TRANSLATION OF SYSTOLIC ALGORITHMS BETWEEN SYSTEMS OF DIFFERENT TOPOLOGY

K. Culik II
Sheng Yu

Department of Computer Science
University of Waterloo
Waterloo, Ontario N2L 3G1
Canada

CS-85-06
May 1985

TRANSLATION OF SYSTOLIC ALGORITHMS BETWEEN SYSTEMS OF DIFFERENT TOPOLOGY[a]

K. Culik II and *S. Yu*

Department of Computer Science
University of Waterloo
Waterloo, Ontario N2L 3G1
Canada

Abstract. The concept of topological transformations of algorithms between different systolic systems has been introduced by the first author and Fris in [1]. Here we will show some interesting applications of this technique. For example, we will show that any algorithm for a linear systolic array can be transformed to a computationally equivalent twice slower algorithm for a unidirectional systolic ring with the same number of processors and that any algorithm for a bidirectional 2-dim array can be transformed to three times slower one for a unidirectional toroid. These translations of algorithms are very simple and can be mechanically performed. Therefore, it is easy to design software which, for example, accepts high level programs for bidirectional linear arrays and produces low level programs for a systolic ring. The former are easy to write while the latter are easy to implement and make fault-tolerant.

1. INTRODUCTION

The concept of topological transformations of algorithms between different systolic systems has been introduced in [1]. In this paper, we will show that algorithms for linear or multi-dimensional iterative arrays can be transformed to algorithms for one-way systolic rings or toroids by the means of topological transformation. We will also use this tool to show that a cellular automaton can be simulated by an iterative array or a one-way systolic ring. In this section, we will illustrate the concept of topological transformation by an example. A formal definition will be given in Section 3.

We need a suitable notion of simulation for systolic networks. This notion of simulation should be strong enough to require that two networks not only compute essentially the same input-output function, but that they do it by using essentially the same algorithm. On the other hand, this simulation should be weak enough to allow that certain elementary computation steps have different locality and timing. We study the similarity between systolic networks by comparing their space-time diagrams which we call unrollings, see [5]. The concept of the topological simulation is based on the similarity of the unrollings of different systolic networks.

We start with an example which, we hope, will give readers an intuitive view of our main tool. Our goal in this example is to design a sorting algorithm for a systolic ring using the well known odd-even transportation sorting method [17]. Since this method can be easily implemented on the systolic network N shown in Figure 1.1, we will give an algorithm for N first, and then transform it into an algorithm for a systolic ring by using our tool of topological transformation. Note that the unrolling of N is shown in Figure 1.2 and is called a trellis [2,3,7].

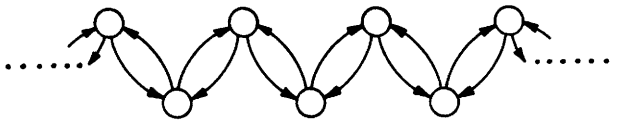


Figure 1.1

Let $LI(RI)$ and $LO(RO)$ denote the left (right) input and left (right) output respectively. A simple sorting algorithm (for each cell of N) is as follows.

if $LI = B$ **or** $RI = B$ **then**

$$\begin{bmatrix} LO := LI; \\ RO := RI; \end{bmatrix}$$

else

$$\begin{bmatrix} LO := min(LI,RI); \\ RO := max(LI,RI); \end{bmatrix}$$

Observe that the graph in Figure 1.3 shows a different geometrical representation of the same directed graph of Figure 1.2. Clearly, these two

[a] This work was supported by the Natural Sciences and Engineering Research Council of Canada under Grant A-7403.

This paper will be appeared in the Proceeding of 1985 International Conference on Parallel Processing, August 1985.

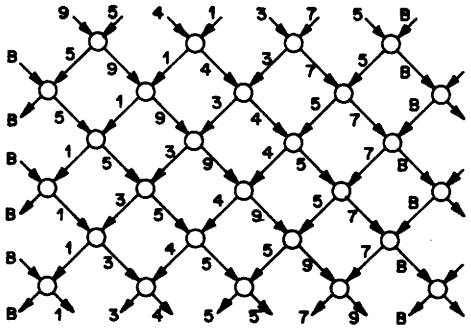


Figure 1.2

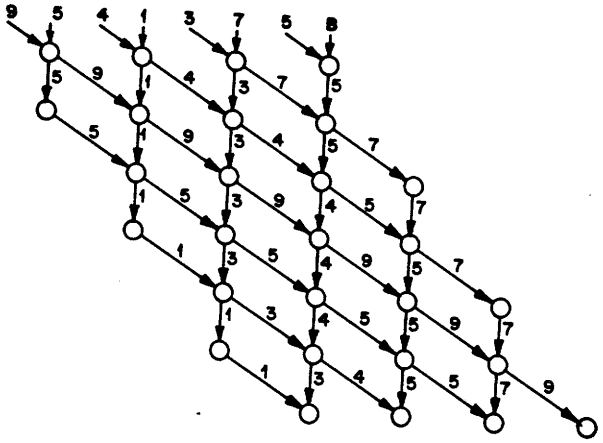


Figure 1.3

graphs are topologically equivalent. However, we now observe that Figure 1.3 is also an unrolling of a systolic network, namely the unrolling of a one-way linear array A shown in Figure 1.4. Since their unrollings are topological equivalent, the sorting algorithm for network N can be directly transformed to one for network A, which requires simpler communication but more processors. To further improve this, we observe that instead of using a new processor on the right hand side at every even step in Figure 1.3, we can actually re-use the "abandoned" processor on the left hand side. The graph rolls back to the left instead of stretching to the right. Thus, Figure 1.3 can be further topologically transformed to obtain Figure 1.5. We notice that the Figure 1.5 is the unrolling of a one-way systolic ring shown in Figure 1.6. So, we can now write down the sorting algorithm for this one-way systolic ring by

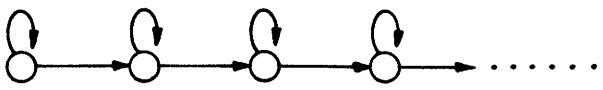


Figure 1.4

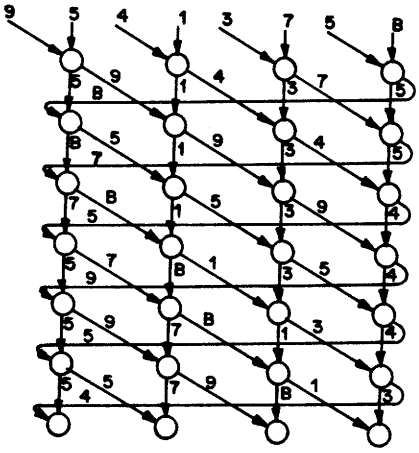


Figure 1.5

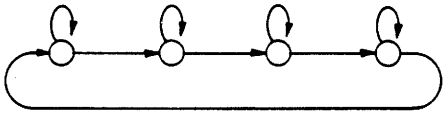


Figure 1.6

simply changing the names of the variables of the previous algorithm. Here we use IN,OUT and R to denote the input, output and the register of a cell, respectively. For simplicity, we omit the accesses of external input and output.

if $IN = B$ **or** $R = B$ **then**

$$\begin{bmatrix} OUT := R; \\ R := IN; \end{bmatrix}$$

else

$$\begin{bmatrix} OUT := max(IN,R); \\ R := min(IN,R); \end{bmatrix}$$

In the above example, the resulting algorithm for a systolic ring uses the same number of processors and the same number of steps as the one for network N but simpler communication. Assuming that the original algorithm is correct, we have also proved the correctness of the new algorithm for a different systolic network.

2. SYSTOLIC NETWORKS AND THEIR UNROLLINGS

Recently, systolic networks or algorithms have been studied extensively. See, for example, [5,7,9,11,12]. Here we use a similar definition of systolic networks as given in [12]. Informally, a systolic network is a directed graph, finite or infinite, with a function associated with each node and positive delay with each directed edge. There is one distinct node, called the "host", which is the "I/O interface" of the network to the external world. At each time step, every processor gets the inputs from all its incoming edges and produces outputs to its outgoing edges. For formal definition see [1].

In this paper, we mainly restrict our interest to the systolic networks with single serial input stream from the host and single serial output stream to the host.

An unrolling of a network is essentially a space-time diagram. It is an infinite digraph obtained by redrawing a new copy of the whole network at each time step and changing the destination of every communication line to the one described as follows. Let c_i^t denote the node c_i of network N at time t. If (c_i,c_j) is a directed edge of N and k is the delay of this edge, then in the unrolling of N, there should be a directed edge (c_i^t,c_j^{t+k}) rather than (c_i^t,c_j^t), for any $t \geq 0$. Since we assume that all the delays are positive, there are no cycles in any unrolling. We usually use $\hat{N}$ to denote the unrolling of N.

The delays associated with the edges of N are important in the formation of the unrolling $\hat{N}$, but they are no longer meaningful in the unrolling which has been formed. Each edge in $\hat{N}$ simply represents a relation of data dependency. The whole unrolling is a dataflow diagram.

3. SIMULATION OF SYSTOLIC NETWORKS

Two systolic networks N_1 and N_2, which produce the same output for the same input, are naturally considered equivalent. Similarly, if any input string of N_1 is translated into an input string of N_2 by a finite transduction and the output of N_2 is translated back to the output of N_1 by another finite transduction, see Figure 3.1, then we say that N_2 simulates N_1.

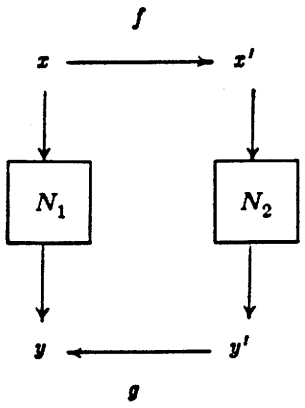


Figure 3.1

This input-output simulation is too general for our purpose since we are interested in a systolic implementation of a given algorithm. Thus, we are free to choose a time and location to perform its elementary steps but we want to preserve the structure of the algorithm. We introduce a stronger concept of simulation for systolic networks, called topological simulation.

In an unrolling $\hat{N}$ of a systolic network N, the output value of a node depends on the output values of certain other nodes. Since the edges of the unrolling $\hat{N}$ have no delays attached to them and they simply represent the data dependency, the shape of a unrolling, either skewed or straight, is not important to the computation. An unrolling $\hat{N}$ of a systolic network N is actually a "Hasse diagram" of a partially ordered set (*poset*) [6]. All the nodes of $\hat{N}$ constitute the *poset* and the edges of $\hat{N}$ show the relation. The elements are distinguished by their assigned functions. Two unrollings, whatever shapes they have, will be considered to be equivalent if they specify the same partial order of elementary computation steps. A partial ordering still can be preserved when the *poset* is enlarged by adding new elments or is projected to a smaller set, etc. Consequently, if we reshape an unrolling without changing the existing partial ordering, or we insert new nodes between the original ones or group several nodes into one in some regular way etc, and if the newly formed "Hasse diagram" is an unrolling of

another network N', then the computations of N can be recovered step by step from the computations of N'. Then we say that N' topologically simulates N. This is an intuitive explanation of the concept of topological simulation.

Let N_1 and N_2 be two systolic networks. We say, more formally, that N_2 topologically simulates N_1 if there is a mapping ρ from unrolling $\hat{N}_1$ to $\hat{N}_2$ and a function τ such that for any node $\hat{v}$ in $\hat{N}_1$ there is a node $\hat{v}' = \rho(\hat{v})$ in $\hat{N}_2$ which satisfies $\sigma_1(\hat{v}) = \tau(\sigma_2(\hat{v}'),\hat{v})$, for any computation σ_1, σ_2 of N_1 and N_2, respectively. A computation σ on a systolic network N is formally defined as an assignment of values to the edges of the unrolling of N which satisfies all the functions at the nodes.

It is easy to see that if systolic network N_1 topologically simulates N_2, then N_1 with minor changes input-output simulates N_2. The changes will only involve adding the data paths for input and output symbols from or to the host.

4. THE SIMULATION OF ITERATIVE ARRAYS BY SYSTOLIC RINGS AND TOROIDS

Linear or multi-dimensional arrays are simple and powerful models for systolic systems. Many theoretical properties as well as application algorithms for them are known [9,11]. One-way systolic ring is an even simpler structure which has less communication lines and is easier to implement and to make fault-tolerant. See [4,10,13]. But in many cases, it appears to be more convenient to design an algorithm for the linear array rather than for the ring. Thus, it might be advantageous to design an algorithm for the former and then transform it to the one for the latter. In this section, we show that such transformation is always possible and can be performed mechanically.

Observe that, one step operation of a bidirectional array can be simulated by two steps of a unidirectional array, see Figure 4.1. After two steps of simulation, the corresponding node in the simulation migrates to the next node. A more general view of this simulation is shown in Figure 4.2.

Let LI, RI, LO, RO and REG represent the left and right input, left and right output and the register of a cell of a linear iterative array. In each time step the following functions are computed.

$$LO := f_1(LI,REG,RI);$$
$$RO := f_3(LI,REG,RI);$$
$$REG := f_2(LI,REG,RI);$$

To simulate the same computation on a unidirectional linear array, we need two time steps and two registers in a cell for storing the intermediate

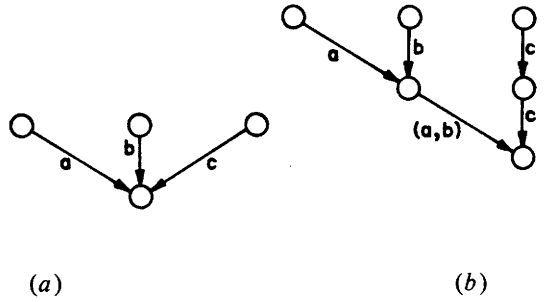


Figure 4.1

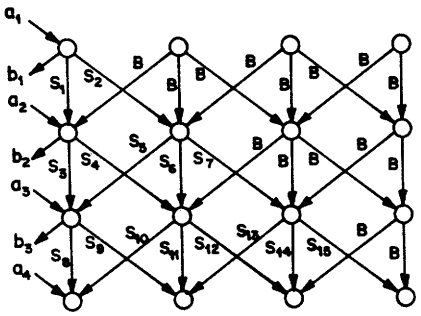


Figure 4.2(a)

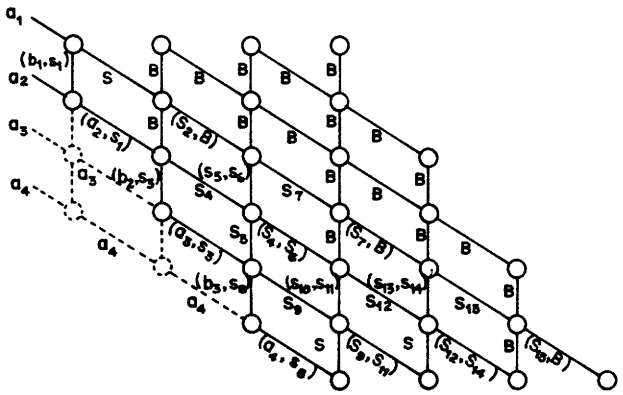


Figure 4.2(b)

results. Let IN, OUT, $R1$ and $R2$ denote the input, output, register 1 and register 2 of a cell, respectively. Then the first step of the two step operation is

$$OUT := f'(IN,R2);$$

and the second step is

$$OUT := f'_3(IN,R1);$$
$$R2 := f'_2(IN,R1);$$
$$R1 := f'_1(IN,R1);$$

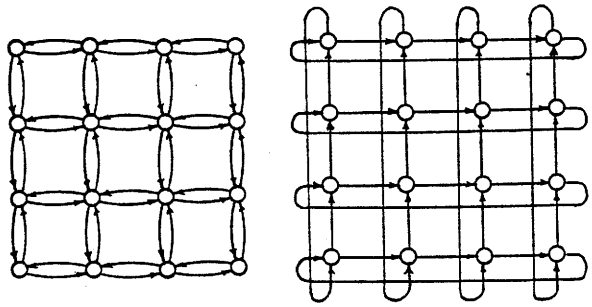
where $f'_i(f'(a,b),c) = f_i(a,b,c)$ for $i = 1, 2, 3$. The input symbols are read by the first cell of the unidirectional linear array and passed to the cell which is currently simulating the first cell of the bidirectional array. Since the input symbols are passed at full speed from left to the right and the image of the simulated first cell is moving at half speed in the same direction, the input arrive at the simulated first cell one symbol every two steps, which is just the speed of the simulation.

By the same observation as in the example of the sorting algorithm given in Section 1, we obtain an algorithm for the unidirectional systolic ring. The image of every simulated cell actually moves along the ring once every the other step. We assume that the reading and writing of external inputs and outputs is always done by a special cell. So, we need a marker to show the location of the simulated first cell and a channel to pass the input symbols to the image of the first cell. The output is passed in the same direction and reaches the special cell from the other end of the ring. Since the output is produced every the other step and transported at full speed, one output symbols will be communicated to the host at each time step.

Notice that in the first step of the two step simulation, the intermediate result $f'(IN,R2)$ has to be passed to the next cell. This does not necessarily increase the size of the alphabet of the working symbols. In many algorithms, such intermediate values are already available.

Let n be the number of cells used by the simulated array, and s be the number of cells in the unidirectional ring. The ideal case is $n = s$. If s is much bigger than n, the output will go through a long path and will be unnecessarily delayed. Some hardware designs provide the devices which allow half or a quarter of the total cells to be used as a ring, e.g., the WATERLOOP project. This flexibility avoids the unnecessary waste of the resources. If $n > s$, there are mainly two solutions. First, we can use the "grouping" technique. Chose a suitable k ($k = \lceil n/s \rceil$) and simulate k cells by one cell. The other solution is to logically divide every cell into k tracks and the whole ring is formed by going around the physical ring k times.

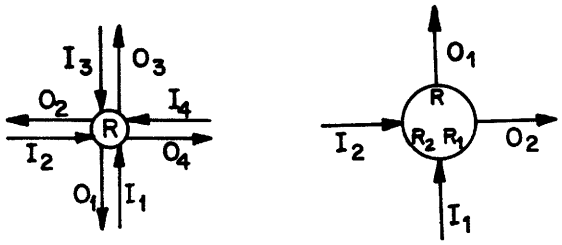
Multi-dimensional bidirectional arrays are very useful for matrix computations. Using the same idea as the one for linear arrays, any n-dimensional bidirectional array can be simulated by a n-dimensional unidirectional toroid, shown for $n = 2$ in Figure 4.3(b), with $n+1$ steps simulating 1 step. We show this for $n = 2$.



(a) (b)

Figure 4.3

Let I_i and O_i, $i = 1,2,3,4$, be the four inputs and four outputs of a node as shown in Figure 4.4(a), and R be the register. Then one step computation on the bidirectional array can be described as follows.



(a) (b)

Figure 4.4

$$O_1 := f_1(I_1,I_2,I_3,I_4,R);$$
$$O_2 := f_2(I_1,I_2,I_3,I_4,R);$$
$$O_3 := f_3(I_1,I_2,I_3,I_4,R);$$
$$O_4 := f_4(I_1,I_2,I_3,I_4,R);$$
$$R := f_5(I_1,I_2,I_3,I_4,R);$$

Now, we use three steps of a toroid to simulate the above computation.

Step I:

$$O_1 = I_1;$$
$$O_2 = f'(I_2,R);$$

Step II:

$$O_1 = f''(I_2,R_2);$$
$$O_2 = f'''(I_1,R_1);$$

Step III:

$$O_1 = f'_3(I_1,I_2);$$
$$O_2 = f'_4(I_1,I_2);$$
$$R = f'_5(I_1,I_2);$$
$$R_1 = f'_1(I_1,I_2);$$
$$R_2 = f'_2(I_1,I_2);$$

where $f_i(f''(f'(a,b),c),f'''(d,e)) = f_i(d,a,e,c,b)$, for $i = 1, 2, 3, 4, 5$.

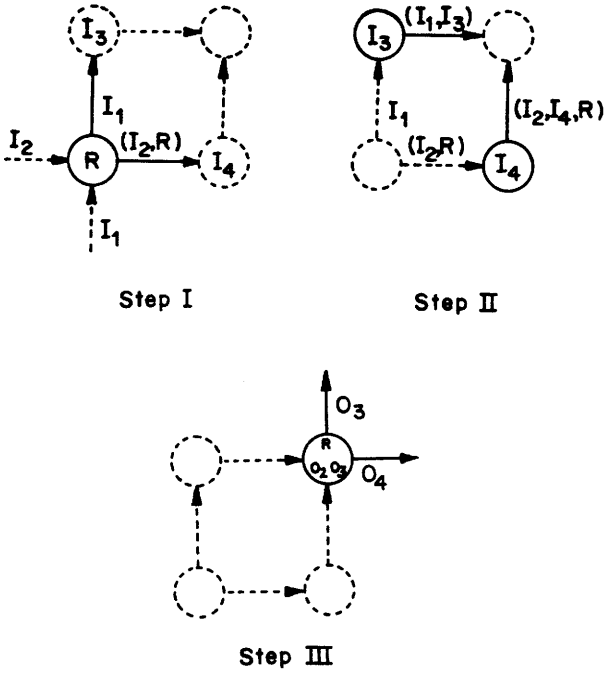


Figure 4.5

Figure 4.5 gives an intuition of this 3-step simulation. The image of a simulated cell moves along a diagonal line to the upper right direction at speed of three steps a shift. There are various ways to define the intermediate functions f', f'' and f'''. The best way of defining them is the one which needs minimal communication between the cells.

5. SIMULATION OF REAL-TIME CELLULAR AUTOMATA BY ITERATIVE ARRAYS IN *2N* STEPS

This topic has been studied in [14,16]. The simulation in [16] takes *6n-2* steps (but can be done in $3n-1$ steps if Systolic Conversion Theorem [12] had been used). Here we show that a cellular automaton as a language acceptor can be simulated by an iterative array in $2n-1$ steps. Recently, this result was also obtained independently by Ibarra, Palis and Kim ([8]) using a different technique. Here, we use the tool of topological transformation. This example suggests that using this technique, we can transform any algorithm for a systolic network with parallel input to an algorithm for a network with serial input.

Figure 5.1 shows a real-time computation of a cellular automaton. To transform it into a computation with serial input, we first turn the unrolling of it anticlockwise a right angle. It becomes a digraph shown in Figure 5.2(a).

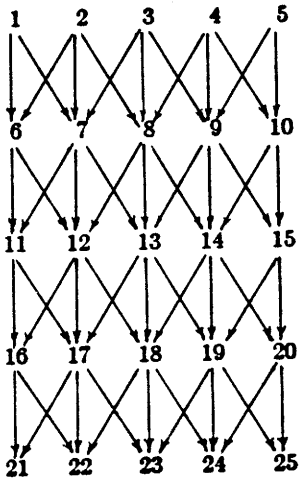


Figure 5.1

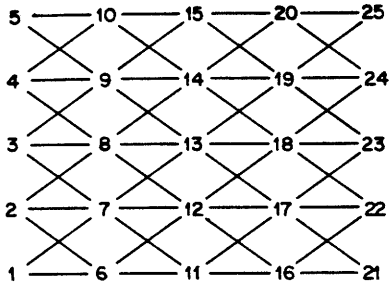


Figure 5.2

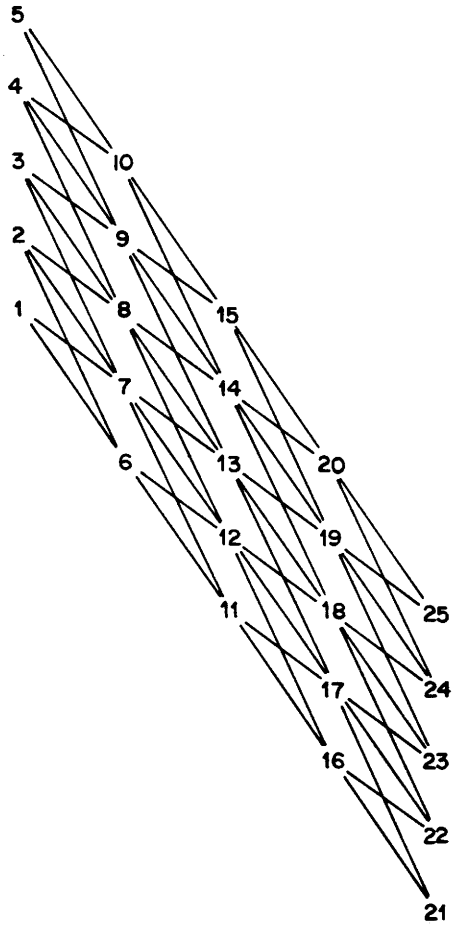


Figure 5.3

But it is not an unrolling of any network. However, by shifting every next column two time steps down, we get the diagram in Figure 5.2(b) which is the unrolling of the network in Figure 5.4. Now this network can be easily simulated by the network shown in Figure 5.5. By using the Systolic Conversion Theorem, the collecting of an output symbol from any cell at any step can be changed to the collecting at the first cell at the same step. So, the result is easily obtained. The details of the transformations of the state transition functions are omitted.

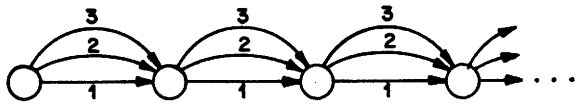


Figure 5.4

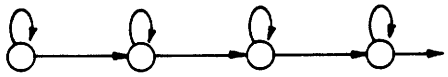


Figure 5.5

References

[1] K. Culik II, and I. Fris, Topological Transformations as a Tool in the Design of Systolic Networks, Department of Computer Science, University of Waterloo, Research Report, CS-84-11, (April 1984).

[2] K. Culik II, J. Gruska, and A. Salomaa, "Systolic Trellis Automata, Part I," *Inter. J. Computer Math., 15*, (1984), 195-212.

[3] K. Culik II, J. Gruska, and A. Salomaa, "Systolic Trellis Automata, Part II," *Inter. J. Computer Math., 16*, (1984), 3-22.

[4] K. Culik II, and S. Yu, Fault-Tolerant Schemes for Some Systolic Systems, Department of Computer Science, University of Waterloo, Research Report CS-82-39, (Oct. 1984).

[5] K. Culik, and J. Pachl, "Folding and Unrolling Systolic Arrays," *ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, Ottawa, (August 1982).

[6] B. Dushnik, and E.W. Miller, "Partially Ordered Sets," *Amer. J. Math. 63*, (1941), 600-610.

[7] O.H. Ibarra, M.A. Palis, and S.M. Kim, "Design Systolic Algorithms Using Sequential Machines," *25 Annual Symposium on Foundation of Computer Science*, Singer Island, Florida, (Oct. 1984), 46-55.

[8] O.H. Ibarra, M.A. Palis, and S.M. Kim, Some Results Concerning Linear Iterative (Systolic) Arrays, Computer Science Department, University of Minnesota, Technical Report 84-16, also to appear in *J. of Parallel and distributed Computing.*

[9] H.T. Kung, "Why Systolic Architecture?" *Computer, 15*, 1(1982), 37-46.

[10] H.T. Kung, and M.S. Lam, "Fault-Tolerant and Two-Level Pipelining in VLSI Systolic Arrays," *MIT Conf. on Advanced Research in VLSI*, (Jan. 1984).

[11] H.T. Kung, and C.E. Leiserson, "Systolic Arrays (for VLSI)," *Proc. Sparse Matrix*, I. S. Duff and G. W. Stewart, ed., *Society for Industrial and Applied Mathematics* (1979), 256-282.

[12] C.E. Leiserson, and J.B. Saxe, "Optimizing Synchronous Systems," *Proc. 22nd Annual Symposium on Foundations of Computer Science*, Nashville, Tennessee, (1981), 23-36.

[13] N.S. Ostlund, WATERLOOP V2/64: A Highly Parallel Machine for Numerical Computation, presented at *Vector and Parallel Processors II*, Oxford, (1984).

[14] A.R. Smith III, "Real-Time Language Recognition by One-Dimensional Cellular Automata," *J. of Computer and System Science, 6*, (1972), 233-253.

[15] H. Umeo, and K. Sugata, "Linear-Time Simulation of Synchronous Cellular Computers," Tec. Rep., TIECE of Japan, EC82-1, (1982), 1-14.

[16] H. Umeo, K. Morita, and K. Sugata, "Deterministic One-Way Simulation of Two-Way Real-Time Cellular Automata and Its Related Problems," *Information Proc Letters, 14*, (1982), 158-161.

[17] J.D. Ullman, *Computational Aspects of VLSI*, Computer Science Press, (1984)