IMPROVEMENTS IN SPARSE MATRIX OPERATIONS OF NASTRAN

Shinichiro Harano
Hitachi, Ltd.

SUMMARY

This paper describes improvements in sparse matrix operations for the NASTRAN program achieved by Hitachi, Ltd.(Japan). To solve a large scale problem at a high speed, a great emphasis was laid on how to make a reduction in execution time needed by matrix operations, since the size of the problem depends largely on speed of matrix operations as well as on hardware and program performance. The descriptions in this paper are presented under Introduction plus five subjects: Sparse Matrix and Matrix Packing, Matrix Decomposition, Forward Elimination and Backward Substitution, Eigenvalue Extraction Methods and Parallel Processing Oriented Matrix Operations. These improvements can be applied to other versions of NASTRAN with a slight modification by using several subroutines which we have developed.

INTRODUCTION

Since the introduction of NASTRAN level 15.5.1 in 1974, we have improved it by a series of program enhancements. Highlights of them are development of the IG/OG (Input Generator/Output Generator) program to perform automatic meshing and edit the results of calculation, and addition of isoparametric elements of two dimensions, three dimensions or axi-symmetry. Dealt with in this paper is another highlight of them.

Recent drastic improvements in hardware performance have brought a gradual moving from the third generation computers, typified by IBM 370 to distributed computers, also typified by IBM 3033. Being in step with such worldwide trends, Hitachi has developed HITAC M-180 closely comparable with IBM 370/168 and HITAC M-200H providing a throughput three times that of IBM 370/168. Along with these hardware breakthroughs, the parallel processing feature appearing with vector and array processors will be increasingly brought in, changing a current software environment greatly.

Accordingly, in putting a further refined processing system for matrix operations into practice under such situation, one must direct his attentions to hardware as well as software dimensions of the break-throughs. We have confirmed that a more effective use of a vector processor is well attainable by the Gaussian elimination of inner product type, also called "the Skyline method", which was proposed by Prof. Wilson (UCB), rather than by the conventional band matrix algorithm.

SPARSE MATRIX AND MATRIX PACKING

Generally, matrices for structural analysis are characterized by sparsity. To take full advantage of this characteristic in matrix operations, NASTRAN carries out matrix data packing. The way of matrix data packing is especially important for a problem where a large scale matrix is to be handled efficiently. So far, in transmission of matrix data between a secondary storage device and a main memory via an input/output buffer, packing routines have been used to transmit data from the input/output buffer to allow matrix data to be referenced. However, this method is less advantageous to handle a large volume of matrix data since it needs much overhead time for the transmission.

New "non-transmit" packing routine has been added to our version of NASTRAN to allow matrix data to be refered to directly from the input/output buffer. The matrix packing format obtained as a result is shown in figure 1. In this figure, the string is a set of successive non-zero terms, plus the row number and length of string ahead of these terms. Further, the number of strings in one column that are resident at one input/output buffer is given to control how to refer to matrix data resident at the buffer. Padding information is also given to adjust a word boundary for data provided in double precision. At the present, the new packing routine to perform a direct reference to the input/output buffer makes only the READ option effective. This non-transmit type of routine called string by string receives or sends:
(1) the start adress of a string in a buffer
(2) the foremost row number of a string
(3) the length of a string
(4) the instruction to show whether or not EOL (end of column detection is to be made.)
(5) type of matrix data

Use of the packing routine permits various routines for matrix handling to perform a direct reference to the input/output buffer if once they have received data addresses. The packing routine offers a buffer-by-buffer backspace feature for efficient backspacing in sequential access. Unlike a conventional backspacing that needs twice back record for a single read of one record (one column), as shown in figure 2, this feature omits overlapping of READ operation and back record, as also shown in figure 3. This feature eliminates the necessity of writing, in decomposition of a symmetric matrix, of a portion of the matrix to its upper triangular matrix from the last to the first columns of the symmetric matrix, thus saving time for generating the upper triangular matrix. Furthermore, the feature requires the writing of only a lower triangular matrix onto the secondary storage device, bringing 10 to 30% reduction in use of the disk space of the storage device.

This new matrix packing technique is fully employed in the matrix decomposition described in MATRIX DECOMPOSITION. Figure 4 reveals how the technique is superior to conventional techniques by comparisons of packing routines to pack and unpack one column of the matrix in respect to CPU time versus non-zero term densities. The CPU time for packing/unpacking of one column is on the ordinate, while non-zero term densities are on the abscissa. The length of one column is 2000 and the whole CPU time has been obtained as the result of 300 iterations. Packing routines mutually compared in this figure are:

(1)	INTPK	: Clears the area for one column to zero to perform element-by-element unpacking. (This is a conventional unpacking routine.)
(2)	UNPACK	: Unpacks a column at one time. (This is a conventional unpacking routine.)
(3)	PACK	: Packs a column at one time. (This is a conventional packing routine.)
(4)	INPNT	: Clear the area for one column to zero and transmit string data directly from the buffer to the appropriate location of the column. (This is the new packing routine.)

Figure 4 also shows CPU time for READ and WRITE operations in case of GINO (General Input Output Routines) as additional information. Although the READ and WRITE operations may be performed irrespectively of non-zero term density of the matrix, they cannot take advantage of sparsity in case of low density due to unsatisfactory efficiency. Further, all elements including non-zero ones are written out onto the secondary storage device, making an increase in disk storage space needed.

As a result, we adopted a combination of the clearing a core space to zero and the new routine of non-transmit type to unpack columns in matrix decomposition. In short, figure 4 reveals that the new unpacking routine permits a speedup approximately 2.1 times the conventional unpacking routines in such a situation that densities of unpacked one column usually fall in the range of 0.4 to 1.0, owing to the method of matrix decomposition described below.

MATRIX DECOMPOSITION

In solving a given system of linear equations, where the coefficient matrix is a large scale sparse matrix, it is general to decompose the matrix as,

$$A = L*D*U \qquad (1)$$

where, A is the original matrix (e.g. stiffness matrix), L is a unit lower matrix, U is a unit upper matrix and D is a diagonal matrix which is usually part of the diagonal portion of L or U. The discussion below assumes that the original matrix is symmetric. To decompose the matrix, the Skyline method was used. The name of "Skyline" is derived from the fact that the contour line of column's all foremost none-zero elements is similar to a skyline. This method divides a portion enclosed in a skyline and a diagonal line into two groups whose sizes are such that the groups are capable of being resident at a free area of a virtual storage space. The contents of these groups may be read from the secondary storage device which the results of calculation may be written out onto as needed. This is shown in figure 5.

Unlike conventional techniques, the Skyline method employs an upper triangular matrix. The diagonal matrix is generated on a diagonal terms of triangular matrix U . The original symmetric matrix is decomposed in the following algorithm.

$$u^*_{ij} = a_{ij} - \sum_{k=1}^{i-1} u_{ki} u^*_{kj} \quad (i=2, \ldots, j-1) \tag{2}$$

$$u_{ij} = u^*_{ij}/d_{ii} \quad (i=1, \ldots, j-1) \tag{3}$$

$$d_{jj} = a_{jj} - \sum_{k=1}^{j-1} u_{kj} u^*_{kj} \tag{4}$$

The method carries out (2) through (4) in succession for $j=2,\ldots,n$ where, n is the dimension of matrix A . $d_{11} = a_{11}$ is assumed. First, the algorithm for the Skyline method used for an incore routine is described with the help of the explanatory illustration of figure 6. This method employs the Gaussian elimination of inner product type, as shown in this figure. All column's elements from foremost non-zero ones to diagonal ones, including zero elements, are stored on the memory. The store address at that time is resident at array M. The address of the I-th row diagonal term is expressed by M(I). The value at the I-th row and the J-th column (position pointed by IJ) is determined by the inner product of vectors P and Q . Vector P has a length of:

$$JH = M(J) - M(J-1) \tag{5}$$

If JH=1, the diagonal terms are those obtained by matrix decomposition, and therefore, processing is skipped. Vector Q, a party of inner product with P, satisfies:

$$J > I > J-JH \tag{6}$$

Let NT be a length of the intersection of vectors P and Q . Since the length of vector Q is,

$$IH = M(I) - M(I-1) \tag{7}$$

length NT is,

$$NT = \mathrm{Min}\left\{ I - (J - JH),\ IH \right\} - 1 \tag{8}$$

The following processing is performed only if NT is positive. Let NS be a start address of element being involved in inner product calculation of vectors P and Q . Then, NS may be expressed as:

$$NS = M(I) - NT \tag{9}$$

The address of the I-th row and the J-th column element to which the inner product value is to be added is,

$$IJ = M(J) - (J-I) \tag{10}$$

The distance (IC) between the addresses of foremost elements of vectors P and Q is,

$$IC = IJ - M(I) \tag{11}$$

Thus, letting X be a vector storing elements on the memory, the foremost elements for inner product calculation of vectors Q and P are,

$$X(NS) \text{ and } X(NS+IC), \text{ respectively.} \tag{12}$$

The length of inner product, then, is NT. Assuming that the value of inner product between vectors P and Q is S, the I-th row and the J-th column is obtainable by:

$$X(IJ) = X(IJ) - S \tag{13}$$

By performing the calculation for all I restricted by (6), the entire J-th column may be obtained. Notice that the value obtained by (13) corresponds to that by (2). In a practical LU-decomposition, as shown by (3), each element of the J-th column must be divided the corresponding diagonal element (See (15)). The J-th row and the J-th column diagonal term is,

$$X(IJ) = X(IJ) - \sum_{K=NS}^{NE} \frac{X(K)*X(K)}{X(KD)} \qquad (I=J,\ KD=M(K)) \tag{14}$$

The last result for the I-th row and the J-th column is,

$$X(K) = \frac{X(K)}{X(KD)} \qquad (K=NS,\ \ldots,\ NE) \tag{15}$$

By carrying out the above process for $2 \leq J \leq n$, the upper triangular matrix of matrix A may be generated in X.

The above algorithm is well applicable to matrix calculation if all matrix elements are capable of being stored on the memory. Otherwise, matrix grouping is needed prior to implement the Skyline method.

Assume that the original matrix is such a matrix as shown in figure 7. First, this matrix is divided into some groups, each of which should not have more elements than those restricted by a core space. In the example of figure 7, the matrix is divided into four groups, each of which has not more than 15 elements. These divided groups provide the information of table I ; headings of the table are:

(1)	K(1)	: the current group number
(2)	K(2)	: minimum group number needed by calculation of group K(1)
(3)	K(3)	: minimum column number in group K(1)
(4)	K(4)	: maximum column number in group K(1)
(5)	M(J)	: pointer array of the diagonal term in the J-th column

With the matrix grouped above, the algorithm of the Skyline is proceeded as follows. Symbols used in the description are:

X	: open core array
IA	: start address of group A element
IB	: start address of group B element
IM	: start address of pointer array of diagonal terms
KA(1), KB(1)	: group numbers for groups A and B
KA(2), KB(2)	: minimum group numbers involved in calculation of groups A and B
KA(3), KB(3)	: minimum column numbers of groups A and B
KA(4), KB(4)	: maximum column numbers of groups A and B

Apart from the explanation of how to determine these values, we begin our discussion with the following assumptions. The areas are already assured for groups A and B (headings are X(IA) and X(IB)) and for the address of diagonal terms (heading is X(IM)). The diagonal term addresses are set in advance. Let NEQ be the number of unknowns of a given system of linear equations, and NGP be the number of groups. All control information for NGP groups, except those for diagonal term addresses, are generated on a scratch file in advance.

Then, the following steps are repeated for P until NGP.

P = 1, ... , NGP (P : group number in current calculation)

Mutually permutated, for $P \neq 1$, are the address of A and that of B, and group information of A and that of B, that is,

$$IA \rightleftarrows IB, \quad KA(k) \rightleftarrows KB(k) \quad (k=1,\ldots,4) \tag{16}$$

Let Q be the minimum group number needed to generate group P. Then,

$$Q = KA(2)$$

For $P \neq Q$, group Q is already on B if Q = P-1; otherwise the control information about group Q is read from the scratch file to be set to KB(k), where k=1,..,4. Then, the correlation values between groups P and Q are added to group P. This process is carried out for Q by an increment of 1 until Q = P-1. For Q = P, the correlation between P and Q becomes that between P and itself on A. After the completion of the above step for group P, the elements of it are written out onto the secondary storage device. The implementation of these procedures for individual groups ($1 \leq P \leq NGP$) allows an upper triangular matrix to be generated in the column direction on the secondary storage device.

The correlation between groups P and Q is obtained as follows.

Assume that group P is on A (heading : X(IA) area), while group Q is on B (heading : X(IB) area). Let NTA be the number of columns of group P on A, and NTB be the number of columns of group Q on B. Then,

$$NTA = KA(4) - KA(3) + 1 \tag{17}$$

$$NTB = KB(4) - KB(3) + 1 \tag{18}$$

Similar to figure 6, handling of groups P and Q is shown in figure 8. Let J=1, ... , NTA be column numbers of group P on A. Then, the column number of P on the entire matrix is,

$$JJ = KA(3) + J - 1 \tag{19}$$

The length of column J is,

$$JH = M(JJ) - M(JJ-1) \tag{20}$$

where, if JJ=1,

$$JH = M(JJ) \tag{21}$$

Similarly, let I=1, ... , NTB be column numbers of group Q on B. Then, the column number of Q on the entire matrix is,

$$II = KB(3) + I - 1 \tag{22}$$

The length of column I is,

$$IH = M(II) - M(II-1) \tag{23}$$

where, if II=1,

$$IH = M(II) \tag{24}$$

Again, let NT be the length of inner product of column J in group P and column I in group Q. Then,

$$NT = Min\left\{IH,\ JH-(JJ-II)\right\} -1 \tag{25}$$

Also, let NS be the start address involved in the inner product of column I in group Q and NE be the address of the last portion. Then,

$$NS = M(II) - NT \tag{26}$$

$$NE = M(II) - 1 \tag{27}$$

The displacement (IJ), where the inner product value is added on area A, is expressed as:

$$IJ = M(JJ) - JJ + II \tag{28}$$

If NS>NE, column I in group Q has nothing to do with the calculation for column J in group P; otherwise, inner product must be calculated. Since the start addresses of areas A and B on the open core are IA and IB, respectively, by putting,

$$IC = IJ - M(II) \tag{29}$$

the inner product is,

$$X(IJ) = X(IJ) - \sum_{K=NS}^{NE} X(IB+K-1)*X(IA+K+IC-1) \tag{30}$$

The calculation of contributions from group Q to group P is completed if the above steps are carried out for all columns in group Q.

After calculations on all groups such that,

$$KA(2) \leq Q \leq P-1 \tag{31}$$

the autocorrelation of group P itself is obtained by the same procedure as that used by previous calculation of the correlation between groups P and Q by regarding that area A is the same as area B. In this calculation, I varies within the range:

$$1 \leq I \leq J \tag{32}$$

If I=J, (30) must be replaced by the procedure below. For K such that,

$$NS \leq K \leq NE \tag{33}$$

ID = IJ - KJ + 1 is obtained. If KD = M(ID) is established, the column J is completed by (14) and (15).

The implementation of the above procedure for the range of (32) gives the autocorrelation of group P. The results are written out onto an upper triangular matrix data-block for each column.

The interchange of addresses and control information by (16) are needed for handling a succeeding group. This takes the place of moving group P completed on area A to area B simply by interchanging addresses and control information. This eliminates a data transmission, and further allows one group to be read in the core only once if the correlation between two groups affects only adjacent groups.

As already suggested, the matrix must be prepared for grouping prior to the implementation of the algorithm of the Skyline method if matrix data overflows the core space available. First, the size of each group must be determined to provide such control information as in table I. The size of a group depends upon how the open core is large at execution. Figure 9 presents an open core layout. Given the open core size (NX), the size of the memory to be allocated to one group may be obtained by bisectioning the area excluding a working area needed. This allows up to MAXT elements of one group to be stored. By use of the new unpacking routine, only start row numbers of each matrix column are picked up to create table I information. Such information are stored on a scratch file in such a way that each group is on one record. At the same time, the addresses of diagonal terms of each group at a working area are also stored on array M. Thus, the preparation is completed.

To read group P, The unpacking method must be used in which the input/output buffer can be directly referred to from an input matrix data-block. This is also applied to reading Q, where the buffer is directly referred to from an upper triangular matrix data-block. After the completion of calculation, each group is written out following the end of the upper triangular matrix. The diagonal elements are also written out onto another output data-block for succeeding forward elimination and backward substitution.

The following are the results obtained by applying the Skyline method to practical examples. Table II gives matrix characteristics for four data with a comparision of matrix characteristics in case of the conventional band matrix method. Figure 10 compares CPU time for the band matrix method with that for the Skyline method. The Skyline method in these examples gives two cases in which a vector processor has been applied and it has not been applied. The vector processor has been also applied to the band matrix method, resulting in no improvement of CPU time. This figure, therefore, does not that case. Figure 10 reveals that the Skyline method consumes 33 to 66% of CPU time needed by the band matrix method. If the vector processor is applied to the Skyline method, this value drops to as many as 16 to 28% of the CPU time. Further, for data of 7000 degrees of freedom, the Skyline method has needed CPU time on the same percent basis as the band matrix method.

FORWARD ELIMINATION AND BACKWARD SUBSTITUTION

NASTRAN is designed to proceed forward elimination and backward substitution while retaining vectors of load terms on the memory as much as possible. As the result of matrix decomposition by the Skyline method, an upper triangular matrix is generated and diagonal terms are stored on another file. This means that forward elimination is an inner product type and that store-type calculation is to be carried out after division of load terms by diagonal elements. The procedure for solving a system of linear equations:

$$U^tDUX = B \tag{34}$$

is separated into the forward elimination process

$$U^tY = B \tag{35}$$

and the backward substitution process

$$UX = D^{-1}Y \tag{36}$$

The forward elimination process is shown in figure 11. In this figure, l_{ij}, $l_{i,j+1}$, $l_{i,j+2}$ forms a string starting with the i-th column and the j-th element of the upper triangular matrix (string length is 3 here). For this string, the following calculation is carried out.

$$b_{ia} = b_{ia} - \sum_{k=j}^{j+ST-1} l^{*}_{ik} * y_{ka} \qquad (a=1,2,3) \tag{37}$$

This is performed for all i-th column strings in the upper triangular matrix. Then, this procedure is repeated after setting i=i+1. Since (37) is an inner product type, high speed calculation is possible. Further, as for string's elements, the new packing routine refers to the input/output buffer, saving time for data transmission and reducing time for calling the unpacking routine due to string-by-string call unlike conventional element-by-element call. Thus, the forward elimination process is inner-product-type operation for matrix's factors decomposed by the Skyline method, while the forward elimination process of the conventional method is store-type calculation.

On the other hand, the backward substitution process begins with the generation of $D^{-1}Y$ for load term Y already generated by the forward elimination process. The process allows calculations independent of the following process because all diagonal term elements are already generated on a file as one vector on matrix decomposition. After the division of load terms by diagonal terms, backward substitution is performed by backspacing the upper triangular matrix file buffer-by-buffer. This is shown in figure 12. In the process,

$$y'_{ka} = y'_{ka} - u_{kj}^{*} x_{ja} \qquad (k=i,i+ST-1;\ a=1,\ 2,\ 3) \tag{38}$$

is calculated for each string of each upper triangular matrix column. At that time, the non-transmit unpacking routine is called for each string, and only addresses in the input/output buffer are passed to the routine of forward elimination and backward substitution.

Figure 13 gives the results of forward elimination and backward substitution by use of data shown in table II. This figure reveals that the new method requires only 16 to 54% of CPU time needed by forward elimination and backward substitution of the conventional method. Taking it into account that the coding for both processes are not oriented to the vector processor, more satisfactory results will be expected in respect to CPU time by further improvements.

EIGENVALUE EXTRACTION METHODS

In real eigenvalue extraction methods we attemped to develop these methods in two directions: that is, partly the speeding up of the Inverse Power method, partly the development of a simultaneous iteration method.

At first we describe the Inverse Power method. Eigenvalue extraction methods are generally divided into two groups: tracking methods (the Inverse Power method and the Determinant method) and transformation methods (the Householder method and the Givens method). Though transformation methods are able to solve rapidly an eigenvalue problem in the range of comparatively small scale problems, large scale problems are unfavourable to them. On the contrary, the Inverse Power method has been employed frequently owing to less restriction than transformation methods.

Since the Inverse Power method in NASTRAN is accompanied by movements of shift points, it needs to use iteratively matrix decomposition and FBS (forward elimination and backward substitution). Improvements that were previously mentioned were applied to the Inverse Power method, so that we could improved the CPU performance of the Inverse Power method which is two or three times as much efficient as that of the conventional Inverse Power method. Table III shows that the CPU performance of the new method without the vector processor amounts to 2.7 times that of the old one. If the vector processor is applied to the former, the CPU performance of it will be equal to about 3.3 times that of the conventional one.

Now we describe a simultaneous itetation method which is called the Jennings method. There is a problem to find q eigenvalues in ascending order from the lowest value and q eigenvectors corresponding to them for the general eigenvalue problem:

$$K \bar{x} = r M \bar{x} \tag{39}$$

where K is a symmetric matrix of positive definite type and M is a symmetric matrix of non-negative definite type. The Jennings method is useful for calculating a set of eigenvalues from the lowest value and has no weakpoint that some important eigenvalues are often missed in calculation. The algorithm is shown in figure 14. This method is different from the Subspace Iteration method on operations of orthogonalization shown in (i), (j), (k), (l).

The Jennings method needs the following input data:

(1) ND : number of eigenvalues to be extracted ($2 \leq ND \leq 90$).
(2) LMAX: the maximum number of subspace iterations (the default value is 16).
(3) IEP : convergence parameter (if $IEP < 0$, then $EPS = 10^{IEP}$; otherwise EPS=0.0001).

The dimension "m" of subspace is decided on by

$$m = \min(n,\ 2q,\ q+8) \tag{40}$$

The selection of initial iteration vectors is most important for the convergence of subspace and the convergence ratio is decided on by the "neighborhood" between subspace spanned by eigenvectors and subspace spanned by initial iteration vectors. Assume that

$$K = (k_{ij}) , M = (m_{ij}) \quad : k_{ij} = k_{ji} , m_{ij} = m_{ji} \tag{41}$$

Then, $k_{ii} \neq 0$ is always satisfied as K is positive definite. The matrix Go which is composed of m initial vectors will be generated as follows.

(1) At first, the first column of Go is a vector D, where $D(I) = m_{ii}$

(2) check $k_{ii} \neq 0$, and

$$D(I) = D(I) / k_{ii} = m_{ii} / k_{ii} \tag{42}$$

(3) sort D(I) (I=1, ... , n) and select (m-1) values in decending order from the largest:

$$D(I_2) \geq D(I_3) \geq \ldots \geq D(I_m) \tag{43}$$

where the i-th vector of Go (i= 1, ... , m) is the unit vector, the i-th component of which is equal to 1.

The criterions of convergency are as follows:

(1) q eigenvalues and q eigenvectors are extracted.
(2) number of subspace iterations amounts to LMAX.
(3) there is no CPU time to execute three subspace iterations, because output of the results needs a little time.

Let the eigenvalues in the L-th iteration loop be on a vector E(I) (I= 1, ... , m). After reordering E(I) in ascending order, the eigenvalues in the (L-1)-th iteration loop are stored on a vector E*(I) (I= 1, ... , m). If E(I) satisfies the following relation,

$$\left| \frac{E(I) - E^*(I)}{E(I)} \right| < EPS \tag{44}$$

then E(I) is already converged; otherwise E(I) is not converged. If (44) holds true for all I ($1 \leq I \leq q$), the convergence will be achieved owing to criterion (1). If (44) doesn't hold true for some I and number of subspace iterations is equal to LMAX, the calculation of eigenvalues will be stopped due to criterion (2).

The orthogonalization of subspace is also important for a simultaneous iteration method. If a mass matrix M is non-positive definite and operations of orthogonalization isn't applied to subspace during iteration loops, the orthogonality of subspace will be breaking. For a eigenvalue problem with a non-positive definite mass matrix, the orthogonalization of subspace is necessary for iteration vectors to converge to eigenvectors. Consequently we adopted the Jennings method which orthogonalizes iteration vectors just after the calculation of eigenvalues in subspace. The generalized Jacobian method is adopted in the eigenvalue extraction on subspace.

Table III shows that the CPU performance of the Jennings method without a vector processor (or with it) is 4.0 (or 4.1) times as high as that of the conventional Inverse Power method. Thus, the Jennings method consumes about two-third of CPU time used by the new Inverse Power method. This fact results from the following reason. While the Inverse Power method needs several decompositions of full size matrices in every movements of shift points, the Jennings method is more efficient to solve large scale eigenvalue problems than the Inverse Power method, for the former needs only one decomposition of full size matrix and several decompositions of small scale matrices on subspace.

PARALLEL PROCESSING ORIENTED MATRIX OPERATIONS

Our vector processor adopts a pipeline system and uses a compiler system in which a FORTRAN source program is translated into a set of instructions specially for the vector processor with recource to the option active in compilation. This means that the object program generated by the compiler depends on the skillfulness of coding.

To discuss more specifically, this section presents the results of our test. In this test, we measured CPU time per single term for the length of a DO loop (string length) by carrying out three inner product type operations and one store type operation, in order to determine the basic operation in matrix decomposition. The results are shown in figure 15. As for the inner product type operations, two cases were further considered: the case where the vector processor was applied and the case where it was not applied. The examples of coding used in our test are:

(1) Complete inner product type

```
      REAL*8  A(1000), B(1000), X(ITER), SS
      DO 10 I = 1,ITER
      SS = 0.0D0
      DO 20 J = 1,LL
   20 SS = SS + A(J)*B(J)                        Inner product loop
      X(I) = X(I) - SS
   10 CONTINUE
```

(2) Index explicit type

```
      REAL*8  X(1), SS
      DO 10 I = 1,ITER
      SS = 0.0D0
      DO 20 J = 1,LL                             Explicit index type
   20 SS = SS + X(IA+J-1)*X(IB+J-1)              inner product loop
      X(IC+I-1) = X(IC+I-1) - SS
   10 CONTINUE
```

(3) Subroutine inner product type

```
    REAL*8  X(1), SS
    DO 10 I = 1,ITER
    SS = 0.0D0
    CALL DOTP (X(IA), X(IB), SS, LL)
    X(IC+I-1) = X(IC+I-1) - SS
 10 CONTINUE
```

```
       SUBROUTINE DOTP (A, B, SS, LL)
       REAL*8  A(LL), B(LL),  SS, S
       S = 0.0D0
       DO 10 I = 1,LL
       S = S + A(I)*B(I)
    10 CONTINUE
       SS = S
       RETURN
       END
```

(4) Store type (the three terms operation)

```
    REAL*8  X(1) ,SS
    DO 10 I = 1,ITER
    DO 20 J = 1,LL
    X(IC+J-1) = X(IC+J-1) + X(IA+J-1)*X(IB+J-1)
 20 CONTINUE
 10 CONTINUE
```

The above examples of coding are only for our test and there is no meaning in operation itself. The index ITER is the number of iteration loops and ITER = 2000 in our test. Though the store type (the three terms) operation is applicable to the vector processor by changing its indices, at that time we left it as it was, and then it was not applicable to the vector processor.

A close observation of figure 15 first exhibits that, as for the complete inner product type operation, use of the vector processor brings about an improvement in speed as much as 5.5 times that obtained by the same type of operation without the vector processor. Unfortunately, however, NASTRAN is not oriented to the way of coding for the complete inner product type, since it uses an open core as a working area. Thus, two possible ways for coding are explicit index and subroutine inner product types. Without a vector processor, the subroutine inner product type is more advantageous than the explicit index type. On the other hand, with the processor, the former is less advantageous than the latter.

Further, figure 15 reveals that, with the vector processor, the explicit index type almost keeps in step with the complete inner product type in respect to CPU time. However, without the processor, the former has consumed CPU time as much as 2.2 times that the latter has consumed. For a longer inner product loop, the subroutine inner product type without the vector processor is more advantageous than the explicit index type without it. This is attributable to that there is a difference in optimization level between the operation with the vector processor and that without it. The subroutine inner product type is advantageous if the subroutine's overhead time can be overridden due to a long DO loop; otherwise, it is less advantageous than other types in speed. The result of the store type operation is also exhibited in this figure; this type does not enjoy the maximum benefits of optimization.

An observation of figure 15 also shows that the extent of optimization in various types of operations depends largely on program coding. Of course, it is ideal that the maximum optimization is always possible for any type of operation; however, the extent of optimization varies depending upon type of FORTRAN. Accordingly, in coding the algorithm for the Skyline method of matrix decomposition, we adopted the explicit index type if use of a vector processor was possible; otherwise, we used the subroutine inner product type. In the future we intend to use the explicit index type as long as the optimization feature of FORTRAN is satisfactorily refined.

So far, the inner product type has been more advantageous than the store type in respect to speed thanks to use of registers. However, the advent of a vector or array processor is changing this situation. Actually, in case of HITAC M-200H, the latter has displayed almost the same performance as the former. Further improvements of the parallel processing systems may reverse the superiority of the inner product type to the store type.

As already described, CPU time needed for the store type, inner product type operations accompanied with or without data transmission depends largely on how to make a program. Use of a higher speed computer and parallel processing system is greatly expected to change a current software environment to a large extent. Technological breakthroughs of software and of hardware would interact more closely in improving sparse matrix operations.

CONCLUDING REMARKS

In this paper we discussed about improvements in sparse matrix operations of NASTRAN. Recent advance of parallel processing systems has been changing surroundings in software. Especially, a vector processor attached to a general-purpose computer is favorable to a long DO loop operation. For example, the Skyline method which we have developed this time in the field of matrix triangular decomposition conforms to the pipeline control feature observed in the vector processor. On the contrary, the conventional band matrix method or the wavefront method which adopt store type operations don't adapt themselves to the pipeline control system, for they need complicated indices operations and are difficult to deal with a set of arithmetic data as vectors.

What is more, the way of packing/unpacking and the method of forward elimination and backward substitution were conformed themselves to the Skyline method, so that the CPU time for solving a problem was reduced by half. Further, in real eigenvalue extraction we have improved the CPU performance of the Inverse Power method and added the Jennings method to NASTRAN. The Jennings method is more effective in many cases than the new Inverse Power method.

REFERENCES

1. McCormick, C. W.: Sparse Matrix Operations in NASTRAN. Proc. of 1973 Tokyo Seminar on FEA, 1973, pp. 611-631.

2. McCormick, C. W.: The NASTRAN Program for Structural Analysis. Proc. 2nd U.S.-Japan Seminar, 1972, pp. 551-571.

3. McCormick, C. W.: Application of Partially Banded Matrix Method to Structural Analysis, Sparse Matrix Proceedings, IBM T. Watson Research Center, 1968, pp. 155-158.

4. E. L. Wilson and H. H. Dovey: Solution or Reduction of Equilibrium Equations for Large Complex Structural Systems. Lecture Notes Part 1, SAP Conference, Tokyo, 1978, pp. 13-24.

5. E. L. Wilson: An eigensolution strategy for the Dynamic Analysis of Large Structural Systems. Lecture Notes Part 2, SAP Conference, Tokyo, 1978, pp. 1-19.

6. NASTRAN User's Manual/ Theoretical Manual.

7. Melosh, R. J. and Bamford, R. M.: Efficient Solution of Load-Deflection Equations. Proc. American Society of Civil Engineers, 95, ST4, 1969, pp. 661-676.

8. Irons, B. M.: A Frontal Solution Program for Finite Element Analysis Int. J. Num. Meth. Engng., vol 2, 1970, pp. 5-32.

9. K. J. Bathe and E. L. Wilson: Numerical Method in Finite Element Analysis. Prentice-Hall. Inc., 1976.

10. K. J. Bathe: Solution for Eigenvalue Problems in Structural Mechanics. Int. J. Num. Meth. Engng., vol 6, 1973, pp 213-226.

11. Jennings, A.: A Direct Method for the Solution of Large Sparse Symmetric Simultaneous Equations. Large Sparse Set of Linear Equations, REID, Academic Press, 1971, pp. 97-104.

12. H. Rutishauser: Computational Aspects of F. L. Brauer's Simultaneous Iteration Method. 1969

13. Jennings, A. and D. R. L. Orr: Application of the Simultaneous Iteration Method to Undamped Vibration Problems. Int. J. Num. Meth. Engng., vol 3, 1971, pp. 13-24.

14. Gupta, K. K.: Recent Advances in Numerical Analysis of Structural Eigenvalue problems. Proc. of 1973 Tokyo Seminar on FEA, Univ. of Tokyo Press, 1973, pp. 249-271.

15. Y. Yamamoto and H. Ohtsubo: Subspace Iteration Accelerated by the Use of Chebyshev Polynomials for Eigenvalue Problems with symmetric Matrices. Int. J. Num. Meth. Engng., vol 10, 1976, pp. 935-944.

16. James L. Rogers, Jr.: The Impact of Fourth Generation Computers on NASTRAN. NASTRAN User's Experiences, NASA TM X-3428, 1976, pp. 431-447.

17. Control Data Corporation: Study of the Modification Needed for Efficient Operation of NASTRAN on the Control Data Corporation STAR-100 Computer. NASA CR-132644, 1975

18. Universal Analytics, Inc.: Feasibility Study for the Implementation of NASTRAN on the ILLIAC IV Parallel Processor. NASA CR-132702 1975.

TABLE I.- GROUP CONTROL INFORMATION

Group	K(1)	K(2)	K(3)	K(4)	M(J)					
Group 1	1	1	1	6	1	3	6	8	12	15
Group 2	2	1	7	9	4	7	14			
Group 3	3	2	10	11	4	8				
Group 4	4	1	12	12	11					

K(1)	:	Current group number
K(2)	:	Minimum group number needed by calculation of group K(1)
K(3)	:	Minimum column number in group K(1)
K(4)	:	Maximum column number in group K(1)
M(J)	:	Pointer array of the diagonal term in the J-th column

TABLE II. MATRIX CHARACTERISTICS OF EXAMPLE PROBLEMS

Data name	Nodes	Elements	Total degree of freedom	Band matrix method			Skyline method					Memory size(kB) used
				B	C	R	Divisions of group	Group operation counts	Group read counts	Total columns read	Average length of column	
DATA1	394	352	2082	84	253	83	9	38	36	8328	176	1024
DATA2	489	477	2127	151	47	150	7	13	7	2127	128	1024
DATA3	611	2435	2769	93	288	92	13	66	63	13419	201	1024
DATA4	799	892	4474	156	0	155	18	35	18	4474	147	1024

TABLE III. CPU TIME: OLD, NEW INVERSE POWER METHODS AND JENNINGS METHOD

Data name	Total degree of freedom	Extraction method	Vector processor	Number of eigenvalues to be extracted	CPU time (sec)	
					Total	Eigenvalue extraction
Data 1	2082	Old Inverse Power method	Can not be applied	10	1864	1813
Data 1	2082	New Inverse Power method	Yes	10	344	293
Data 1	2082	Jennings method	Yes	10	241	191
Data 2	2127	Old Inverse Power method	Can not be applied	10	1105	1062
Data 2	2127	New Inverse Power method	No	10	398	345
Data 2	2127	New Inverse Power method	Yes	10	332	278
Data 2	2127	Jennings method	No	10	276	232
Data 2	2127	Jennings method	Yes	10	266	222

FIGURE I. MATRIX PACKING FORMAT

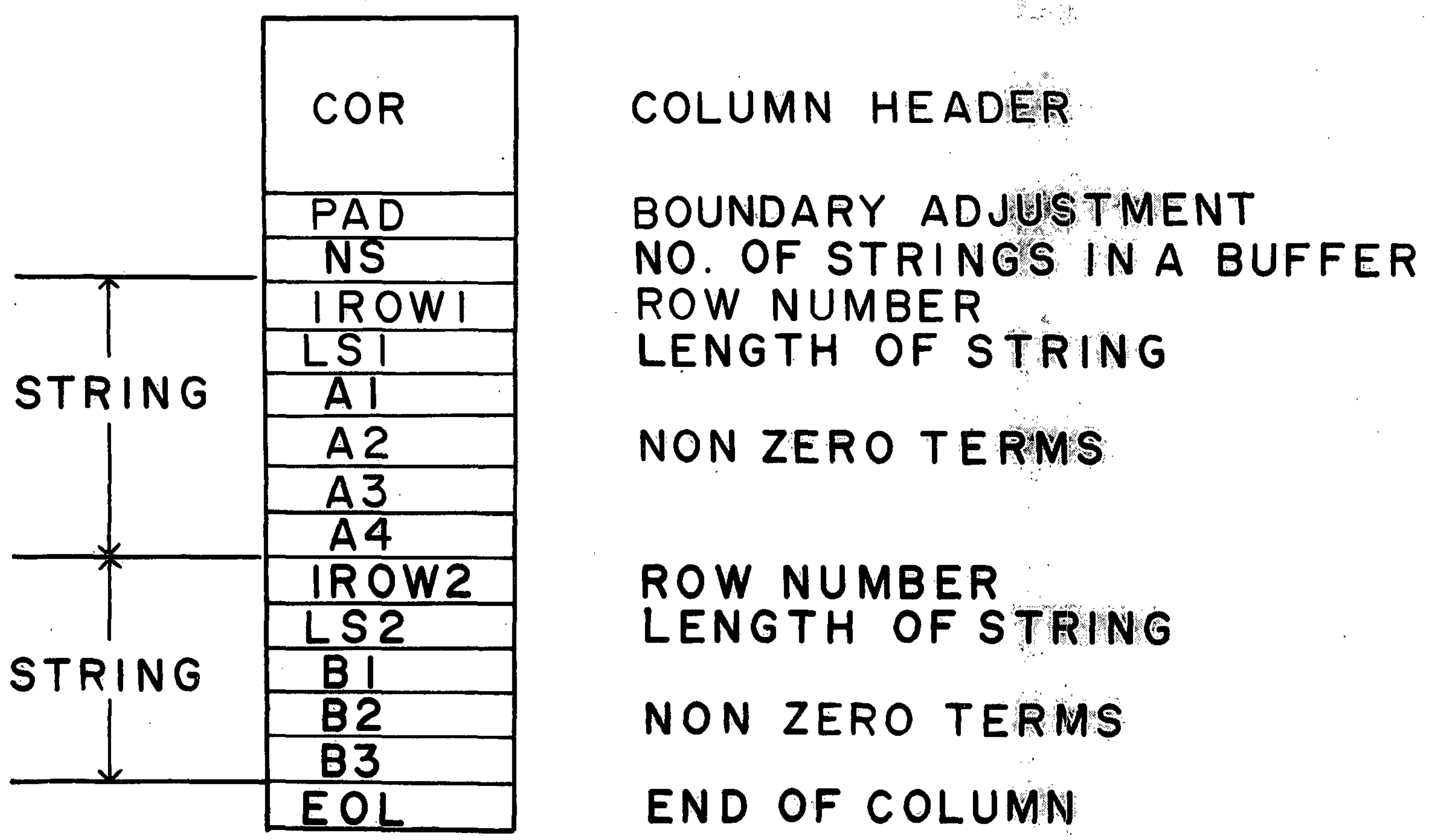


FIGURE 2. CONVENTIONAL BACKSPACING

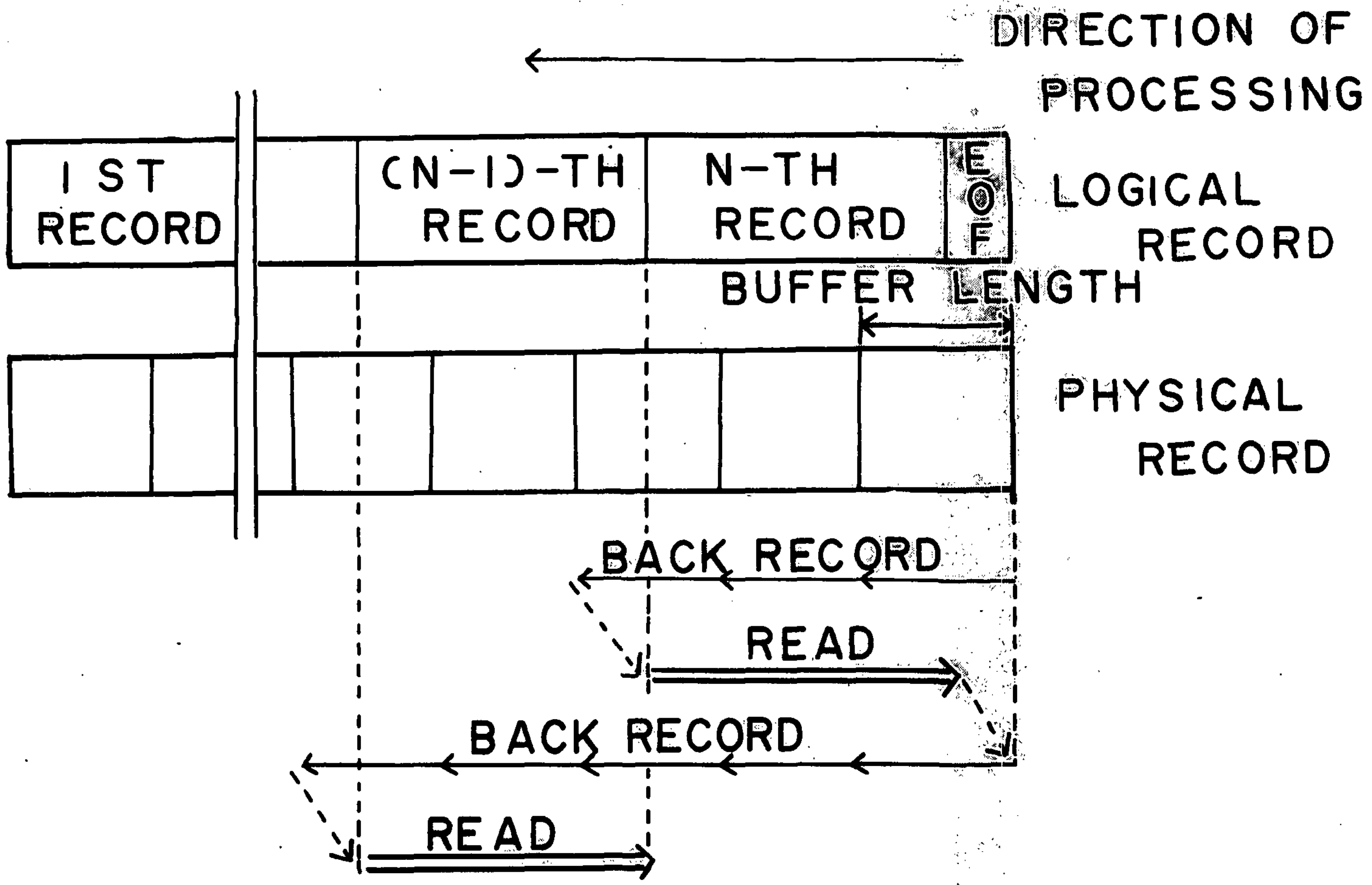


FIGURE 3. BUFFER-BY-BUFFER BACKSPACING

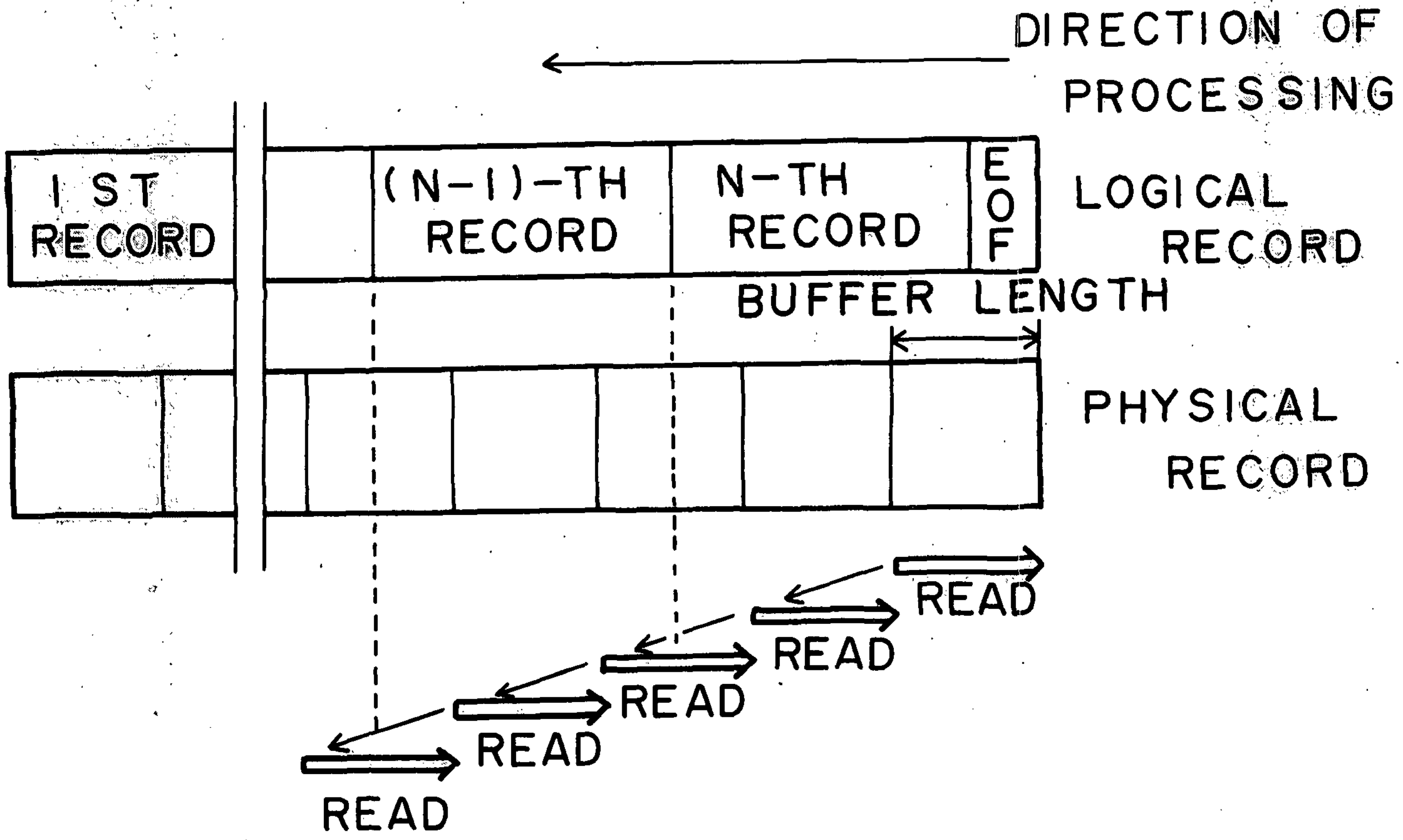


FIGURE 4. CPU TIME FOR PACKING/UNPACKING (2000-DIMENSION VECTOR: M-200H)

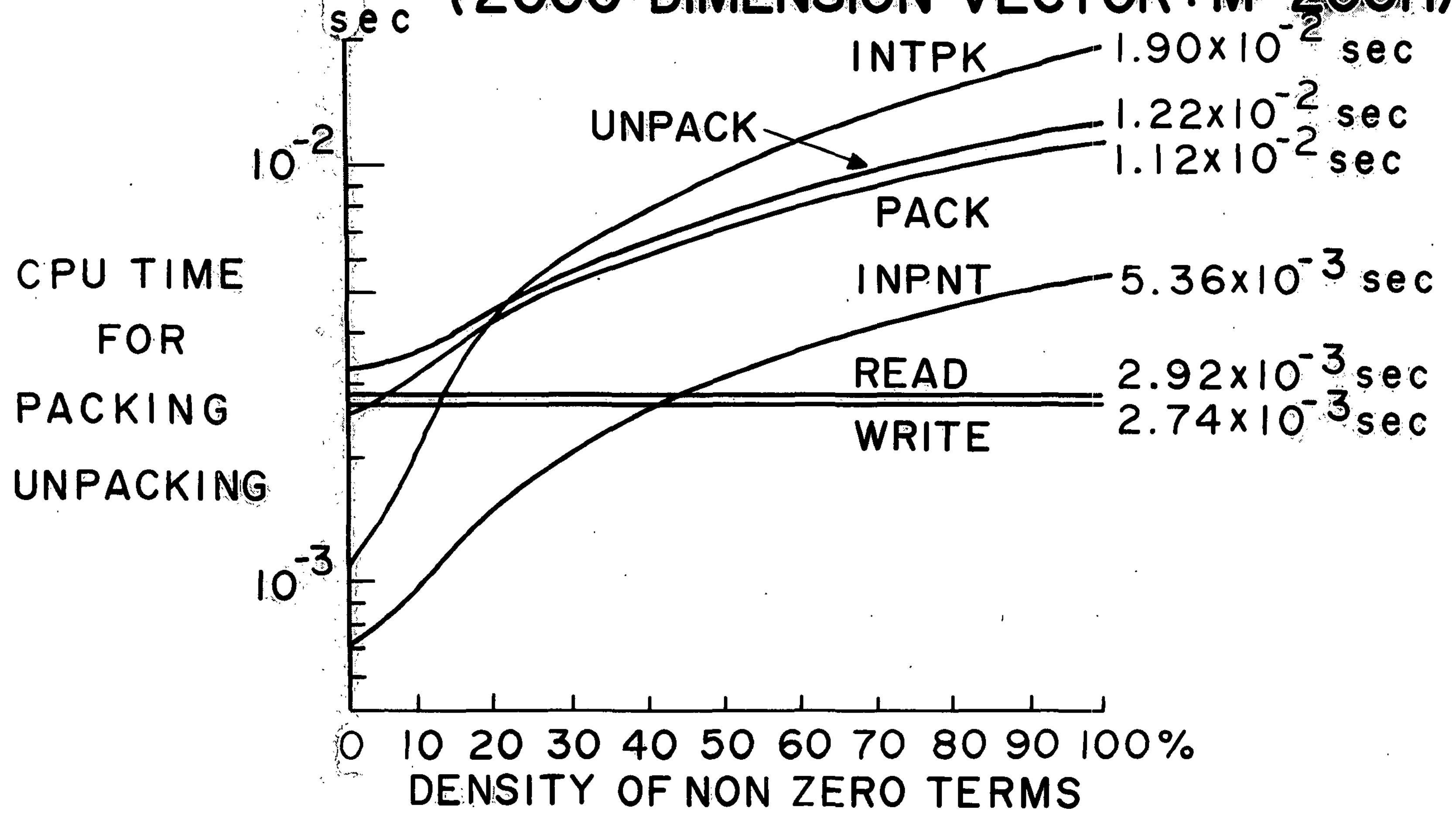


FIGURE 5. MATRIX DECOMPOSITION

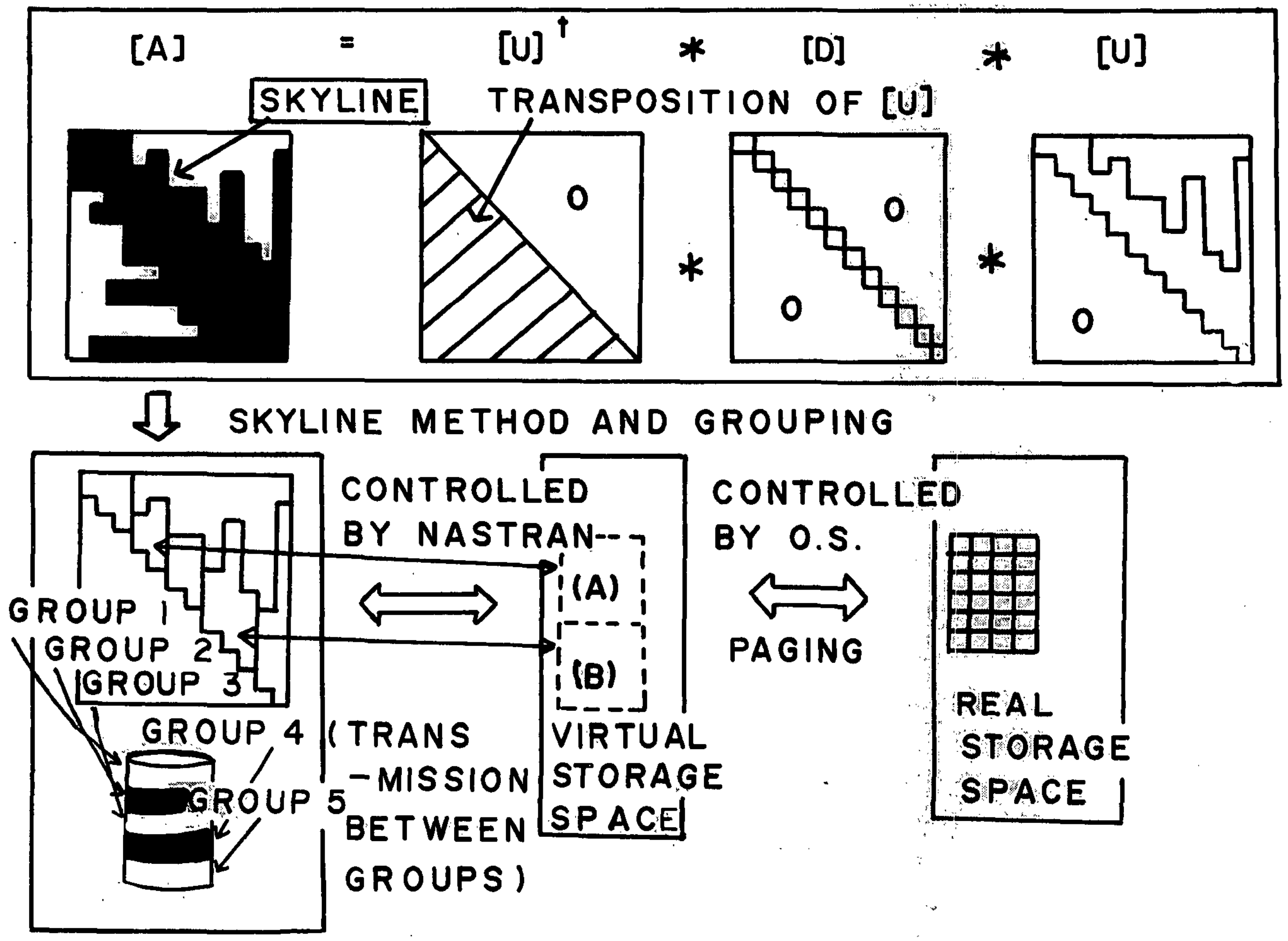


FIGURE 6. ALGORITHM FOR THE SKYLINE METHOD (INCORE TYPE)

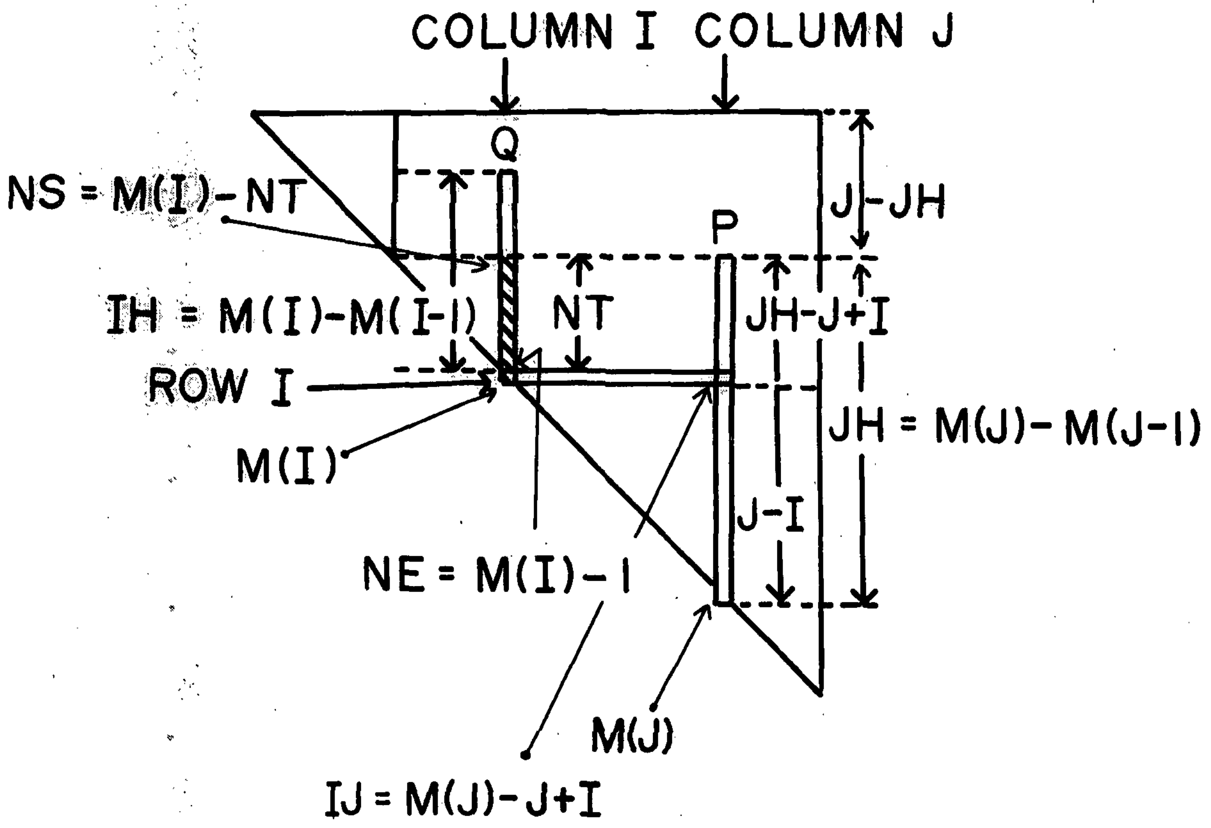


FIGURE 7. MATRIX GROUPING

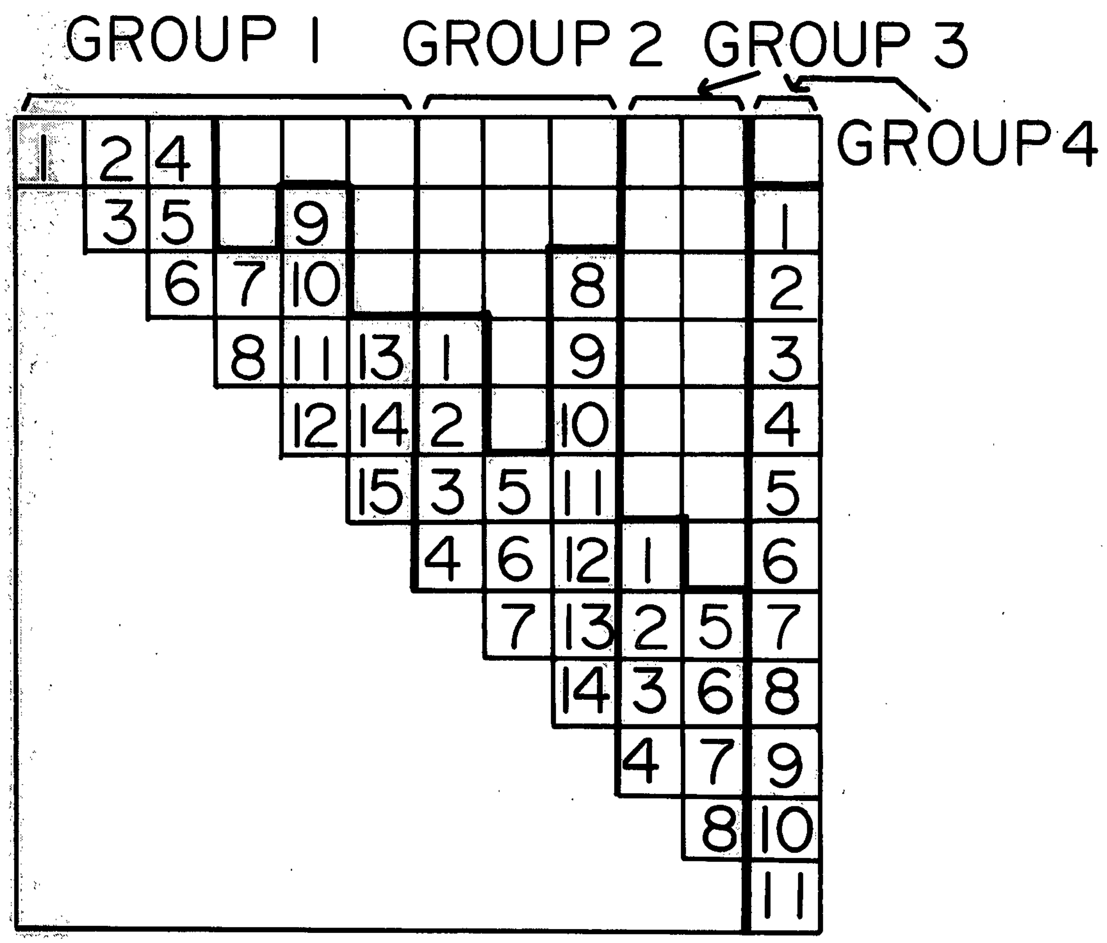


FIGURE 8. SKYLINE METHOD WITH GROUPING

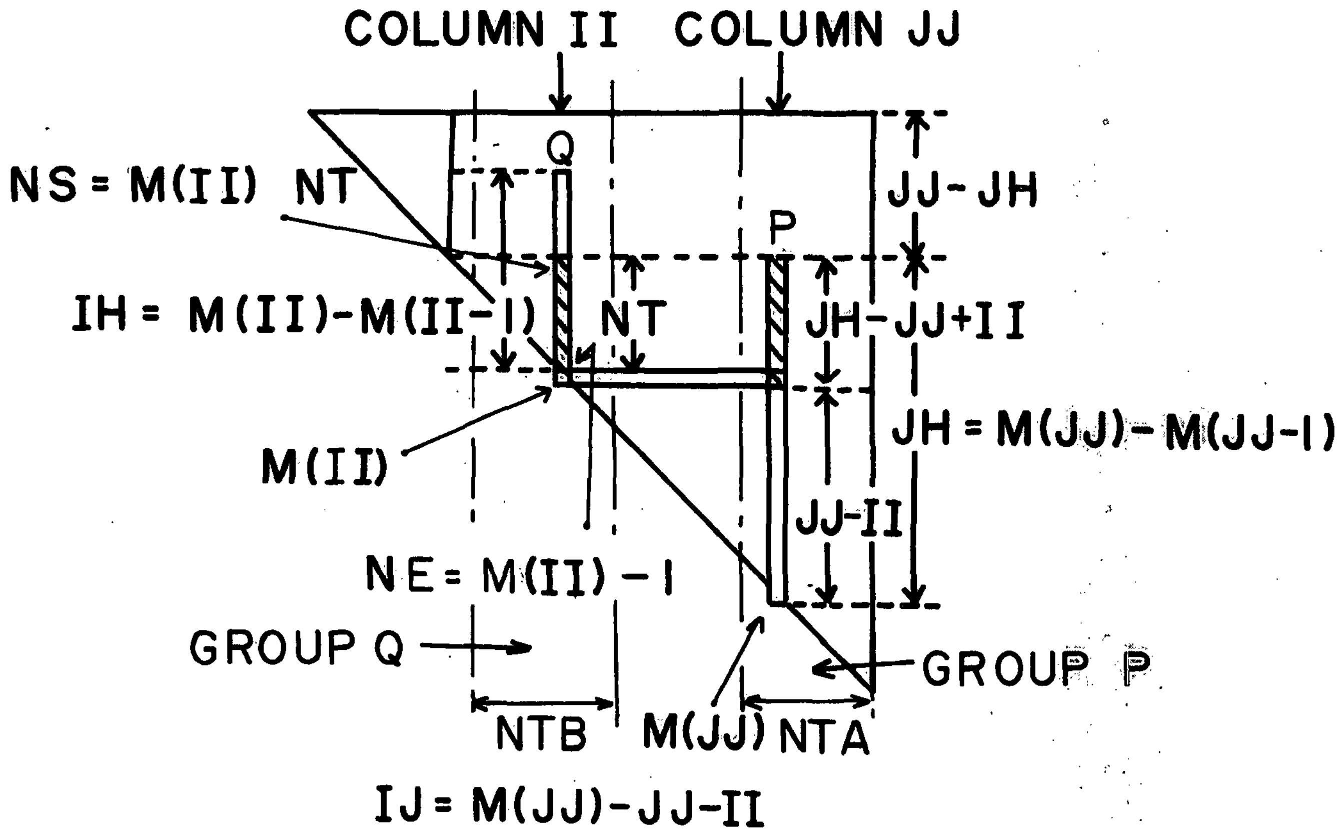


FIGURE 9. OPEN CORE LAYOUT IN SKYLINE METHOD

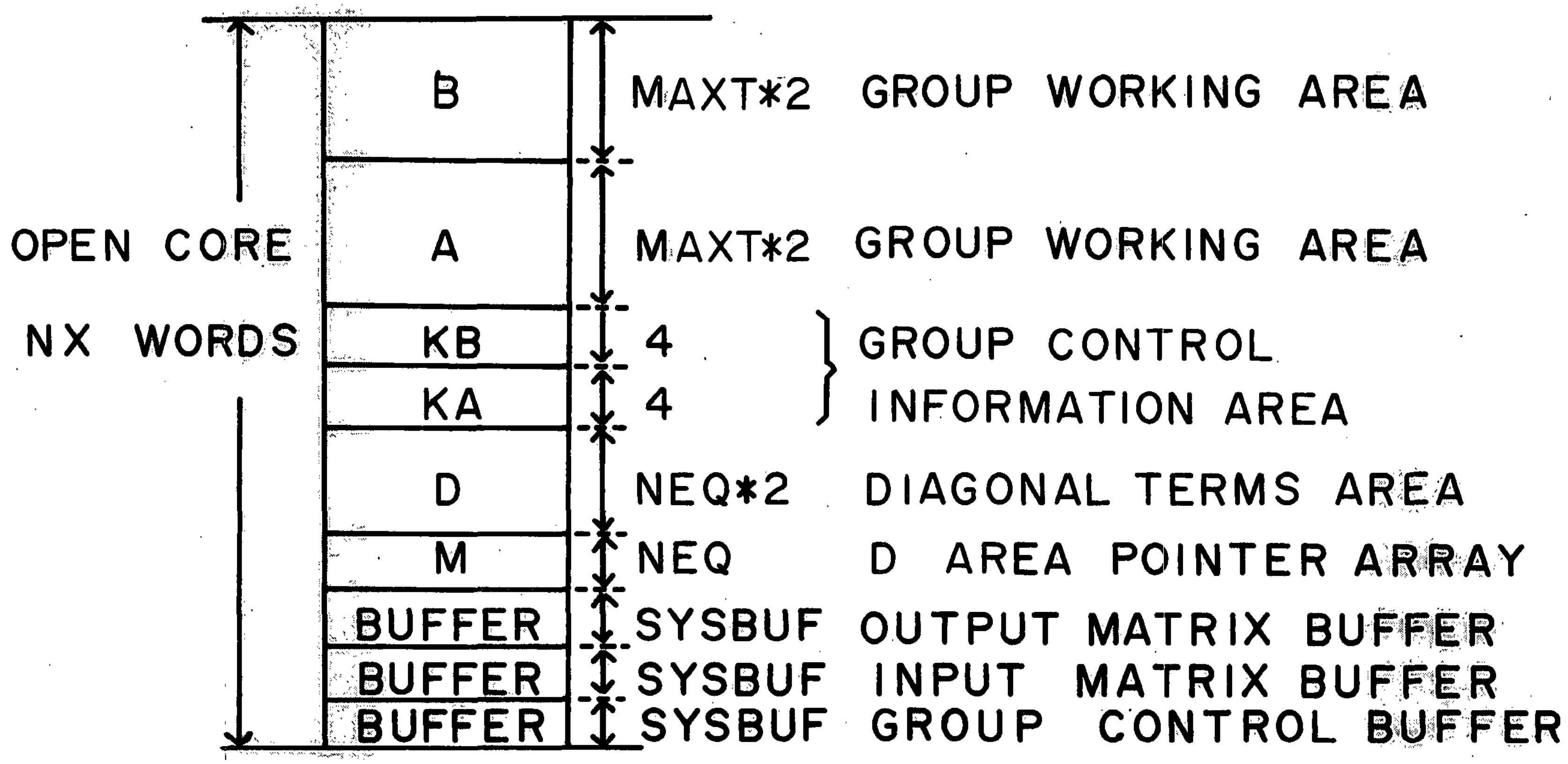

OPEN CORE
NX WORDS
B
MAXT*2 GROUP WORKING AREA
A
MAXT*2 GROUP WORKING AREA
KB
4
KA
4
GROUP CONTROL INFORMATION AREA
D
NEQ*2 DIAGONAL TERMS AREA
M
NEQ D AREA POINTER ARRAY
BUFFER
SYSBUF OUTPUT MATRIX BUFFER
BUFFER
SYSBUF INPUT MATRIX BUFFER
BUFFER
SYSBUF GROUP CONTROL BUFFER

FIGURE 10. CPU TIME: SKYLINE METHOD AND BAND MATRIX METHOD

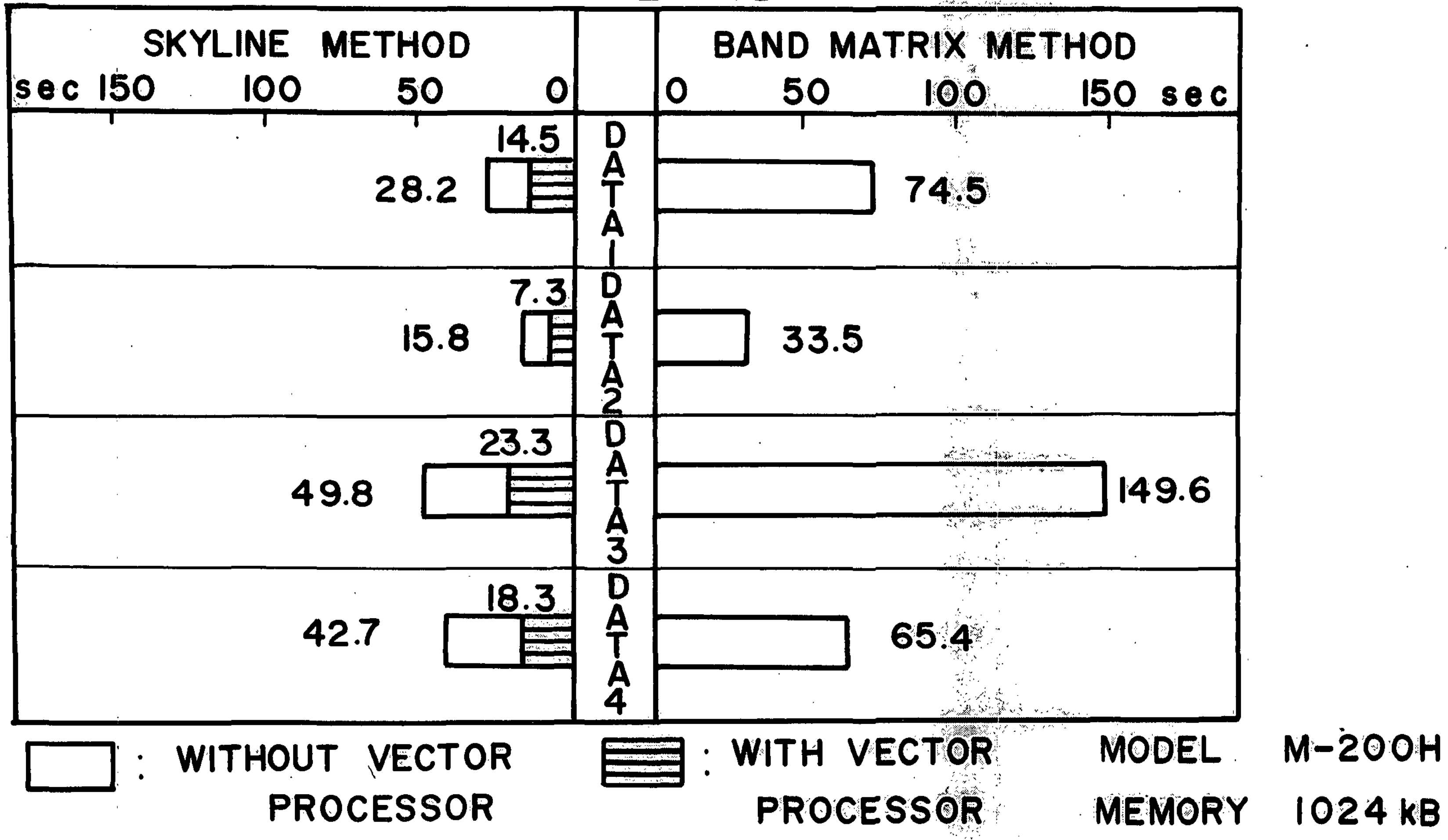


FIGURE 11. FORWARD ELIMINATION PROCESS

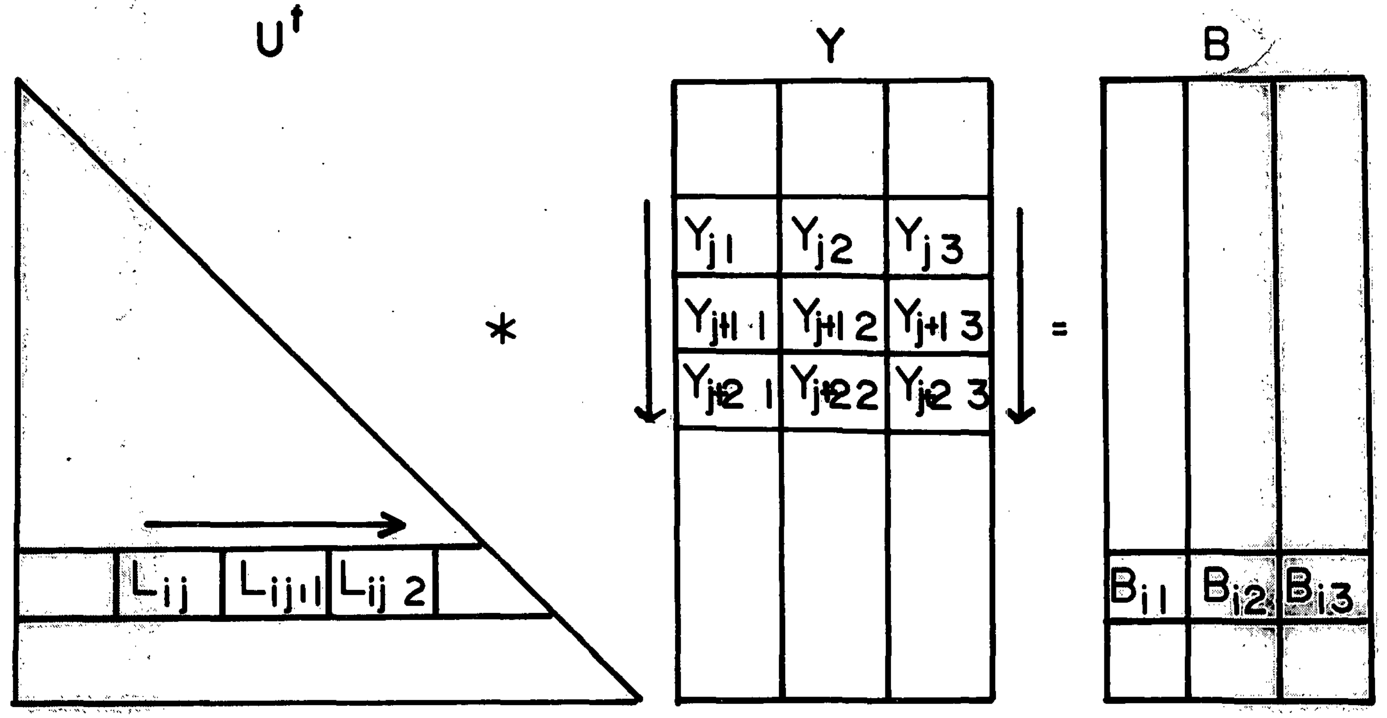


FIGURE 12. BACKWARD SUBSTITUTION PROCESS

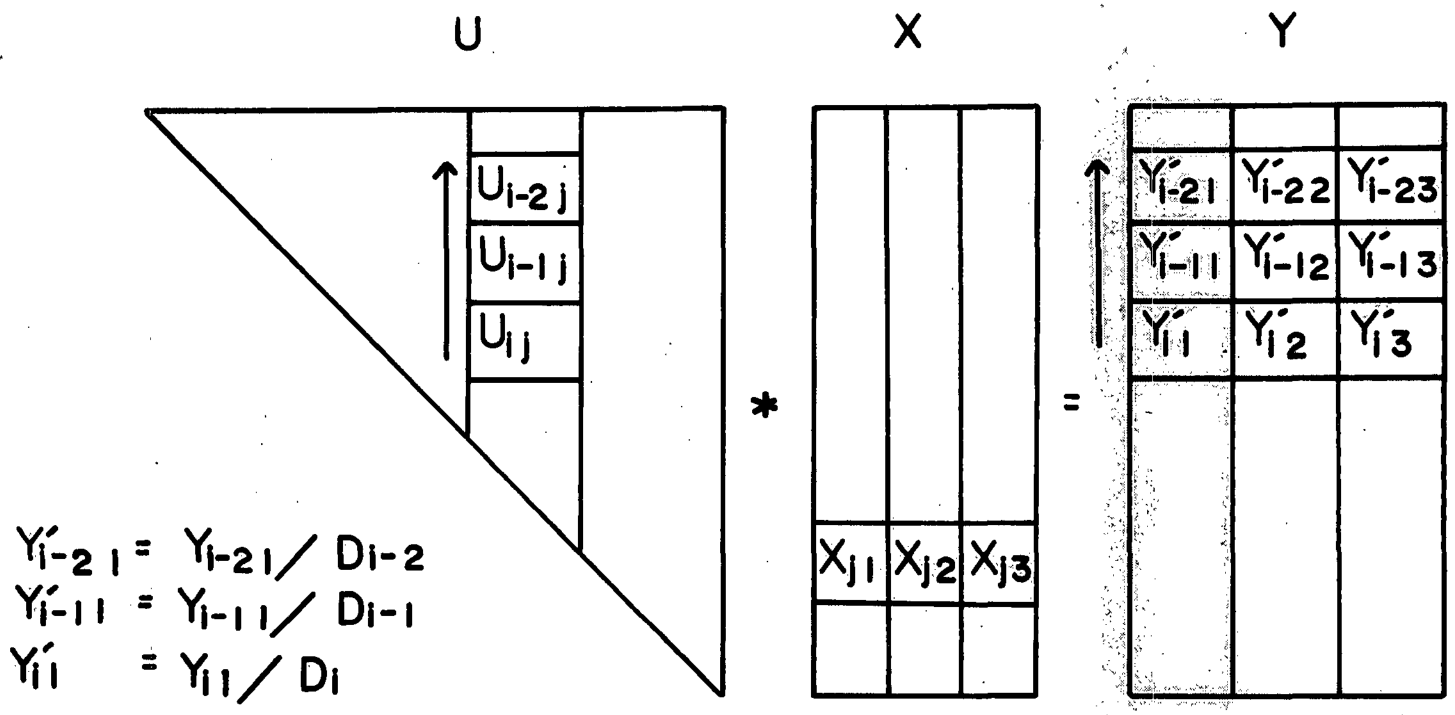
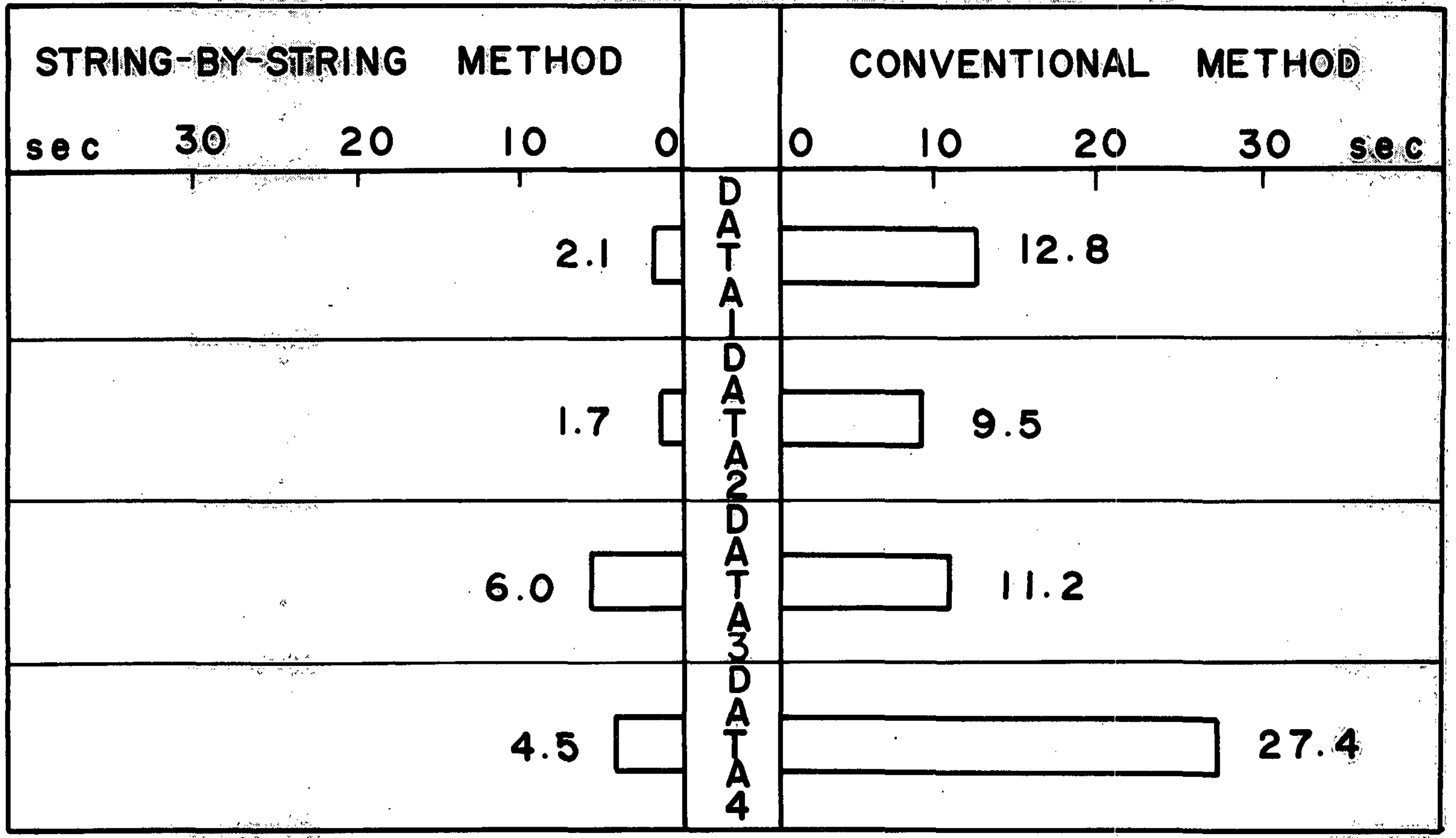


FIGURE 13. CPU TIME: NEW STRING-BY-STRING METHOD AND CONVENTIONAL FBS

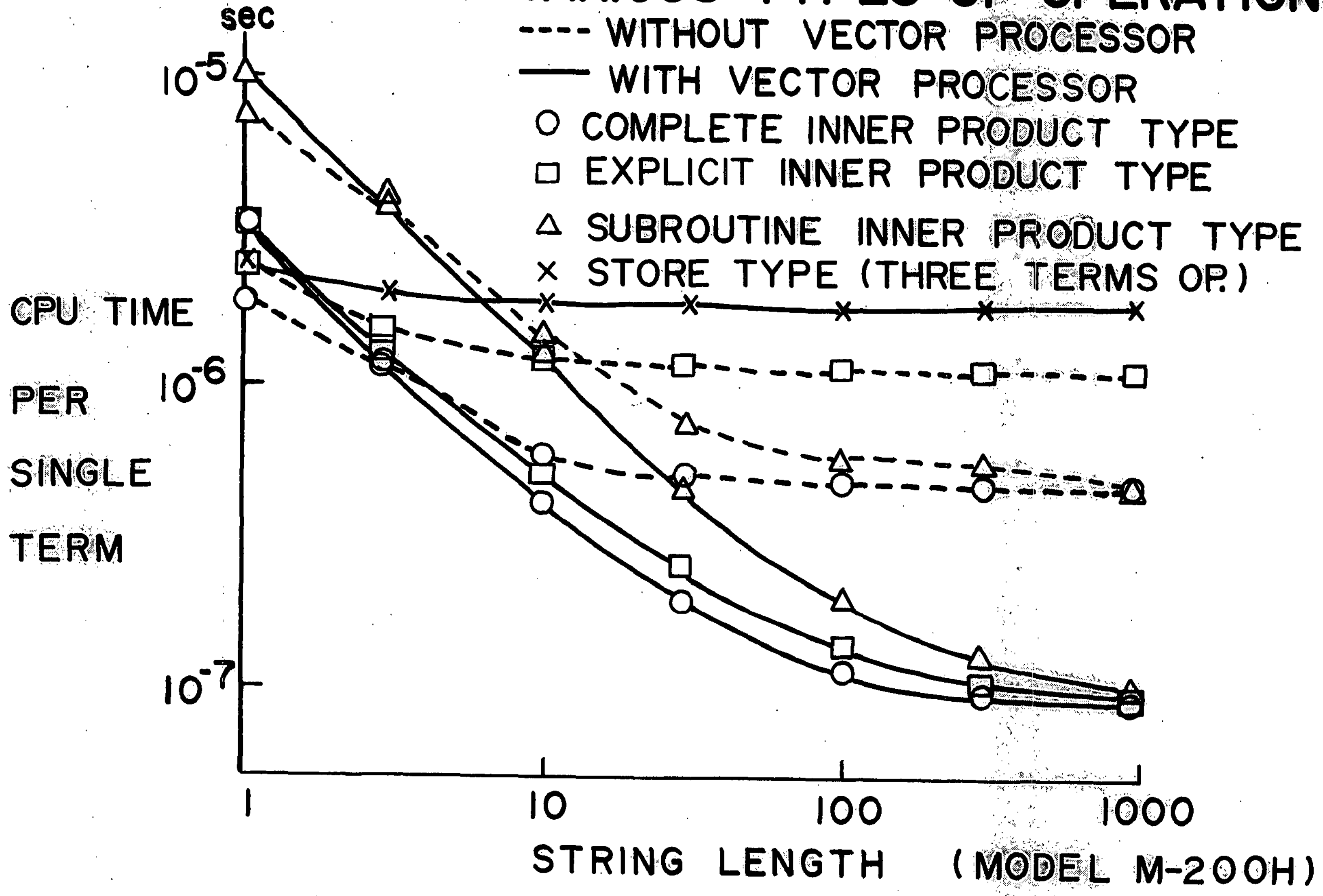


MEMORY 1024kB MODEL M-200H

FIGURE 14. ALGORITHM FOR JENNINGS METHOD

```
(a) DETERMINATE THE DIMENSION OF SUBSPACE   $m = \min\{n, 2q, q+8\}$
(b) CHOOSE INITIAL VECTORS   $[G_0]\ (n \times m)$   AND   $[G] = [G_0]$
(c) $[K][Z] = [G]$   SOLVE   $[Z]\ (n \times m)$
(d) $[\overline{K}] = [Z]^{\dagger}[G]$        CALCULATE   $[\overline{K}]$ , $[\overline{M}]$
(e) $[\overline{M}] = [Z]^{\dagger}[M][Z]$
(f) $[\overline{K}][Q] = [R][\overline{M}][Q]$   SOLVE EIGENVALUE PROBLEM ON SUBSPACE
(g) SORT EIGENVALUES IN ASCENDING ORDER AND GO TO (m) IF
    q EIGENVALUES FROM THE LOWEST ARE FOUND
(h) LET $[P]\ (m \times m)$ BE A MATRIX COMPOSED OF EIGENVECTORS
    CORRESPONDING TO SORTED EIGENVALUES
(i) $[G'] = [M][Z][P]$
(j) $[S] = [G']^{\dagger}[G']$
(k) $[S] = [U]^{\dagger}[U]$   $U^{\dagger}U$-DECOMPOSITION
(l) $[G] = [G'][U]^{-1}$   → GO TO (c)
(m) WRITE OUT EIGENVALUES AND VECTORS TO OUTPUT FILE
```

FIGURE 15. CPU TIME VERSUS STRING LENGTH IN VARIOUS TYPES OF OPERATIONS


sec
10^{-5}
10^{-6}
10^{-7}
CPU TIME PER SINGLE TERM
---- WITHOUT VECTOR PROCESSOR
—— WITH VECTOR PROCESSOR
○ COMPLETE INNER PRODUCT TYPE
□ EXPLICIT INNER PRODUCT TYPE
△ SUBROUTINE INNER PRODUCT TYPE
× STORE TYPE (THREE TERMS OP.)
STRING LENGTH (MODEL M-200H)