A SIMPLE AND EFFICIENT MEMORY OPTIMIZATION TECHNIQUE FOR SOLUTION OF LARGE MATRICES

B. H. BRIZ-KISHORE AND R. V. S. S. AVADHANULU*
Centre of Exploration Geophysics, Osmania University, Hyderabad 500 007, India.
*Computer Division, Administrative Staff College of India, Hyderabad 500 475, India.

Abstract

Large matrix problems pose extensive memory requirements even by decomposition methods. An optimized memory utilization procedure is developed. The concept lies in keeping only the required elements in the core at any level of computations with small auxilliary support. A program is designed and developed on EC-1030 computer, incorporating the present technique for solving the matrices in flow problems of dimensions 30 × 21 with 360 bytes as against 7,560 bytes by conventional methods.

Introduction

Several decomposition methods are available involving large matrices[1] for solving a system of simultaneous equations of the type $AX = F$, in which A is the coefficient matrix and X and F are solution and constant vectors, respectively. The decomposition methods involve in splitting the coefficient matrix into a lower (L) and an upper (U) triangular matrices such that $A = LU$. While handling large matrix problems, intermediate arrays generated during the process of decomposition occupy extensive memory thus making impossible to solve even with 128K bytes. Hence a new core buffer method has been introduced utilizing the recursive procedure for considerable optimization of memory requirements.

A brief mathematical treatment, the concept of core buffer method and an evaluation of the method are given in the following sections.

Mathematical Theory

Let us consider the elements of an intermediate array a at any three rows, say $i-1$, i and $i+1$, as represented in figure 1. As can be seen from the figure, the array element $a_{i,j}$ at ith row and jth column is surrounded on the four sides by the four elements $a_{i-1,j}$, $a_{i,j-1}$, $a_{i,j+1}$ and $a_{i+1,j}$. The generation mechanism of the elements is such that the computation of any element $a_{i,j}$ of the current row utilizes only two

$$\begin{array}{ccc} a_{i-1,j-1} & a_{i-1,j} & a_{i-1,j+1} \\ a_{i,j-1} & a_{i,j} & a_{i,j+1} \\ a_{i+1,j-1} & a_{i+1,j} & a_{i+1,j+1} \end{array}$$

Figure 1. Elements of an intermediate array at $(i-1)$, i and $(i+1)$th rows.

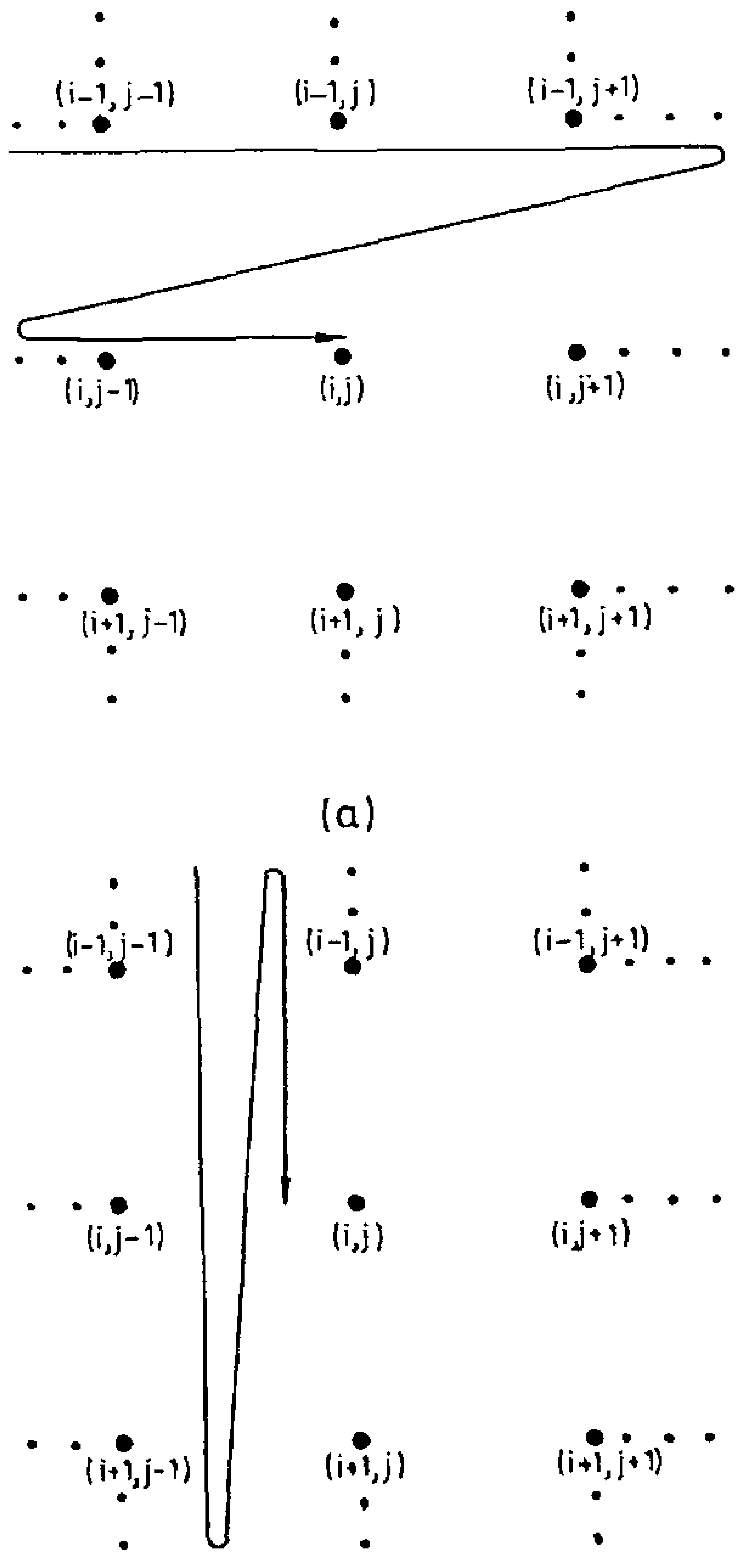
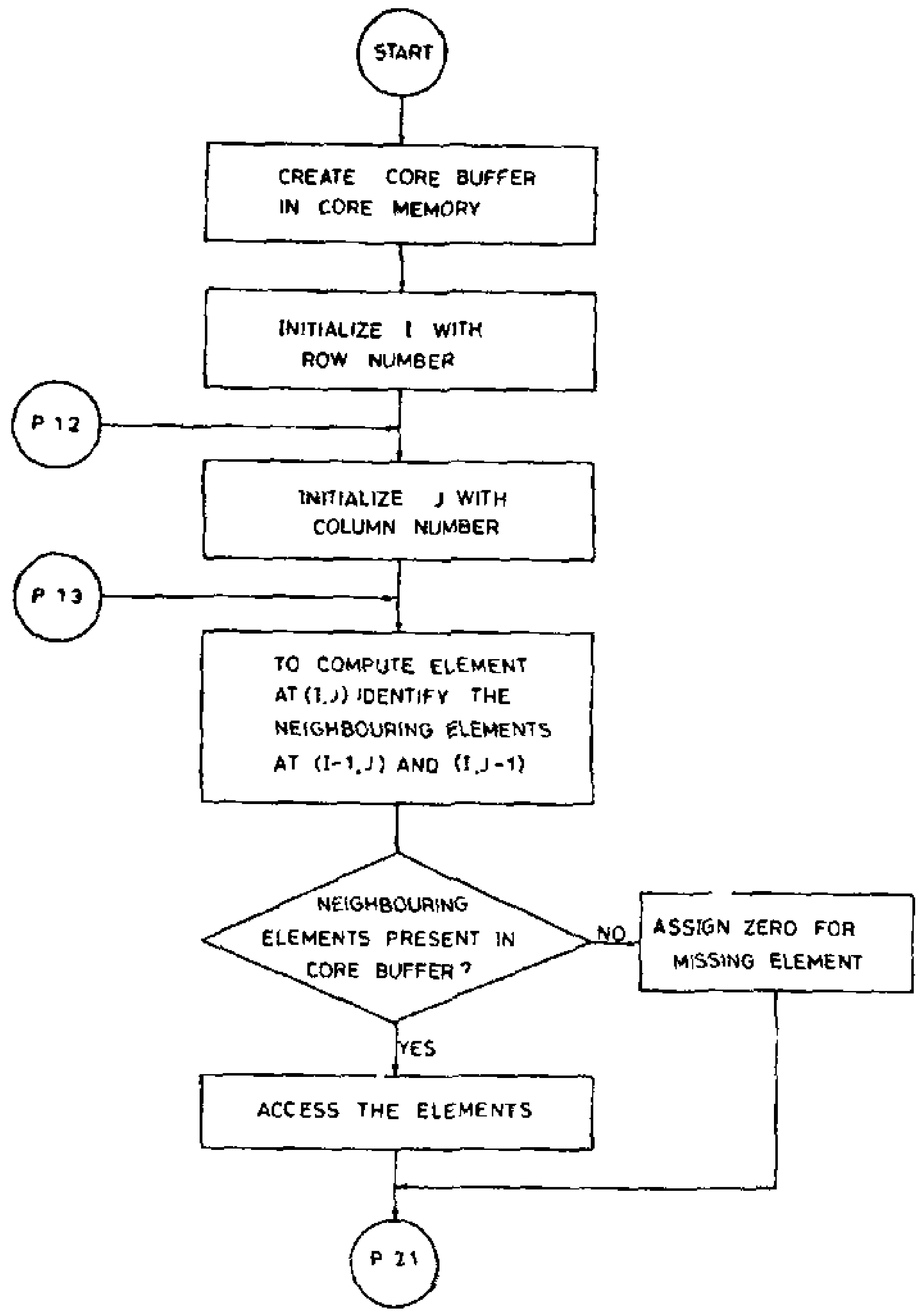


Figure 2. Accessing mechanism of array elements. **a.** Row-wise. **b.** Column-wise.

values *i.e.*, $a_{i-1,j}$, the corresponding element of the previous row (also referred to as top element) and $a_{i,j-1}$, the immediately preceding element of the current row (left side element). It can be seen from figure 2a that while computing $a_{i,j}$, the top element $a_{i-1,j}$, is used before the left side element $a_{i,j-1}$ and thereafter no reference is made to the top element. A similar procedure is seen for column-wise processing (figure 2b) also. This indicates that an element is required for a brief period during its related computations only and can get replaced by the next one for the subsequent computations. This principle of non-requirement of all the four elements in any computation of a single element is conveniently utilized in core buffer method for reduction of memory requirements of the intermediate arrays.

Core Buffer Method

This method involves in the creation of a buffer with a set of locations in the core memory to contain elements of a row of the intermediate array (NC); the execution sequence of the developed code is presented in figures 3(a) and 3(b). The buffer is initialized with the first row elements (figure 4a) as defined by the boundary conditions, representing the elements (1, 1), (1, 2), (1, 3)..... When the computational process


START
CREATE CORE BUFFER IN CORE MEMORY
INITIALIZE I WITH ROW NUMBER
P 12
INITIALIZE J WITH COLUMN NUMBER
P 13
TO COMPUTE ELEMENT AT (I,J) IDENTIFY THE NEIGHBOURING ELEMENTS AT (I-1,J) AND (I,J-1)
NEIGHBOURING ELEMENTS PRESENT IN CORE BUFFER ?
NO
ASSIGN ZERO FOR MISSING ELEMENT
YES
ACCESS THE ELEMENTS
P 21

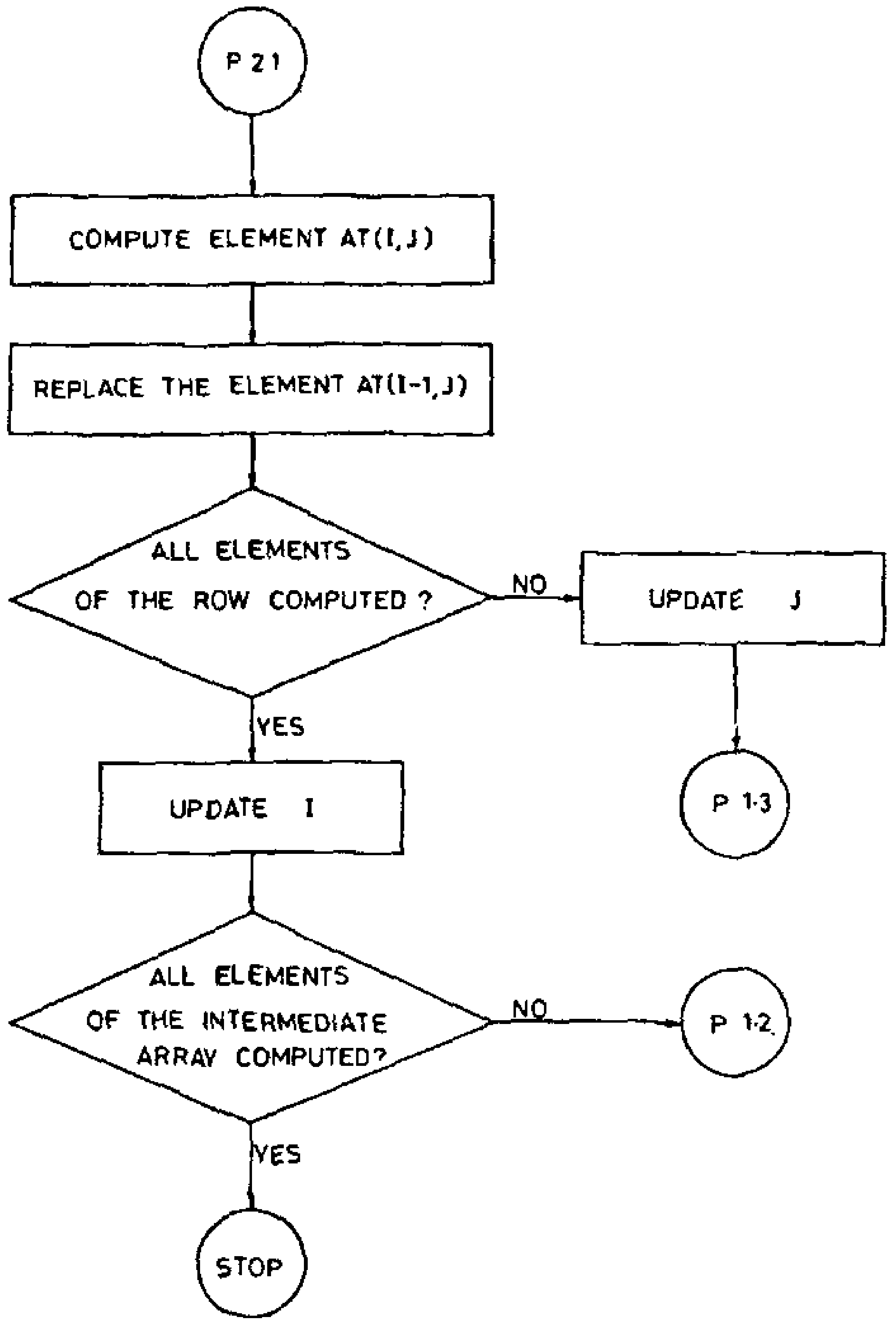

P 21
COMPUTE ELEMENT AT(I,J)
REPLACE THE ELEMENT AT(I-1,J)
ALL ELEMENTS OF THE ROW COMPUTED ?
NO
UPDATE J
P 1·3
YES
UPDATE I
ALL ELEMENTS OF THE INTERMEDIATE ARRAY COMPUTED?
NO
P 1·2
YES
STOP

Figure 3. a, b. Execution sequence in core buffer method.

commences for the element (2, 1), the procedure requires the values of (1, 1) and (2, 0) elements. As (2, 0) is an element outside the boundary its value is taken as zero and hence the element (1, 1) only is utilized to compute (2, 1). Since the element (1, 1) is not required subsequently, that element is evicted from the core buffer and its position is allotted for the element (2, 1) that is acquiring a new value. Thus the core buffer is now with the elements (2, 1), (1, 2), (1, 3).... as shown in figure 4b.

The generation mechanism further proceeds to compute the value of the next element (2, 2) which requires the values of (1, 2) and (2, 1). As can be seen from figure 4b, these two values are available in the same core buffer and hence are immediately accessed to compute (2, 2). This new value of (2, 2) replaces the element (1, 2) and updates the core buffer (figure 4c) to represent (2, 1), (2, 2), (1, 3).... This process is repeated from left to right by incrementing the column indicator j to compute any jth element of the current ith row and replaces the value of $(i-1, j)$th element.

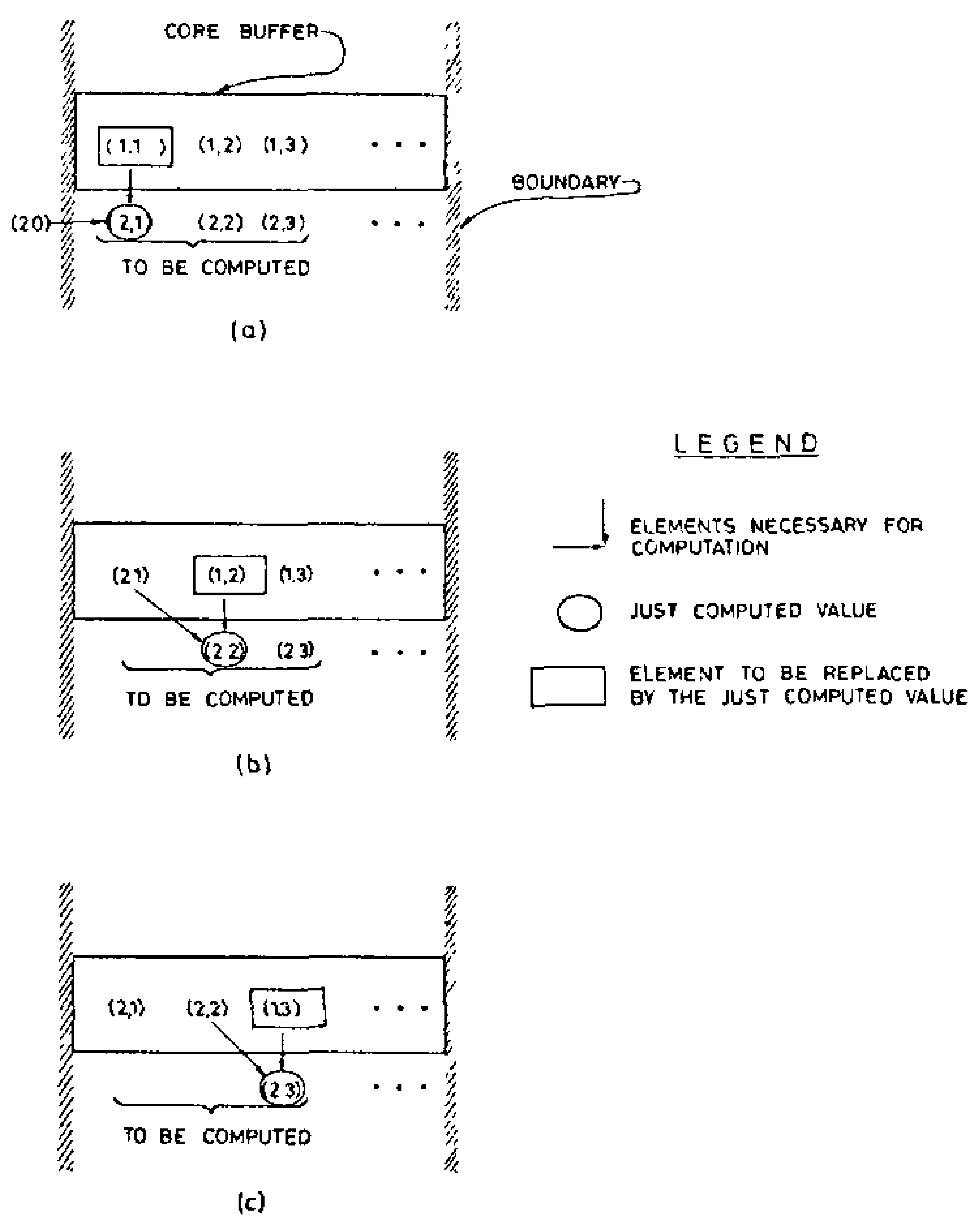


Figure 4. Contents of core buffer. **a.** After initialization with first row elements. **b, c.** After computation of (2, 1) and (2,2).

At the end of each row, the contents of the core buffer are saved on the disc to form a record of elements for subsequent computations. Similar steps are adopted for all the rows (NR) and the corresponding records are created on the disc file.

This technique is utilized for solving a flow problem of dimensions 30 × 21, involving 630 nodes and three intermediate arrays. The problem could be solved using 360 bytes of memory on EC-1030 computer by the presently developed technique as against 7560 bytes in the conventional methods, thereby reducing about 97% of memory requirement. However, the time requirement in both the methods is same and to be less than two minutes, since the number of accesses to the core has not changed in the core buffer method.

Conclusions

The conventional decomposition techniques also require extensive memory for several intermediate arrays in solving large matrices. The storing of non-required elements is replaced by the new concept of core buffer method which rendered possible the solution of large matrices on microcomputers with minimum configuration.

1. Remson, I., Hornberger, G. M. and Molz, F. J., *Numerical methods in subsurface hydrology*, (New York: Wiley-interscience), 1971.

ORGANOTIN COMPLEX COMPOUNDS OF N-SUBSTITUTED BENZOHYDROXAMIC ACIDS (2)

B. PRADHAN* AND A. K. GHOSH
Department of Chemistry, North Bengal University, Darjeeling 734 430, India.
* Present address: Department of Chemistry, Raiganj University College, Raiganj 733 134, India.

Abstract

Some triphenyl tin derivatives of N-substituted benzohydroxamic acids have been prepared and the stability of these chelates tested. Also, structures of some tin derivatives of N-phenyl benzohydroxamic acid have been discussed.

Introduction

The authors[1] have reported three classes of new organotin compounds, R_2SnLX, $RSnL_2X$ and $RSnLX(OM_e)$ along with the preparation and properties of some R_2SnL_2, where LH is the hydroxamic acid ligand. In this communication, some triphenyl tin compounds of N-substituted hydroxamic acids have been reported. The tentative structures of $Bu_2Sn(PBHA)_2$, $PhSn(SCN)(PBHA)_2$, $Ph_2Sn(SCN)(PBHA)$, $Bu_2Sn(SCN)(PBHA)$ and P.M.R. spectra of $PhSnCl(OCH_3)(PBHA)$, where PBHA being N-phenyl benzohydroxamic acid also are being reported.