the storage space required for pivotal rows 1, 5 and 6, all of which have six active columns. At any particular stage of the decomposition, the previous contributions are accessed according to the number of active columns immediatly preceding the pivotal row, and the results of the current calculations are stored according to the number of active columns in the pivotal row.

As the decomposition proceeds, the number of active columns can increase by any number up to the number of rows remaining in the matrix. However, if it is assumed that the number of active columns will never decrease by more than one for each new pivotal row, it is possible to store the current calculations dynamically in the same array with previous calculations without interference. This is equivalent to assuming that once a column becomes active it remains active until the column intersects the diagonal (column number = pivotal row number). This assumption will not cause errors in the calculations but will result in the performance of a number of zero operations and will require additional working space in main storage.

The shaded part of Figure 5 indicates the nonzero terms in the upper triangular factor of a matrix where the original nonzero terms are indicated with X's. It can be seen that columns 7, 9 and 13 are terminated at row 3, and columns 11 and 14 are terminated at row 6. The new matrix decomposition routine in Level 16 provides for the termination of active columns by changing their status from active to passive. Columns may also change their status from passive to active as indicated in Figure 5 by column 9 at row 7, or column 11 at row 10. The provision for passive columns reduces the number of active columns when row 4 is pivotal from 6 to 3, with an associated reduction in main storage requirements and the number of arithmetic operations. The complete details of the decomposition procedure will be given in the Level 16 NASTRAN Manuals.

The storage management indicated in Figure 4 applies only when there is sufficient working storage for all of the terms generated by the pivotal row. When the number of active columns exceeds the capacity of working storage space, an automatic spill logic is provided. The overhead for the new spill logic is substantially less than provided in the original matrix decomposition routine. Both the CPU cost and the number of secondary storage transfers have been substantially reduced. It should be possible to economically run large problems with about half as much main storage in Level 16.

In order to improve the efficiency of sparse matrix operation in NASTRAN, the inner loops are usually written in assembly language. In general the use of assembly language will reduce the number of instructions and will allow for more effective use of the high speed registers. In the case of the new decomposition routine, three separate inner loops are provided. The differences in the inner loops are associated with the need to combine the previously completec results in the working storage space. Special provision is made when no previously calculated results need to be combined. This applies to the first row of the matrix and for all rows immediately following the creation of passive columns. Special provision is also made for the case of consecutive active columns. This option improves the efficiency of indexing for band matrices and when there are large numbers of active columns adjacent to the diagonal. The third loop provides for the general case in which a test must be made inside the loop for the existence of previously calculated terms.

The aim in the NASTRAN decomposition routine has been to provide a general purpose routine which will operate efficiently for different orderings of nonzero terms, including the cases of band matrices and partitioning or substructuring types of matrix ordering. The NASTRAN decomposition routine has been designed to take advantage of different sequences of nonzero terms along with the use of an ordinary step-by-step elimination procedure. Test runs with square frameworks of 2600 order have given improvements in running time by factors of 2 to 4. It is easy to design problems which will show substantially greater improvements in efficiency, particularly if the new spill logic is used with reduced main memory requirements or unusual sequences for matrices are employed.

The familiar ordering for a band matrix of a square array is shown in Figure 6. Figure 7 indicates the ordering of the same problem with partitioning. In this case, the square array has been divided into four partitions with each of the partitions numbered first and the boundary points numbered last. Figure 8 indicates the locations of the nonzero terms in the triangular factor when the square array is ordered for partitioning. The X's indicate the original nonzero terms and the 0's indicate nonzero terms created during the decomposition operation. In the case of the band matrix, the number of nonzero terms in the triangular factor is 129, whereas Figure 8 contains only 102 nonzero terms. Since the time for the forward/backward substitution operation is directly proportional to the number of nonzero terms in the triangular factor, the time for the forward/backward substitution operation when the square array is ordered for partitioning is only about 80% of that when the array is ordered for a band. The number of multiplications for the decomposition when ordered for a band is 294, whereas the number indicated in Figure 8 is only 177. This indicates that the time for the decomposition when ordered for partitioning is only about 60% of that when ordered for a band. This example indicates the kinds of savings that are possible in decomposition and equation solution, when the decomposition routine can locate the nonzero terms in a triangular factor in a routine fashion. Even greater savings are possible when the partitions are not strongly connected.

EQUATION SOLUTION

The forward/backward substitution operation for the new equation solution routine in Level 16 is performed by holding as many columns of the right hand side in main memory as possible. The forward and backward substitution operations are performed by reading the triangular factors from secondary storage and performing the indicated arithmetic operations. These operations are performed in place, and at the conclusion of the backward substitution operation the solution vectors are stored in the same locations as the original right hand sides. The nonzero terms of the triangular factors are located directly in the I/Ø buffers in strings, using the new nontransmit option of the matrix packing routines.

The general procedure for the forward pass in equation solution is indicated in Figure 9. The operation for each column begins by locating the first nonzero string in the lower triangular factor. The nonzero terms of the current column of the lower triangular factor are indicated by the letter L in Figure 9. Next, the associated term in the first column of the right hand side is tested for zero. If the right hand side term is nonzero, the multiply-add operations are performed for the string, and the results are stored in the first column in the locations indicated by the number 1 in Figure 9. Similar operations are performed for all columns on the right hand side having nonzero entries in the row associated with the current column of the lower triangular factor. The X's on Figure 9 indicate nonzero operations exist in columns 1, 2 and 5. The results for the second string of the current column in Figure 9 are stored in the locations indicated by the number 2. The forward pass is completed by performing similar operations for all columns in the lower triangular factor.

The back substitution operation is performed by reading the strings of the triangular factor in reverse order. The general procedure for the backward solution is indicated in Figure 10. The operation for each row of the upper factor begins by locating the last nonzero string in the current row, indicated by the letter U in Figure 10. The dot product of the string is made for each column on the right hand side in turn, without testing for zero. The location of the second operand is indicated by the number 1 in Figure 10. The partial solutions are accumulated in the current row of the right hand side, as indicated by the X's in Figure 10. The backward pass continues by performing similar operations for all nonzero strings in the current row of the upper triangular factor. The locations of the second operand for the next nonzero string of the upper triangular factor are indicated by the number 2 in Figure 10. When all of the dot products have been performed for the current row of the upper triangular factor, the solution will be located in the positions indicated by the X's. The backward solution is completed by performing the same operations for each row of the upper triangular factor. The final solution is then transferred to secondary storage. If additional right hand sides exist, the next group of columns can be transferred to main storage and a new forward pass started.

It can be seen that the forward/backward substitution operation in Level 16 takes full advantage of the sparsity of the triangular factors. Also, all terms in the triangular factors are located directly in the buffers, so full advantage is taken of the string notation and the nontransmit packing option. Full advantage is also taken of the sparsity of the right hand side in the forward pass. Test runs with the new equation solution routine show improvements in running time by a factor of 10 over those used in Level 15.

FUTURE DEVELOPMENTS

One of the more important future hardware developments will be the availability of much faster arithmetic units. The improvements in speed will come from the use of parallel processors and the use of vector processors. The modular design of NASTRAN should make it possible to take advantage of these new hardware developments by changing only the matrix operation routines. In some cases, it may only be necessary to change the inner loops in the matrix operation routines. In any event, the basic packing routines and the string notation should be useful with these new types of arithmetic units.

Another important hardware development will be the use of high capacity, high speed, secondary storage devices. These high speed storage devices will consist of such things as extended core storage devices and fixed head drums, as well as high speed, high density disc storage devices. The organization of the NASTRAN packing and I/Ø routines lends itself to easy modification for use with different types of secondary storage devices. A modification has already been made for Level 16 to include the use of extended core storage devices on CDC machines.

The organization of the NASTRAN I/Ø routines and the use of working storage and main memory adapt well to the use of paging devices, such as are used with buffer memories and virtual memory machines. The NASTRAN matrix routines tend to access blocks of information in main memory in a sequential manner. The net result is, that even for large problems, only a small amount of working space needs to be resident in main memory at any one time. Furthermore, particularly with the new Level 16 matrix routines, the number of transfers between main storage and secondary storage have been substantially reduced, with a resulting reduction in the work load for paging devices.

CONCLUSIONS

The following conclusions are drawn relative to improvements in NASTRAN efficiency:

1. Most problems will run at least twice as fast on Level 16 as Level 15 due to improvements in times for matrix assembly, equation solution, decomposition, and matrix packing.

2. The use of multiple I/Ø buffers in main memory along with the nontransmit read and write options can make an individual functional module competitive

with core held programs, because no transfers to secondary storage are made and indexing is done directly into the working arrays.

3. The modular design of NASTRAN has made modification easy and should continue to make it relatively easy to adapt NASTRAN to new hardware and improvements in the state of the art for matrix operations and data processing.

Table 1. Summary of MPYAD Operations

[A][B] + [C] = [D]

Matrix Density		MPYAD Method	Arithmetic Time	
[A]	[B]		Strings	Density
Sparse	Dense	1	[B]	[A]
Dense	Sparse	2	[A]	[A] & [B]
Sparse	Sparse	2	[A]	[A] & [B]
Dense	Dense	1	[B]	[A]

$[A]^T[B] + [C] = [D]$

Matrix Density		MPYAD Method	Arithmetic Time	
[A]	[B]		Strings	Density
Sparse	Dense	1	[B]	[A]
Dense	Sparse	3	[A]	[B]
Sparse	Sparse	2	[A]	[A]
Dense	Dense	1	[B]	[A]

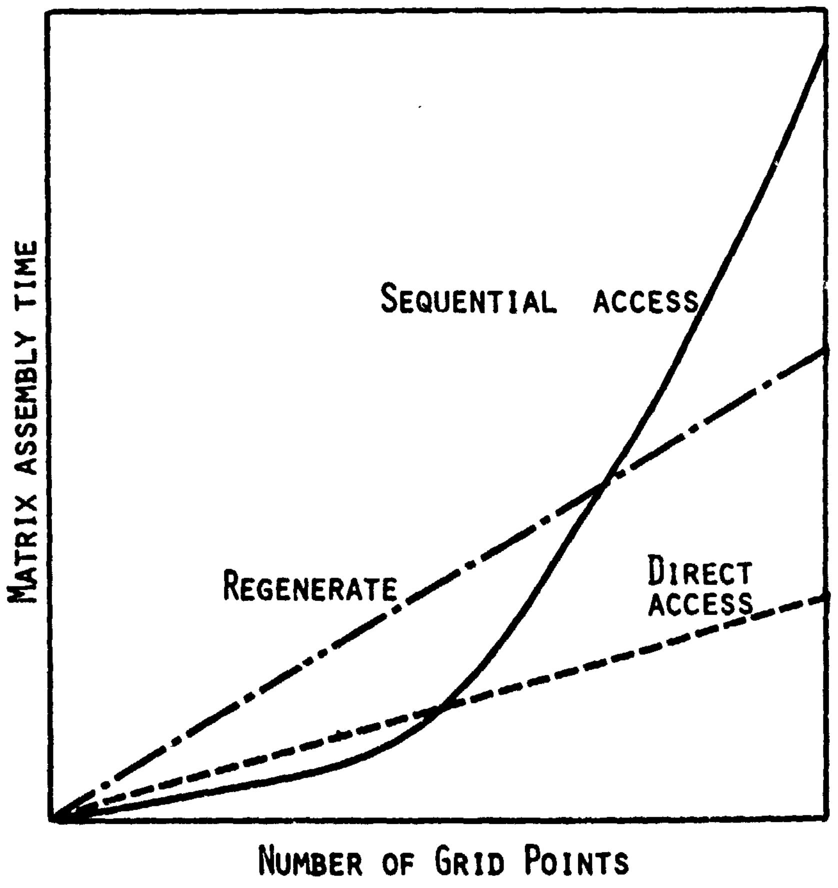

Figure 1.- Comparison of matrix assembly procedures.

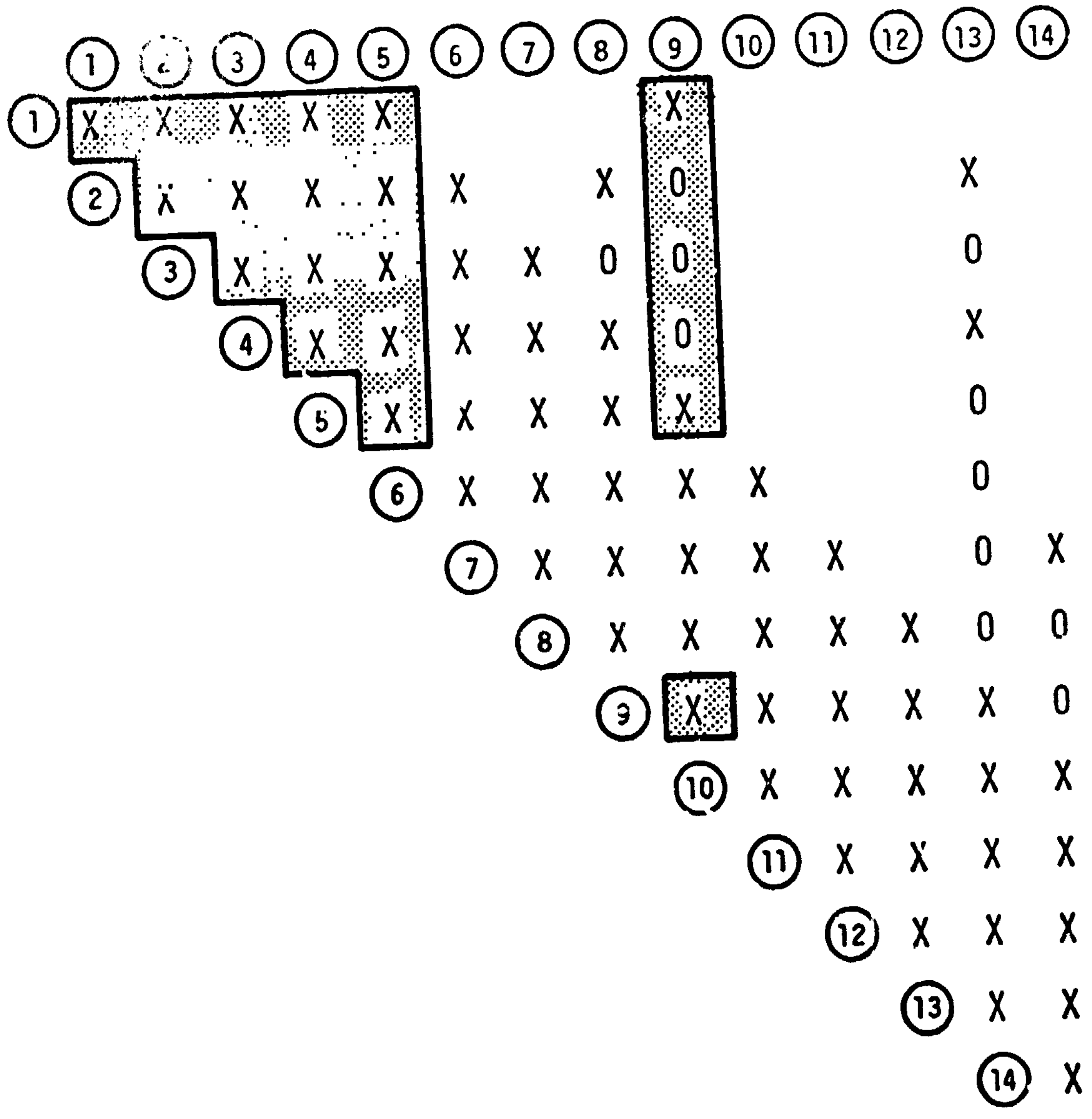

Figure 2.- Decomposition with first row as pivotal row.

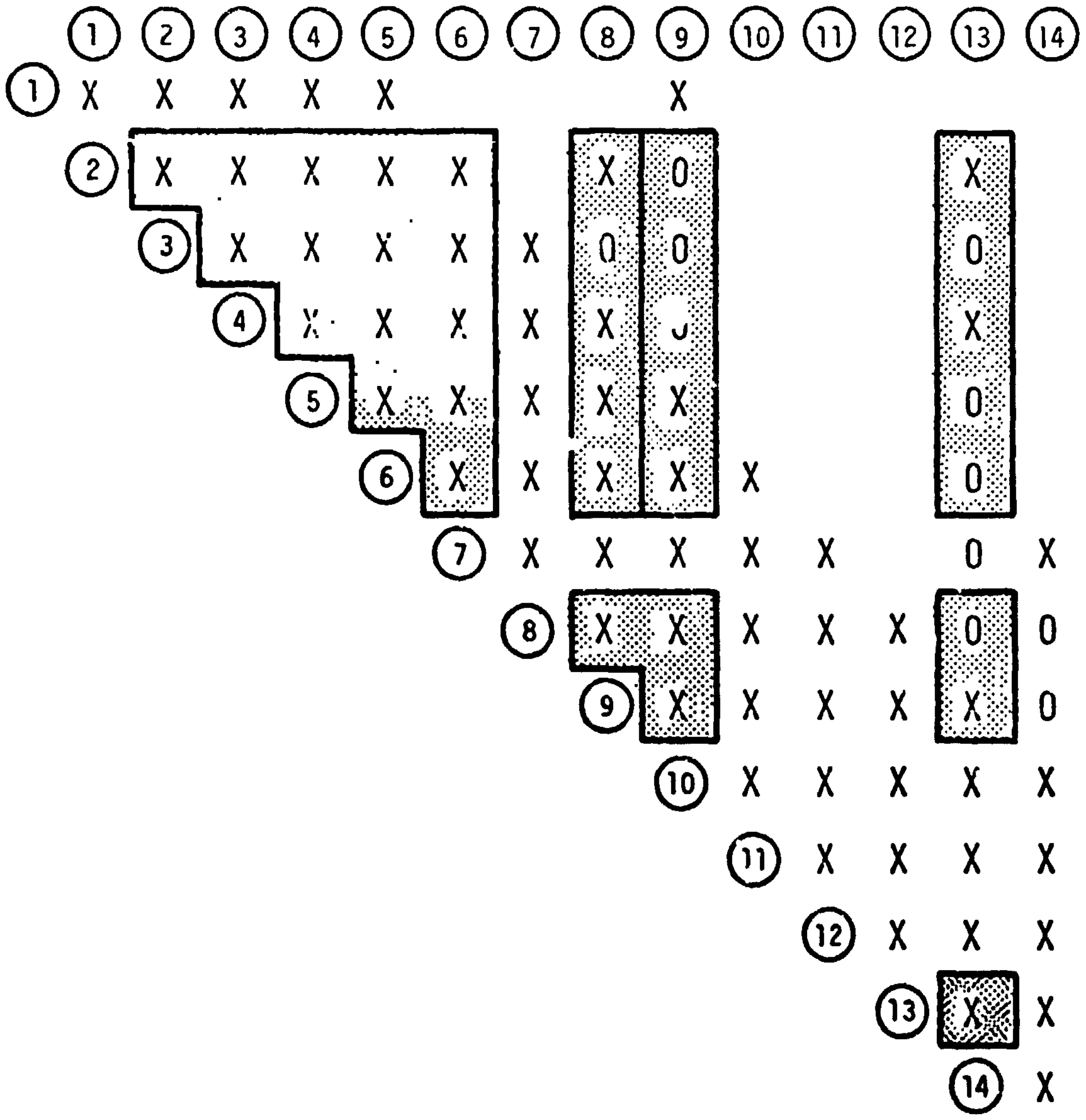

Figure 3.- Decomposition with second row as pivotal row.

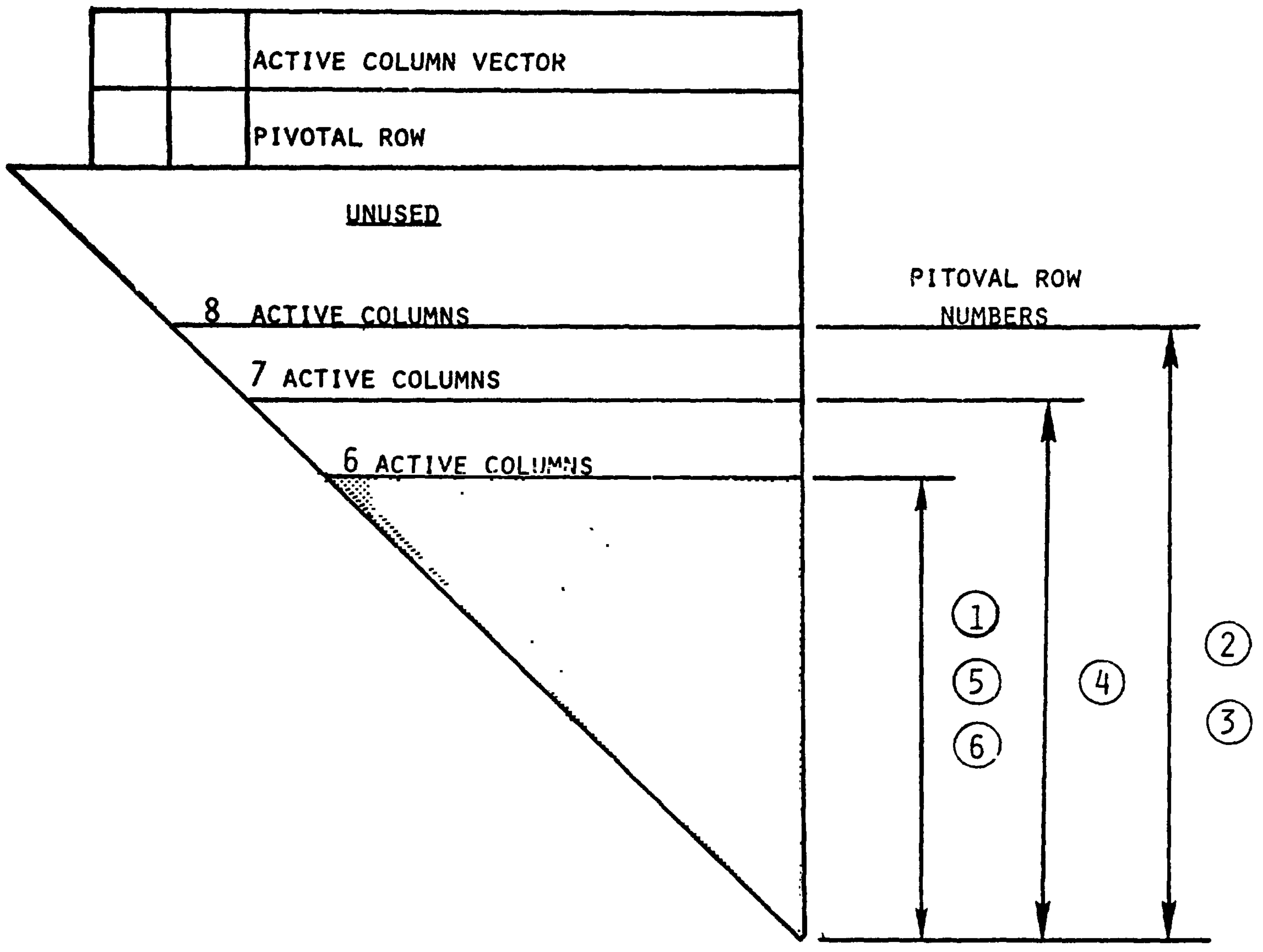

Figure 4.- Compact working storage for triangular decomnosition.

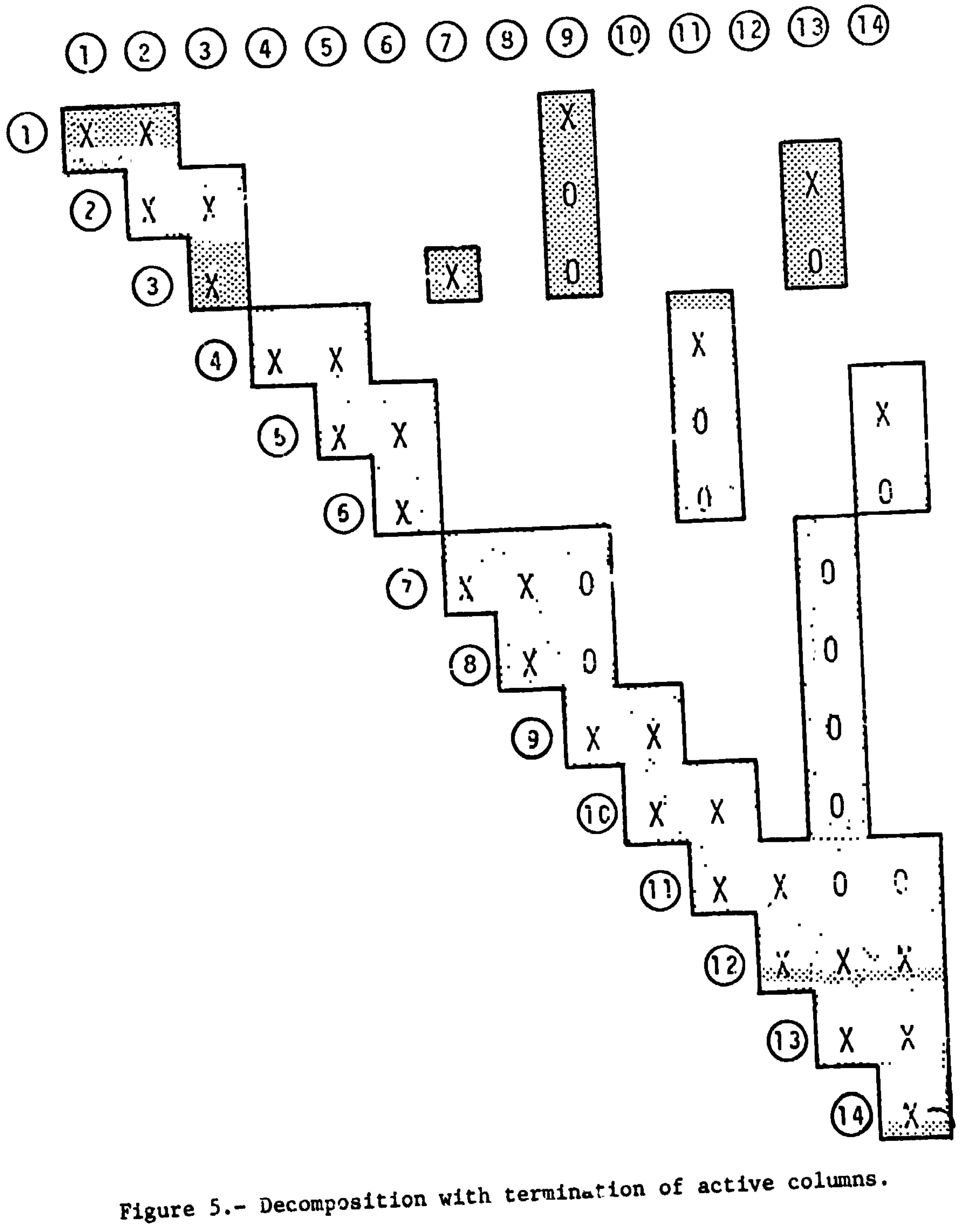

Figure 5.- Decomposition with termination of active columns.

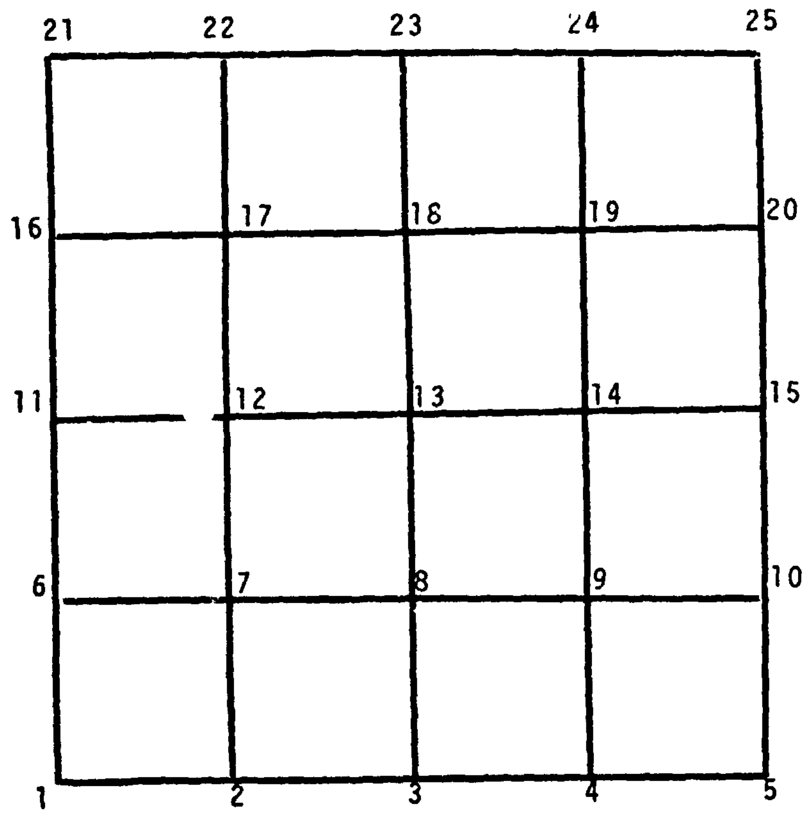

Figure 6.- Ordering for band matrix.

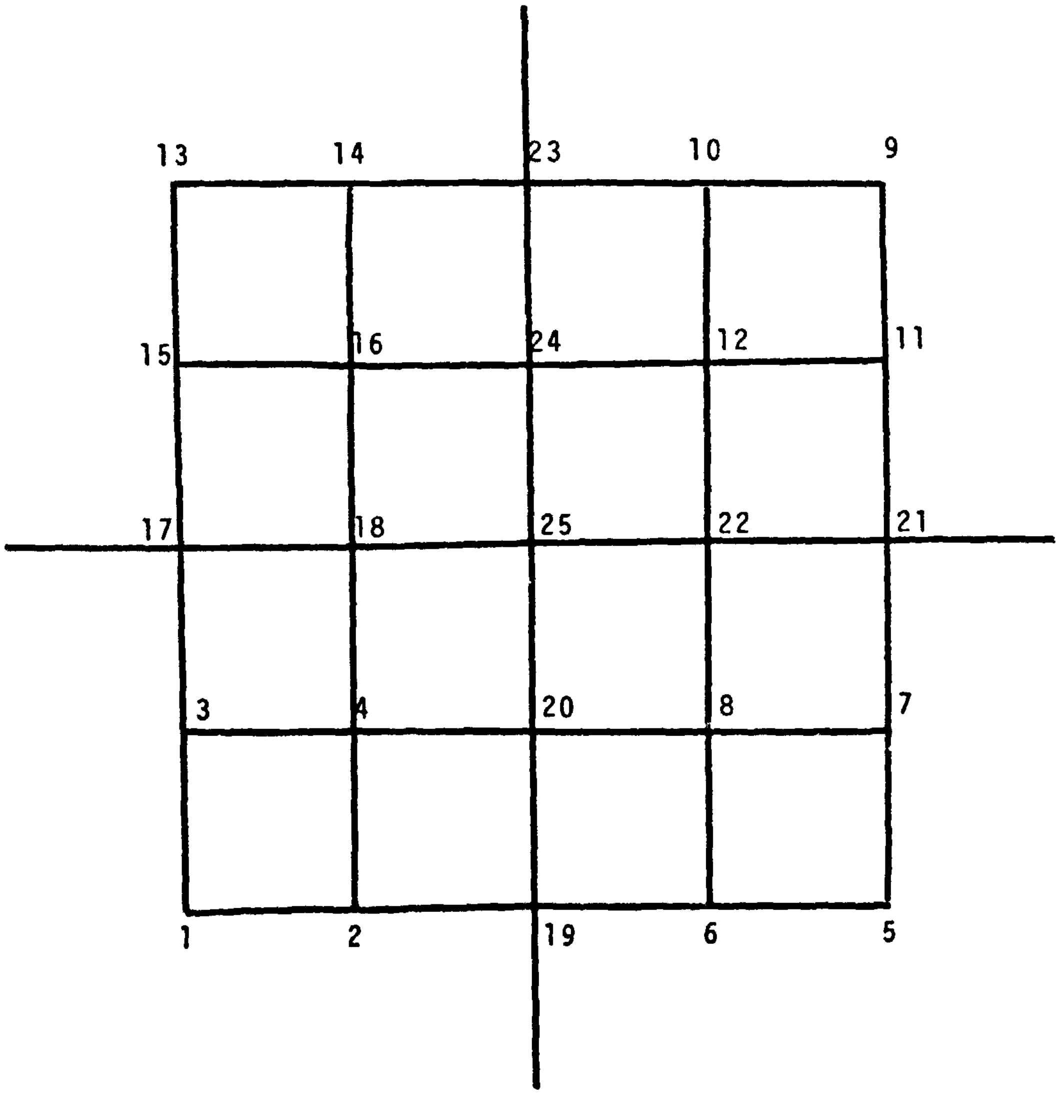

Figure 7.- Ordering for partitioning.

```
X X X
  X O X                             X
    X X                         X   O
      X                         O X O X
        X X X
          X O X                     X
            X X                     O   X
              X                     O X O X
                X X X
                  X O X                     X
                    X X                 X   O
                      X                 O X O X
                        X X X
                          X O X             X
            SYM.
                            X X X           O
                              X O X         O X
                                X X O O
                                  X O O     O O X
                                    X X O O O O O
                                      X O O O O X
                                        X X O O O
                                          X O O X
                                            X X O
                                              X X
                                                X
```

Figure 8.- Nonzero terms in triangular factor when ordered for partitioning.

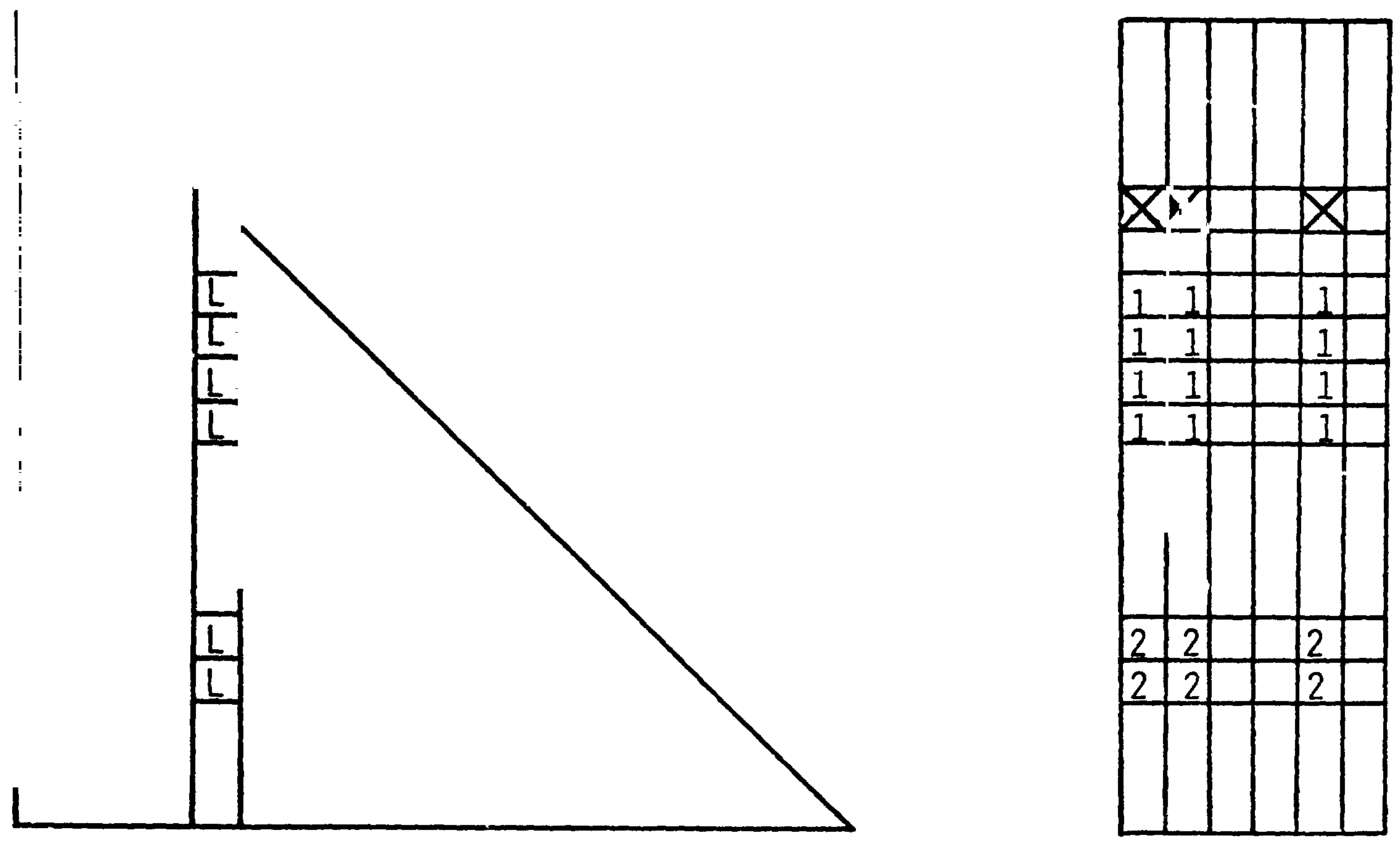

Figure 9.- Forward pass for equation solution.

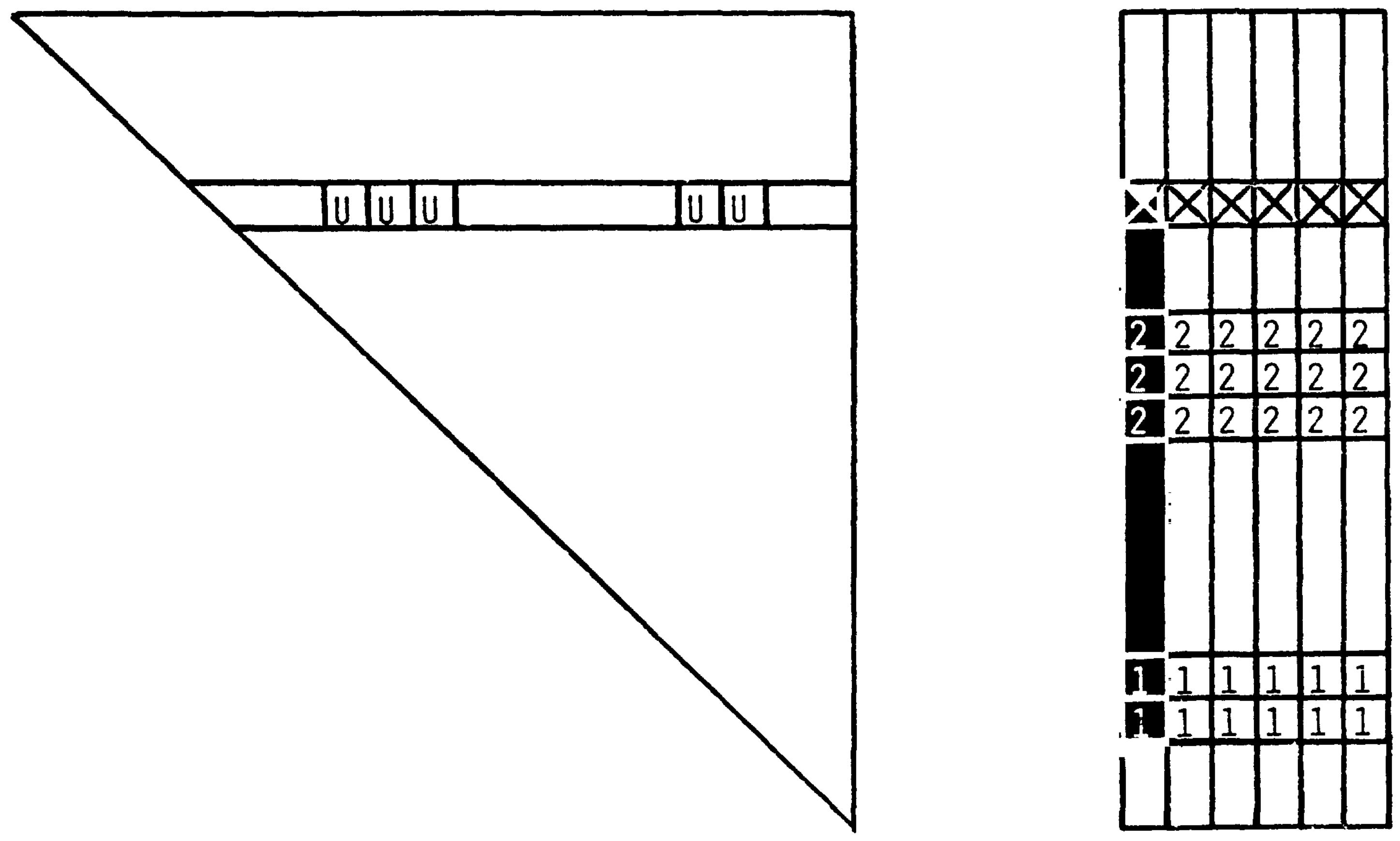

Figure 10.- Backward pass for equation solution.