[10] R. H. Dennard, F. H. Gaensslen, H. N. Yu, V. L. Rideout, E. Bassous, and A. R. LeBlanc, "Design of ion-implanted MOSFETs with very small physical dimensions," *IEEE J. Solid-State Circuits*, vol. SC-9, pp. 256–268, Oct. 1984.
[11] G. Bilardi, M. Pracchi, and F. P. Preparata, "A critique of network speed in VLSI models of computation," *IEEE J. Solid-State Circuits*, vol. SC-17, pp. 696–702, Aug. 1982.
[12] J. D. Ullman, *Computational Aspects of VLSI*. Rockville, MD: Computer Science, 1984.
[13] C. A. Mead and L. Conway, *Introduction to VLSI Systems*. Reading, MA: Addison-Wesley, 1980.
[14] F. P. Preparata, "A mesh-connected area-time optimal VLSI multiplier of large integers," *IEEE Trans. Comput.*, vol. C-32, pp. 194–198, Feb. 1983.
[15] F. T. Leighton, "Complexity issues in VLSI: Optical layouts for the shuffle-exchange graph and other networks," Ph.D. dissertation, Dep. Math., Massachusetts Inst. Technol., Cambridge, MA, Sept. 1981.
[16] F. P. Preparata and J. Vuillemin, "Area-time optimal VLSI networks for computing integer multiplication and discrete Fourier transform," in *Proc. I.C.A.L.P.*, Haifa, Israel, July 1981, pp. 29–40.
[17] G. Bilardi and M. Saccafzadeh, "Optimal discrete Fourier transform in VLSI," in *VLSI: Algorithms and Architect., Proc. of the Int. Workshop Parallel Comput. VLSI*, Amalfi, Italy, 1985, pp. 78–89.
[18] P. Duris, O. Sykora, C. D. Thompson, and G. Vrto, "On the area required for selection, discrete Fourier transformation and Walsh–Hadamard transformation," to be published.
[19] B. A. Bowen and W. R. Brown, *VLSI Systems Design for Digital Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1982, pp. 280–281.
[20] H. C. Card, W. Pries, and R. D. McLeod, "Contributions to VLSI computational complexity theory from bounds on current density," *Integration*, vol. 4, pp. 175–183, 1986.
[21] R. W. Keyes, "Physical limits in digital electronics," *Proc. IEEE*, vol. 63, pp. 740–767, May 1975.
[22] B. Chazelle and L. Monier, "A model of computation for VLSI with related complexity results," *J. Ass. Comput. Mach.*, vol. 32, pp. 573–588, July 1985.
[23] D. Kleitman, F. T. Leighton, M. Lepley, and G. L. Miller, "An asymptotically optimal layout for the shuffle-exchange graph," *J. Comput. Syst. Sci.*, vol. 6, pp. 339–361, June 1983.
[24] A. Aggarwal, "On I/O placement in VLSI," in *Proc. 21st Ann. Allerton Conf. Commun., Contr., Comput.*, Monticello, IL, Oct. 1983, pp. 236–243.
[25] G. Bilardi and F. P. Preparata, "A minimum area VLSI network for $O(\log N)$ time sorting," *IEEE Trans. Comput.*, vol. C-34, pp. 336–343, Apr. 1985.

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.**

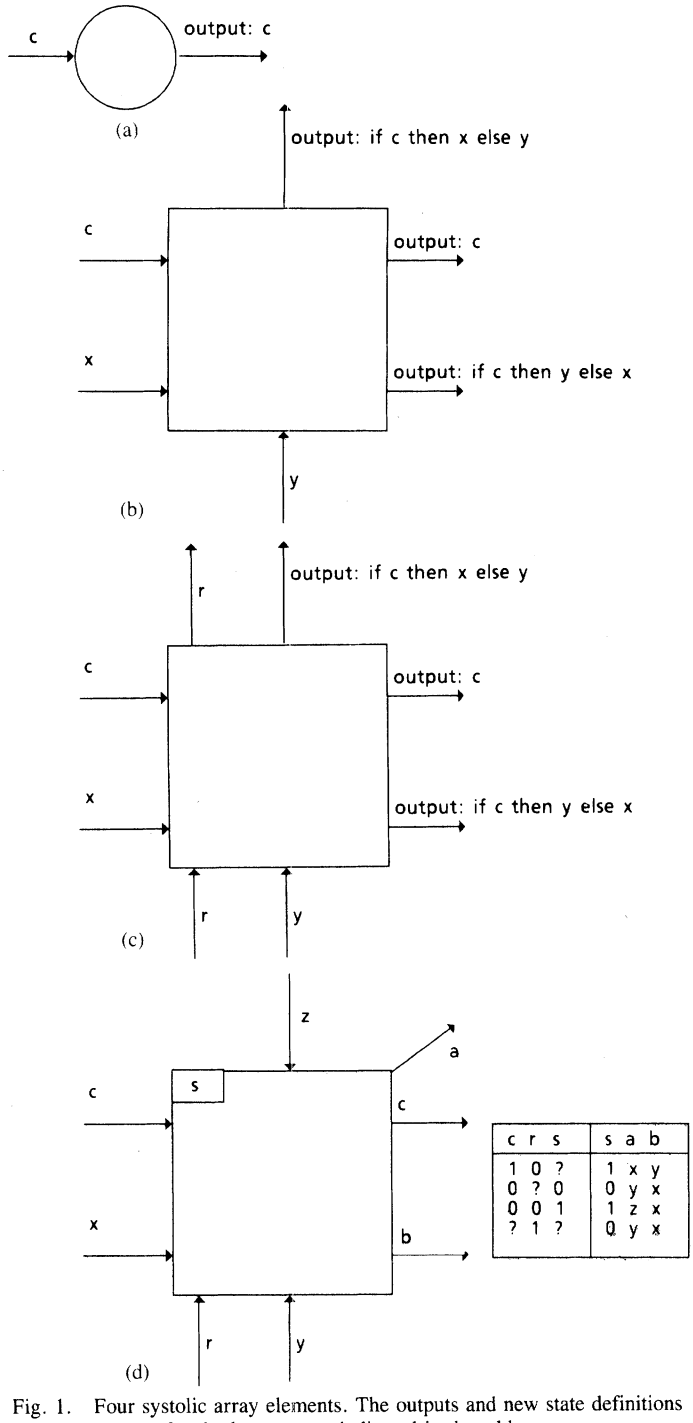
I. Introduction

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



c r s	s a b
1 0 ?	1 x y
0 ? 0	0 y x
0 0 1	1 z x
? 1 ?	0 y x

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

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.

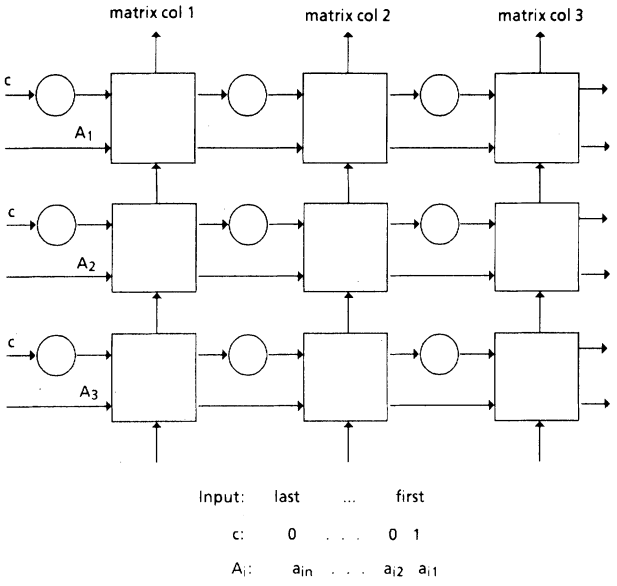


Fig. 2. A two-dimensional systolic array for matrix transpose ($n = 3$).

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 $n^2 + n$.

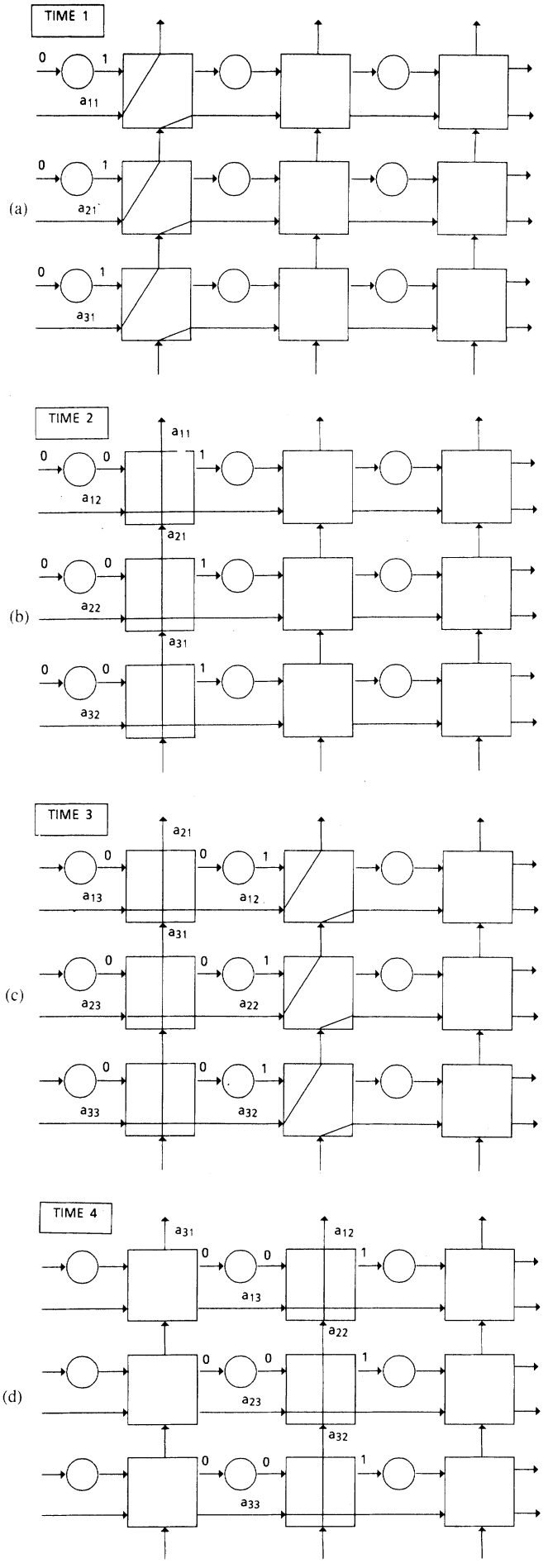


Fig. 3. Transpose of a 3×3 matrix using the two-dimensional array.

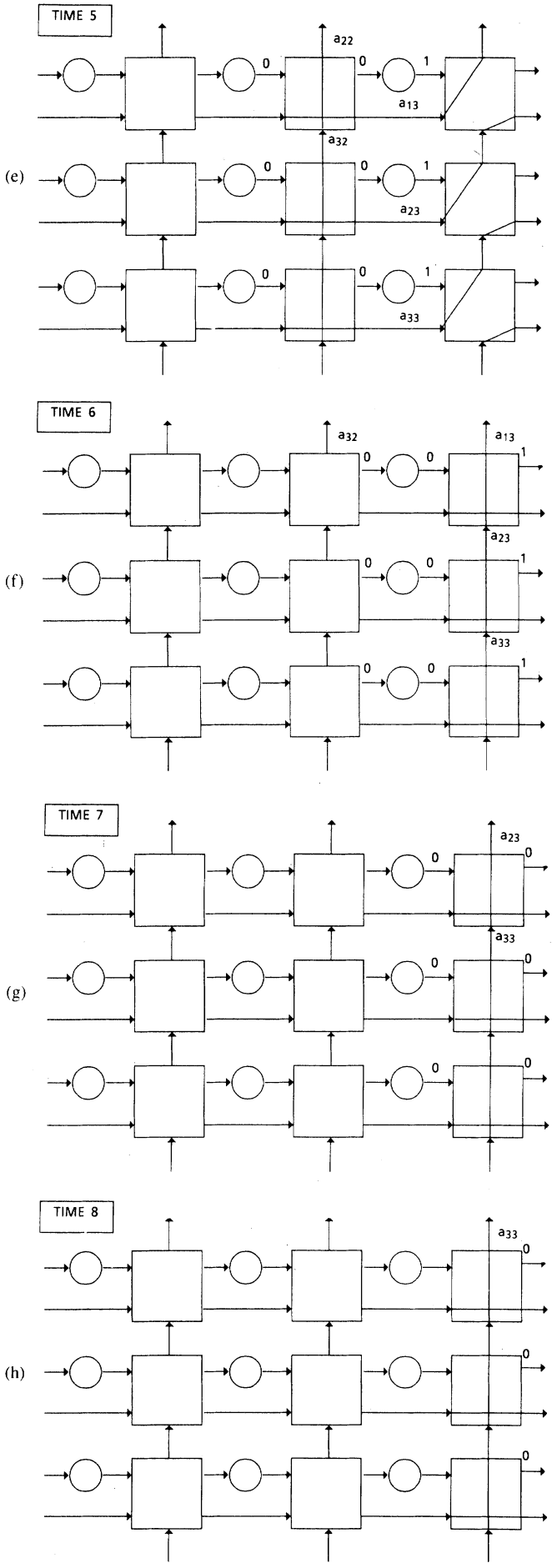


Fig. 3. (*Continued*).

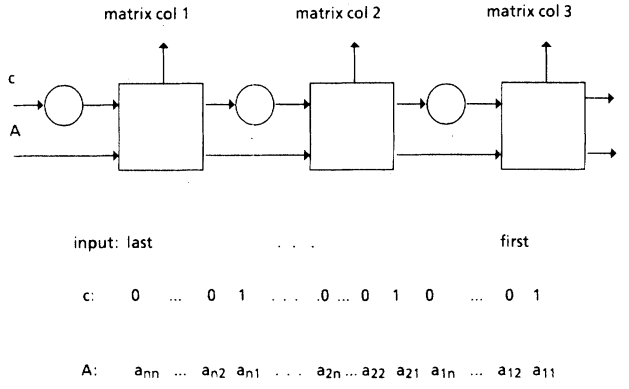


Fig. 4. A one-dimensional systolic array for matrix transpose ($n = 3$).

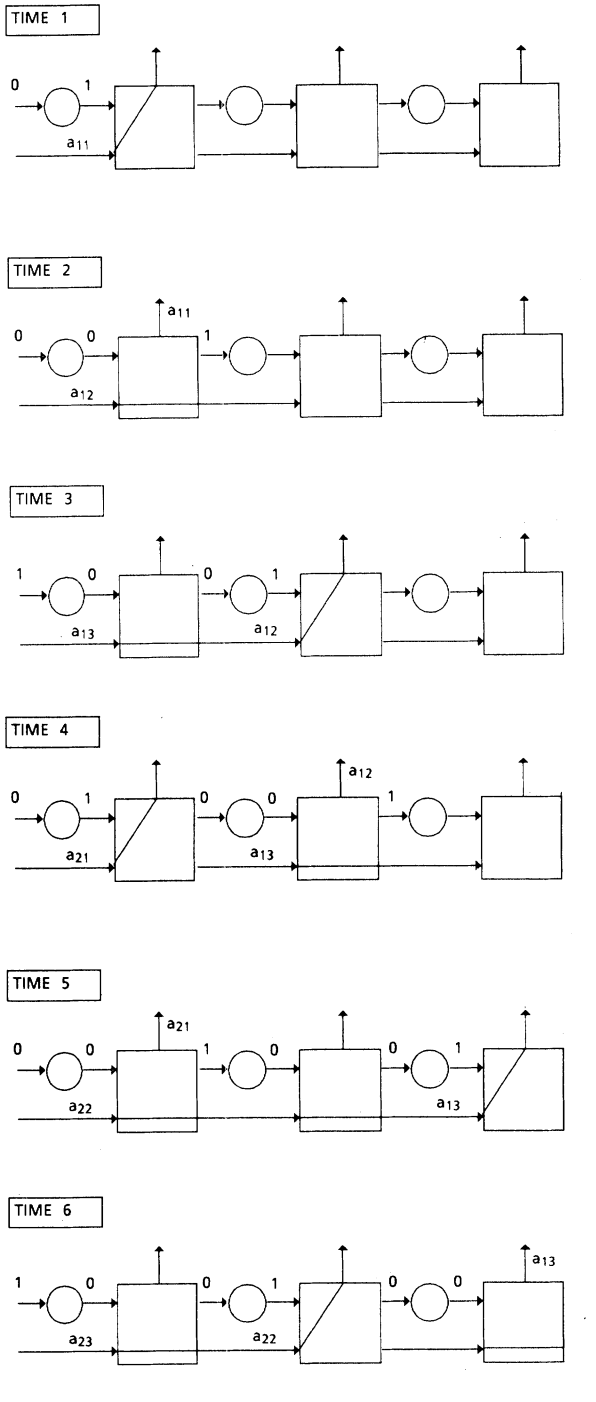


Fig. 5. Transpose of a 3 × 3 matrix using the one-dimensional array.

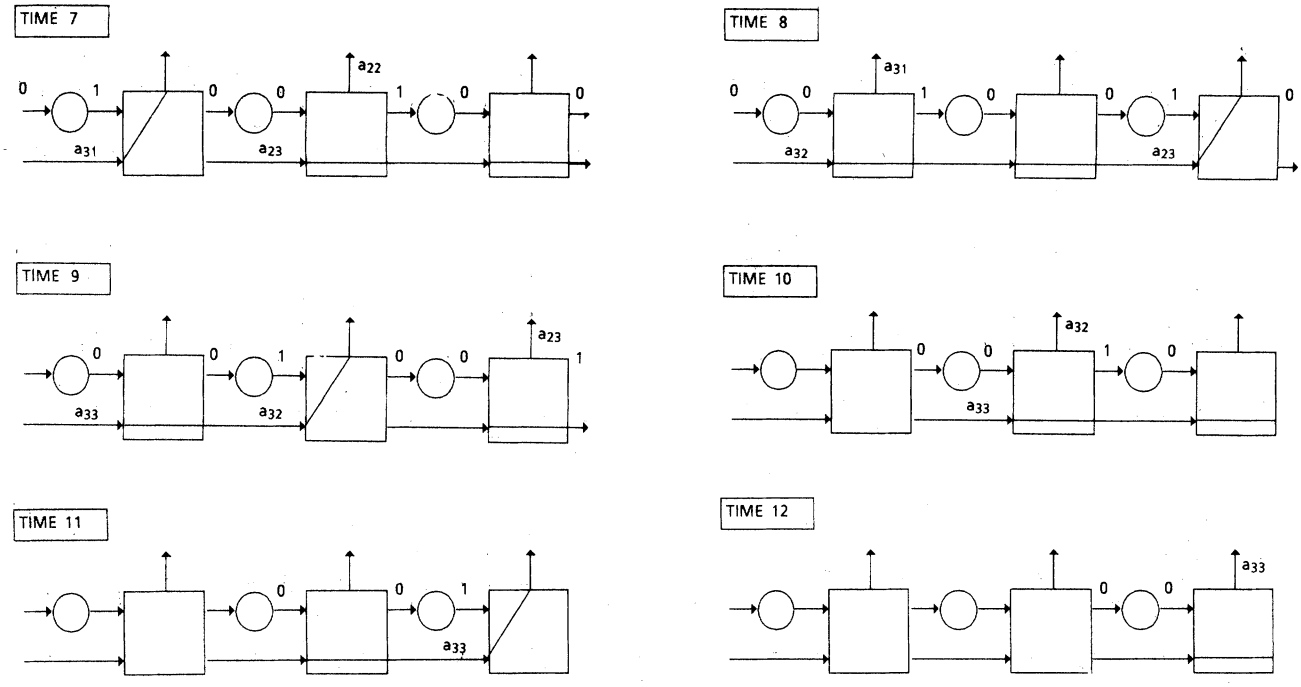


Fig. 5. *(Continued).*

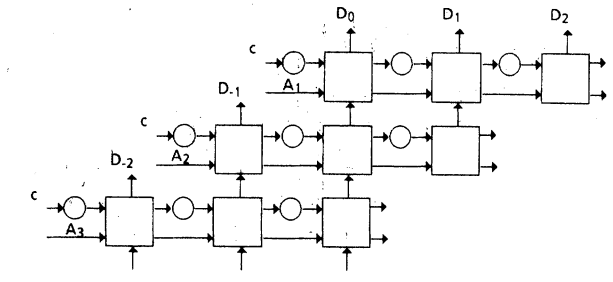


Fig. 6. An array for converting from row ordering to diagonal ordering ($n = 3$).

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.

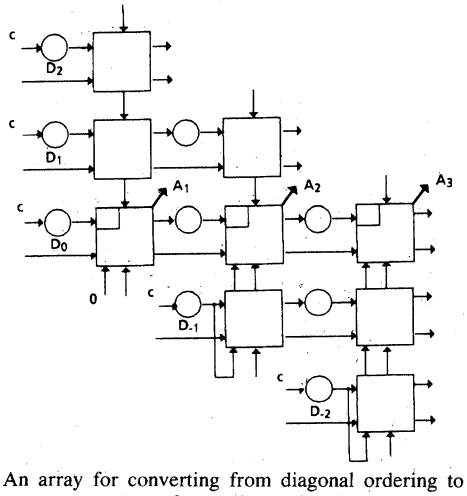


Fig. 7. An array for converting from diagonal ordering to row ordering ($n = 3$; notation as in Fig. 6).

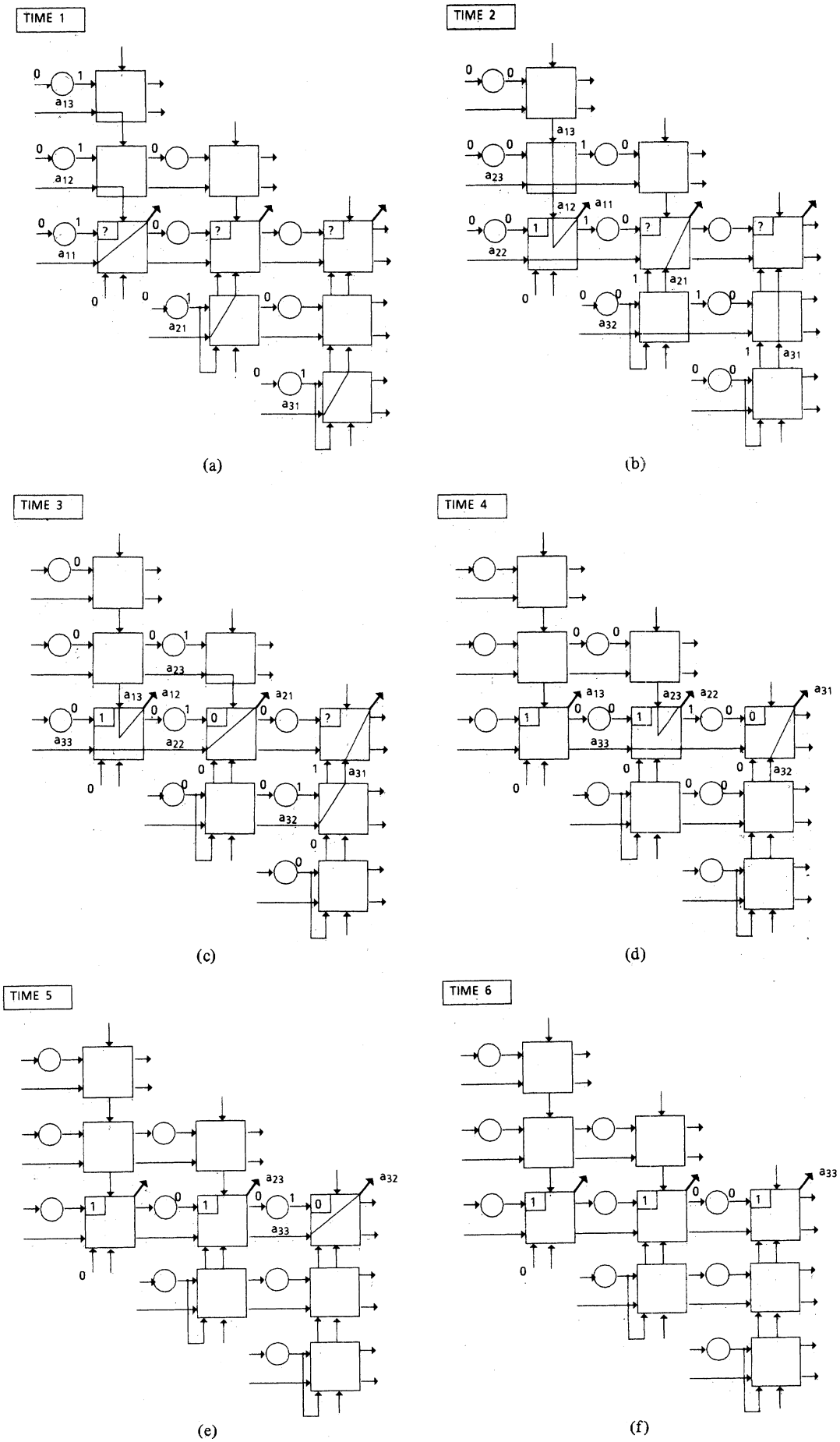


Fig. 8. Reordering of a 3 × 3 matrix using the array of Fig. 7

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.

Exact Product Form Solution for Queueing Networks with Blocking

I. F. AKYILDIZ

Abstract—**This work investigates closed queueing networks with blocking composed of two stations with multiple servers. Blocking occurs when a job wanting to enter a full station is forced to remain in its source station, thus blocking the source station until room is available at the destination station. This type of blocking is known as classical blocking. We show that, for a two-station closed queueing network with blocking, there exists an equivalent nonblocking network with the same state space structure. Utilizing this concept, we demonstrate that two-station closed queueing networks with blocking have product form solutions.**

Index Terms—**Blocking, equilibrium state probabilities, normalization constant, performance analysis, performance measures, queueing networks, state space transformations.**

I. INTRODUCTION

The basic results of product form networks are given in Baskett, Chandy, Muntz and Palacios [4]. They show that queueing networks with different classes of jobs, exponential and nonexponential service time distributions, and different queueing disciplines (FCFS, RR-PS or LCFS-PR) have product form solution. This was a remarkable result for the queueing network theory. The term "product form" means that the equilibrium state probabilities can be expressed as a product of terms for each queue in the network. The product form networks, also known as BCMP or separable networks, are based on the assumption that all stations have infinite capacities. If the stations have finite capacity, blocking can occur in the network. Since blocking causes interdependencies between stations, blocking queueing networks cannot be analyzed by existing product form algorithms.

Formally we distinguish between three types of blocking: *classical*, *rejection*, and *service* blocking.

In the first case, blocking occurs when a job completing service at station i cannot proceed to station j because station j is full. The job is forced to wait in the station i's server until it is allowed to enter the destination station j. The station i's server stops processing until station j releases a job [1], [2], [11], [13], [15], [20]. In the second case, blocking occurs when a job completing service at station i attempts to join destination station j. If station j is full at that moment, the job is refused. The rejected job goes with a certain probability (called rejection probability) back to the station i's server and receives a new service. This is repeated until some job completes a service at station j and a place becomes available [3], [9], [16], [22].

Manuscript received May 13, 1985; revised September 11, 1985.

The author is with the Department of Computer Science, Louisiana State University, Baton Rouge, LA 70803.

IEEE Log Number 8611265.

In the third case, blocking occurs when a job in front of queue at station i declares its destination station j before it starts its service in station i's server. If the destination station j is full, the ith server becomes blocked, i.e., it cannot serve jobs. When a departure occurs from destination station j, the ith server becomes unblocked and the job begins receiving service [5], [8], [10], [19].

In recent years there has been a growing interest in the development of computational methods to analyze queueing networks with blocking. The interest developed primarily from the realization that these models are useful in the study of subsystem behavior in computers and communication networks, in addition to providing detailed descriptions of several computer-related applications such as flexible manufacturing systems.

Several investigators in recent years have published results on queueing networks with classical, rejection and service blocking. A bibliography concerning queueing network models with blocking is given by Perros [14]. Formal comparisons between the distinct classes of blocking have been carried out by Onvural and Perros [12].

In this paper we introduce the state space transformation concept which provides exact results for two-station closed queueing networks with classical blocking.

II. PRODUCT FORM SOLUTION

We will consider closed queueing networks with $N = 2$ stations and K single-class jobs. Each station consists of a single queue served by ($m_i \geq 1$), servers each with exponentially distributed service time with mean value $1/\mu_i$ (for $i = 1, 2$). The service discipline in each station is first-come, first-served. Each station has a fixed finite capacity M_i where M_i = (queue capacity + m_i), (for $i = 1, 2$). Cases in which the stations can have infinite capacity are also allowed. Any station whose capacity exceeds the total number of jobs in the network can be considered to have infinite capacity. It is obvious that the total number of jobs K must be smaller than the sum of the station capacities, that is,

$$K < \sum_{i=1}^{2} M_i.$$

A job which is serviced by the ith station proceeds to the jth station with probability p_{ij}, (for $i, j = 1, 2$), if the jth station is not full. That is, if the number of jobs in the jth station k_j is less or equal to M_j for $i, j = 1, 2$. Otherwise, the job is blocked in the ith station until a job in the jth station has completed its servicing and a place becomes available. Note that the case $i = j$ is allowed. A station can have a transition back to itself, which will be shown by an example in Section IV. The queueing network with the above assumptions is known as the network with classical blocking which we will investigate in this work.

As is generally known, the following binomial coefficient formula is valid for closed queueing networks without station capacity limits. It indicates the number of possible ways that K jobs can be distributed into N stations.

$$Z = \binom{N+K-1}{N-1} \tag{1}$$

where Z is the total number of states in a closed queueing network.

For queueing networks with two stations (1) is simplified to

$$Z = K + 1. \tag{2}$$

Structure for the state space of the two-station network is illustrated in Fig. 1.

The state (k, n) denotes that k jobs are in the first station and n jobs are in the second station. The transition rates from one state to another are equal to the service rates μ_i multiplied by the number of servers m_i in each station. Fig. 1 shows that there must be at least ($K - m_i$) waiting places in each queue (m_i jobs can be in service) to ensure that all states are feasible. Since each station in the network

0018-9340/87/0100-0122$01.00