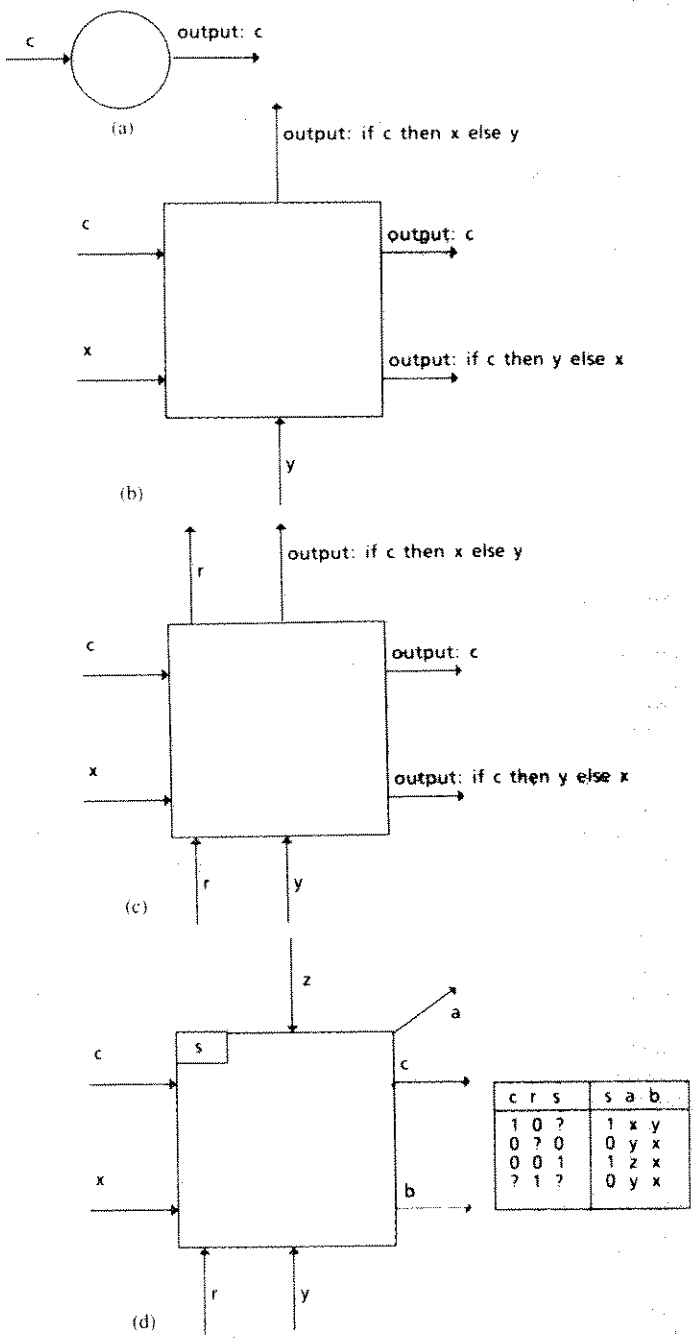

Fig. 1. Four systolic array elements. The outputs and new state definitions for the last one are indicated in the table.

Systolic Arrays for Matrix Transpose and Other Reorderings

DIANNE P. O'LEARY

***Abstract*—In this correspondence, a systolic array is described for computing the transpose of an $n \times n$ matrix in time $3n - 1$ using n^2 switching processors and n^2 bit buffers. A one-dimensional implementation is also described. Arrays are also given to take a matrix in by rows and put it out by diagonals, and vice versa.**

***Index Terms*—Matrix transpose, parallel computation, systolic array.**

I. Introduction

The task of computing the transpose of a matrix arises in many matrix algorithms, for example, in computing congruence transformations $B = UAU^T$ and in computing a step of the QR algorithm for finding eigenvalues of a matrix.

Systolic arrays, introduced by Kung and Leiserson [4], are a very popular implement for parallel matrix computation. However, there seems to be no consensus on the best way to transpose a matrix for use in a systolic array algorithm. For example, Bojanczyk, Brent, and Kung [2] suggest using "a buffer that supports fast two-dimensional addressing," and the speed of their algorithm depends critically on the speed of this hardware. Ullman [5] discusses a systolic algorithm of Atallah and Kosaraju [1] which transposes a matrix in place.

We present in this paper a rather simple systolic array which puts out the transpose of a matrix. The matrix is pumped in systolically, rather than being preloaded as in the above two methods. We also present two modifications of this array; one takes a matrix in by rows and puts it out by diagonals and the other reverses this process. A one-dimensional version of the matrix transpose array is also given. Alternate designs for the diagonal-to-row and row-to-diagonal conversions are given by Ipsen [3] who also presents an algorithm for transposition of a matrix in diagonal format.

To keep diagrams uncluttered, we adopt the convention that unlabeled inputs and outputs are either indeterminate or of no interest. The definitions of the four building blocks of the arrays are given in Fig. 1. The circles represent buffers which cause the control

Manuscript received April 5, 1985. This work was supported by the Air Force Office of Scientific Research under Grant AFOSR-82-0078.

The author is with the Department of Computer Science, University of Maryland; College Park, MD 20742.

IEEE Log Number 8611264.

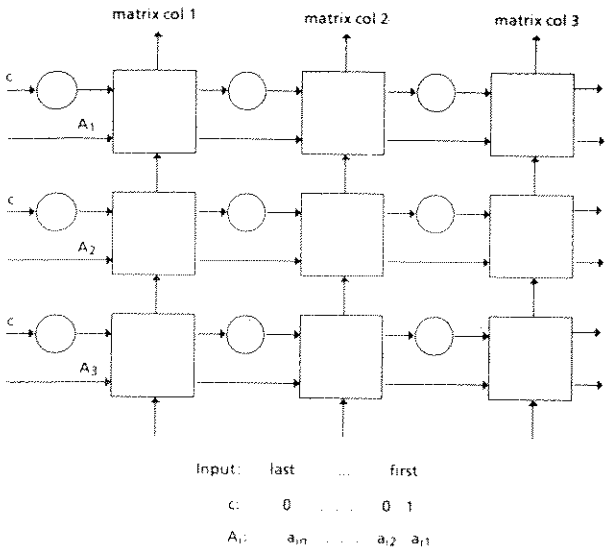

Fig. 2. A two-dimensional systolic array for matrix transpose ($n = 3$).

bits to travel at the proper velocity. They take a single bit as input and give a single bit as output. The squares are very simple switching processors. The first receives as input a control bit c from the left, and possible matrix elements x from the left and y from below. If c is zero, then x is sent to the right and y is sent up. If c is one, then y is sent to the right and x is sent up. The second square is similar to the first but has an extra single bit input and output called r. The third is a variant of this switch which includes one bit s of memory. Its operation is defined by the accompanying table in Fig. 1.

II. The Systolic Array for Matrix Transposition

The architecture for the first systolic array is shown in Fig. 2 for the case $n = 3$. The matrix is fed into the array from the left, one row per row of processors. A control bit sequence is also fed from the left in each row of processors; a 1 is sent one time unit before the beginning of the row, and $n - 1$ 0's follow. (The first column of bit buffers can be eliminated; then the control bit 1 would be synchronized with the beginning of the row, and one time unit would be saved.)

The transpose of the matrix exits from the top, one column per column of processors. A simulation for the 3×3 case in which all rows are started simultaneously is given in Fig. 3. Element (i, j) enters the array at time j and exits at time $2j + i - 1$, giving a total time of $3n - 1$ for transposing a matrix of size $n \times n$.

Elements in a given column come out at consecutive times, just as they were loaded. The array still works if later rows are loaded delayed relative to earlier ones, as long as the earlier sequences of control signals are padded by extra zeroes.

The array can be modified in case the elements in each row are not available at consecutive time units. For example, if elements are loaded as $(i, 1)$, *, *, $(i, 2)$, *, *, $\cdots$ (i, n) where * represents an indeterminate, then each bit buffer should be replaced by three such buffers.

III. A One-Dimensional Implementation of the Transposition Array

The array discussed in the previous section can be easily modified to use n processors and n bit buffers, but then the time is increased to $n^2 + n$, and elements in a given column exit at intervals of time n. The array is shown in Fig. 4 for the case $n = 3$. The processors are the same as before, although the switching processor can be simplified since the y input is always indeterminate. The matrix is entered by rows $(1, 1), (1,2), \cdots, (1, n), (2, 1), (2, 2), \cdots, (2, n), \cdots, (n, 1), (n, 2), \cdots, (n, n)$, and, as before, the control signal is 0 except at the beginning of a row. A simulation of the array is given in Fig. 5. Element (i, j) enters the array at time $(i - 1)n + j$ and exits j time units later, for a total time of $n^2 + n$.

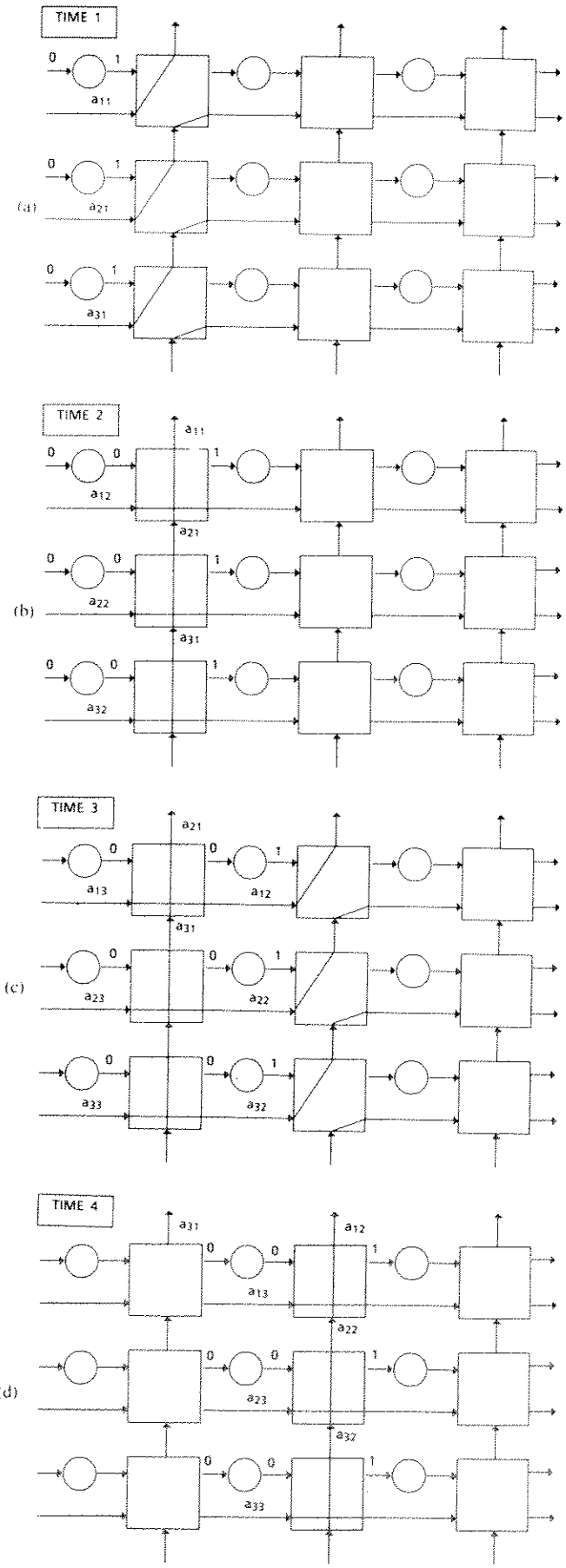

Fig. 3. Transpose of a 3×3 matrix using the two-dimensional array.

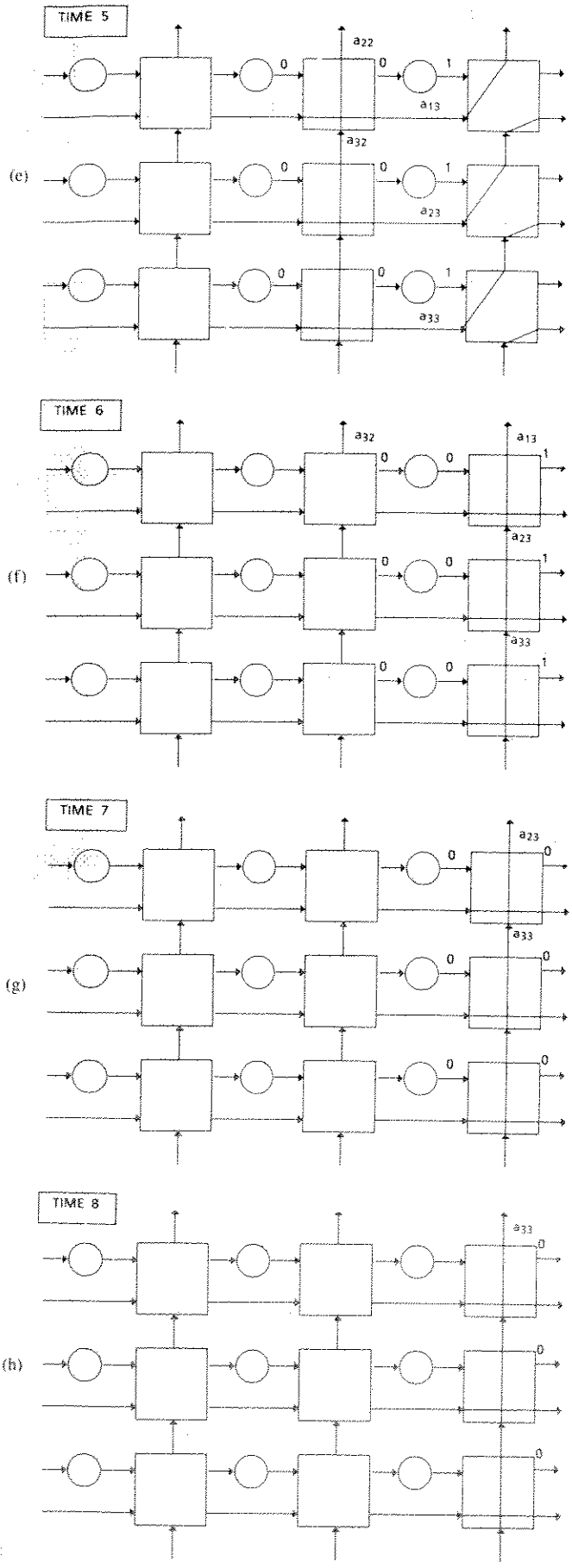

Fig. 3. (*Continued*).

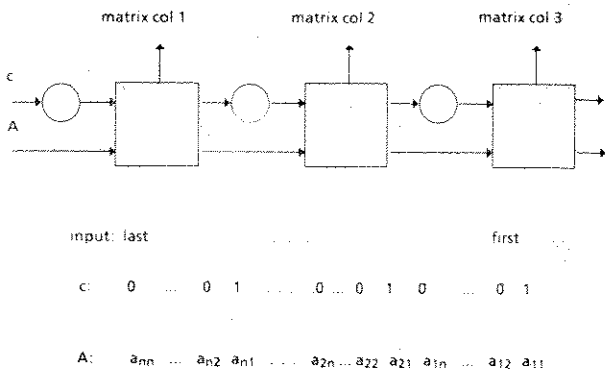

Fig. 4. A one-dimensional systolic array for matrix transpose ($n = 3$).

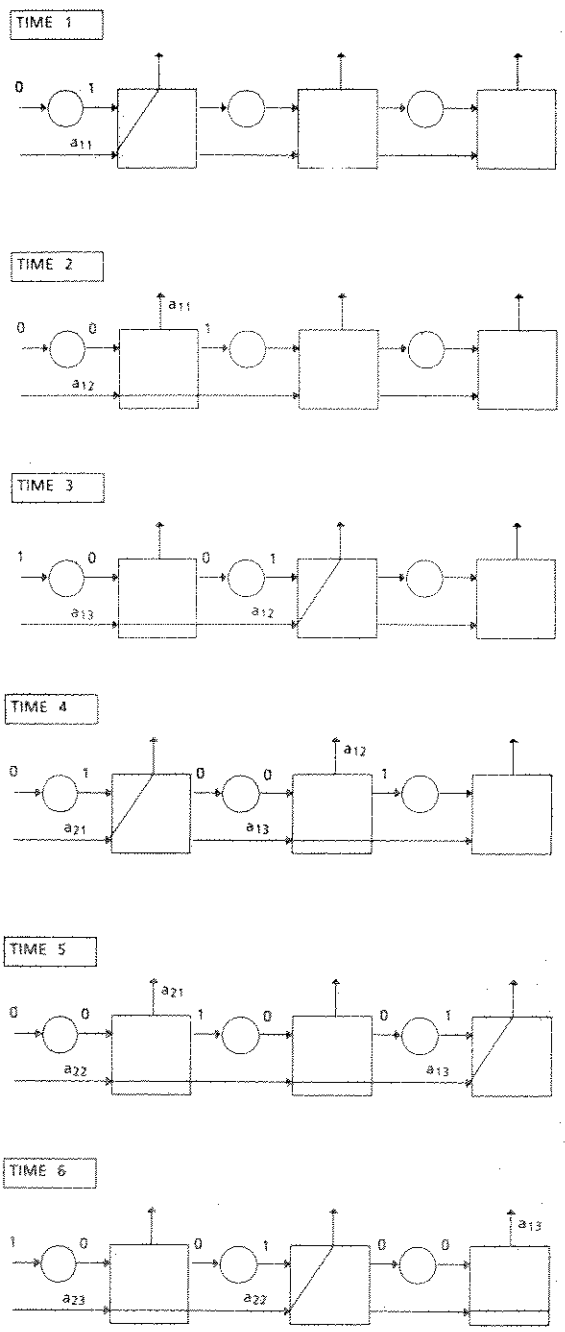

Fig. 5. Transpose of a 3 × 3 matrix using the one-dimensional array.

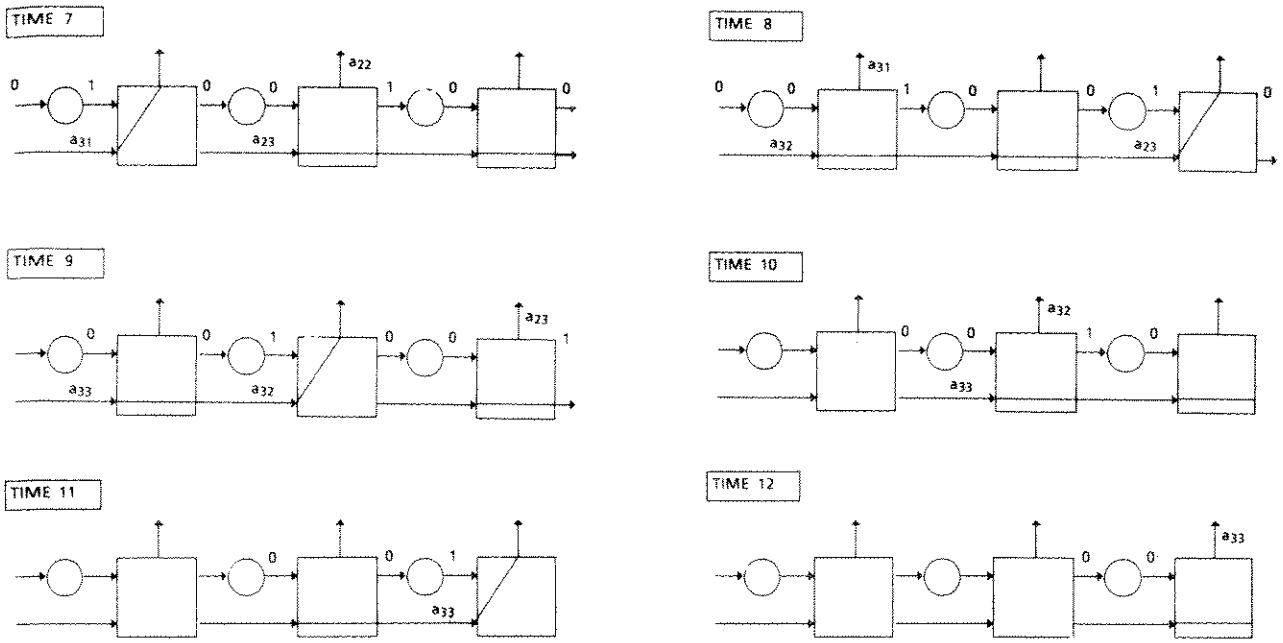

Fig. 5. *(Continued)*.

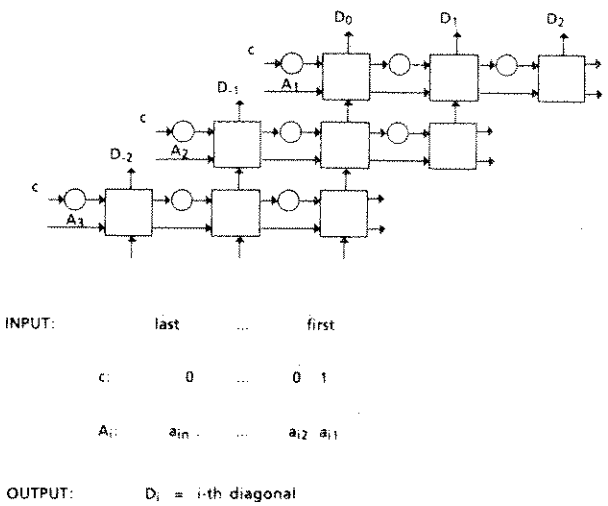

Fig. 6. An array for converting from row ordering to diagonal ordering ($n = 3$).

IV. In by Rows, Out by Diagonals

We present in this section two other arrays which modify the orderings of the matrix elements. The first, shown in Fig. 6, consists of n^2 processors plus bit buffers. It takes a matrix in by rows and puts it out by diagonals. We use the notation D_k to denote the kth diagonal, consisting of those elements a_{ij} for which $j - i = k$. The operation takes time $4n - 1$ if all rows are started simultaneously, and elements on a given diagonal are put out at intervals of four time units. The operation of the array is similar to the examples above, so a simulation is omitted.

The second array of n^2 processors plus bit buffers, shown in Fig. 7, transforms a matrix from diagonal ordering to row ordering. The time is $2n$, again assuming that all diagonals are started simultaneously. Here, the array must be primed with zero control signals before the matrix input is begun, and the processors are slightly more complicated than in the arrays above. The center row of switches perform one of three routings instead of only two, and the processors below pass an extra control signal. (See Fig. 1.) A simulation of the array is given in Fig. 8.

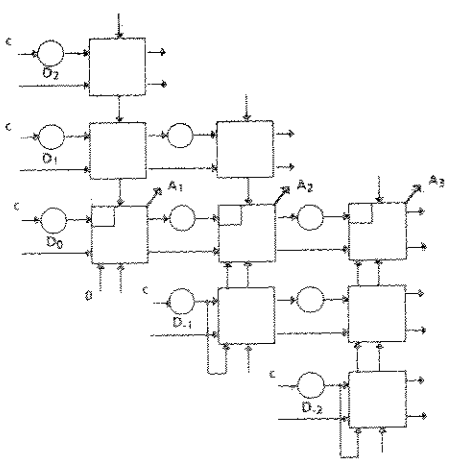

Fig. 7. An array for converting from diagonal ordering to row ordering ($n = 3$; notation as in Fig. 6).

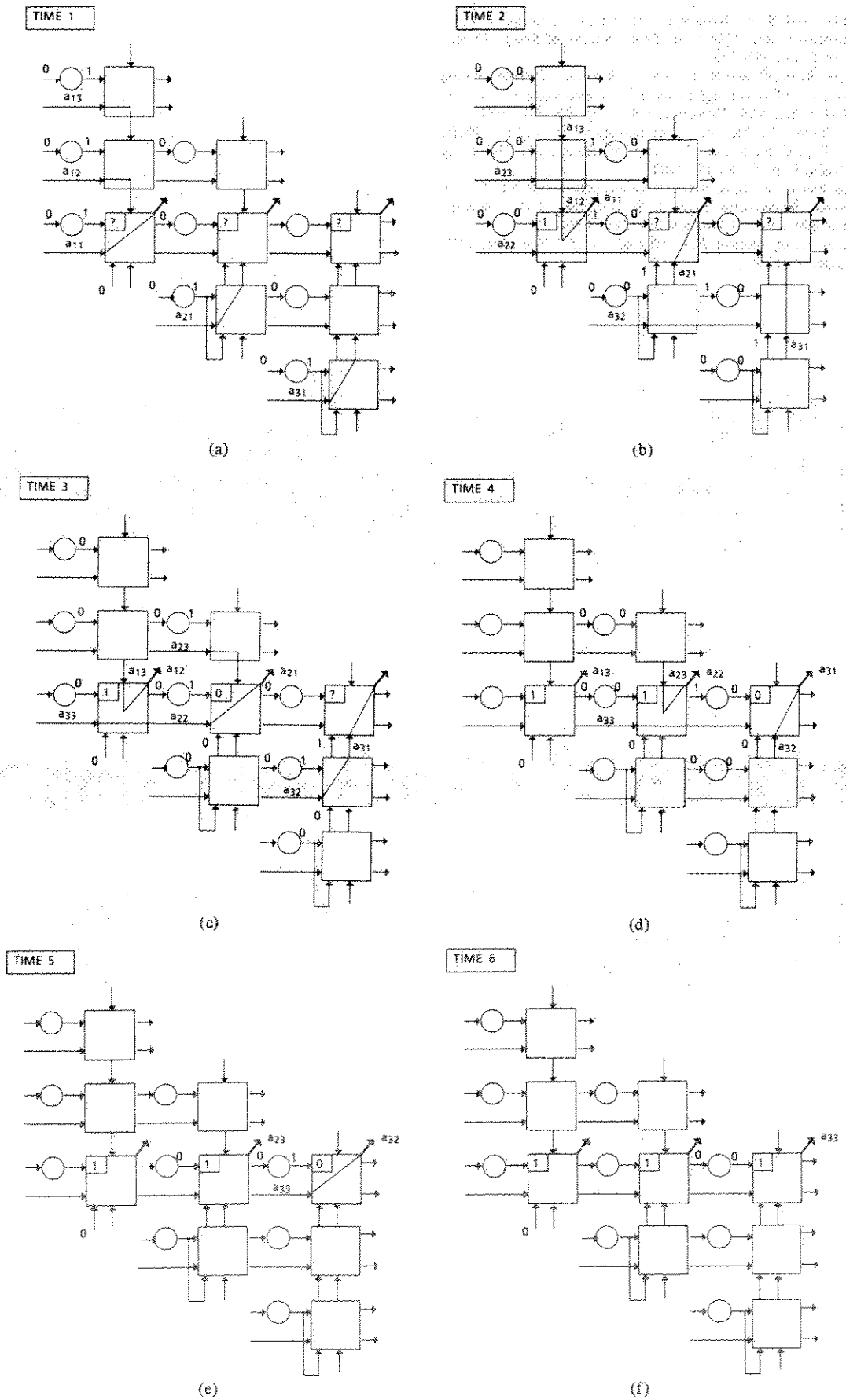

Fig. 8. Reordering of a 3 × 3 matrix using the array of Fig. 7

REFERENCES

[1] M. J. Atallah and S. R. Kosaraju, "Graph problems on a mesh-connected processor array," in *Proc. 14th Ann. ACM Symp. Theory Comput.*, 1982, pp. 345-353.

[2] A. Bojanczyk, R. Brent, and H. T. Kung, "Numerically stable solution of dense systems of linear equations using mesh-connected processors," *SIAM J. Sci. Stat. Comput.*, vol. 5, pp. 95-104, 1984.

[3] C. F. Ilse Ipsen, "Stable matrix computations in VLSI," Ph.D. dissertation, Dep. Comput. Sci., Pennsylvania State Univ., University Park, PA, Tech. Rep. CS-83-17, 1983.

[4] H. T. Kung and C. E. Leiserson, "Systolic arrays (for VLSI)," in *Sparse Matrix Proceedings 1978*, I. S. Duff and G. W. Stewart, Eds. Philadelphia, PA: SIAM, pp. 256-282, 1979.

[5] J. D. Ullman, *Computational Aspects of VLSI*. Rockville, MD: Computer Science, 1984.