ON SOME EQUIVALENT CONFIGURATIONS OF SYSTOLIC ARRAYS

C.Y. Chang and K. Yao
Electrical Engineering Dept.
University of California
Los Angeles, CA 90024

ABSTRACT

A systematic approach is presented for designing systolic arrays and their equivalent configurations for certain general classes of recursively formulated algorithms. A new method is also introduced to reduce the input bandwidth and storage requirements of the systolic arrays through the study of dependence among the input data. Many well known systolic arrays can be rederived and also many new systolic arrays can be discovered by this approach.

I. INTRODUCTION

A systolic array is a network of processors that rhythmically process and pass data among themselves. It provides pipelining, parallelism, and simple adjacent neighbor cell interconnection structure so that it is suitable for VLSI implementation. While most of the earlier systolic array algorithms were discovered heuristically [1-3], there has been various work on systematic approaches to the design of systolic array algorithms [4-6]. In this paper, we shall present a systematic approach for designing systolic arrays and especially focus on their equivalent configurations for certain general classes of recursively formulated algorithms. In order to reduce the input bandwidth and storage requirements of the systolic arrays, the dependence among the input data is also investigated in details. It is shown that many well known systolic arrays can be rederived and also many new systolic arrays can be discovered by this systematic approach. For simplicity of illustration, we mainly consider the linear systolic array in this paper. The same idea can also be generalized to the two dimensional mesh-connected systolic arrays.

II. IMPLEMENTATION OF RECURSIVELY FORMULATED ALGORITHMS

Consider two simple but important ways of data flow pattern in a linear systolic array as shown in Figure 1 and 2. In these two figures, P_i, Q_j, and b_{ij} are three given input data sequences and R_j is to be the output data sequence, where $0 \le i \le m-1$ and $0 \le j \le n-1$. For the systolic array shown in Figure 1, Q_j and R_j are stored in the j^{th} processor, where R_j will be updated while P_i is moving to the right and b_{ij} is moving down. For the systolic array shown in Figure 2, P_i is stored in the i^{th} processor and R_j will be updated as it is moving to the right with Q_j while b_{ij} is moving down. All of the data movements are synchronized. The R_j's will successively have the required output data after m steps. For convenience, according to the R_j's behavior of these two systolic arrays, they are respectively named as R-stay and R-move linear systolic arrays. There is great similarity between these two systolic arrays. It can be shown that a large class of interesting problems in the real world can be implemented by these two types of linear systolic arrays. Besides, various different but equivalent configurations of linear systolic arrays can also be derived from them.

Procedure 1 : Given any problem which can be formulated so that it has P_i, Q_j, and b_{ij} as three input data sequences and R_j as the output data sequence, where $0 \le i \le m-1$ and $0 \le j \le n-1$, if R_j can be generated through the following recurrence equation

$$R_j^{[i+1]} = f(P_i, Q_j, b_{ij}; R_j^{[i]}), \qquad (1)$$

where $R_j^{[0]}$ contains some initial value, f is any function of four variables P_i, Q_j, b_{ij}, and $R_j^{[i]}$, and $R_j^{[m]}$ is the required output data R_j, then this problem can be implemented by the R-stay linear systolic array of n processors and the R-move linear systolic array of m processors. □

The complexity and the configuration of the systolic array depend on the complexity of the function f and the generation procedure of b_{ij}. Some regularity and dependence among b_{ij}'s may greatly simplify the whole system.

III. MAPPING INTO FAN-IN TYPE LINEAR SYSTOLIC ARRAY

Note that for the two linear systolic arrays shown in Figure 1 and 2, the input bandwidth and storage requirements are large in comparison to the number of processors in the array, which may be either infeasible or inefficient for many applications of interests. This is mainly because the dependence among the b_{ij}'s is not efficiently utilized so that each processor needs its own external input connection due to the existence of all the b_{ij}'s. It is expected that under certain circumstances not all of these external input connections are required. In this paper, we are also very interested in the issue of reducing the input bandwidth and storage requirements by showing under what conditions these external input connections can be removed so that only the very first processor is allowed to have such a connection, i.e., the input sequences can only be fanned in through the systolic array. It is shown that the existence of certain patterns of dependence among the b_{ij}'s allows themselves to be fanned-in generated by slightly modifying the operations involved in each processor without losing the property of adjacent neighbor interconnection structure. These conditions are shown in the following two procedures.

Procedure 2 : For the R-stay linear systolic array, if b_{ij} can be determined through the following dependence equation

$$b_{ij} = T(b_{i-1,j}; b_{i,j-1}; u_i; v_j), \qquad (2)$$

where u_i is a variable which depends only on i, v_j is a variable which depends only on j, and T is a function of four variables, then b_{ij} can be generated by the fan-in scheme systolic array as shown in Figure 3 rather than being broadcast as shown in Figure 1. Also note that $b_{-1,j}$ as well as v_j, which depends only on j, can be preloaded in the j^{th} processor, and $b_{i,-1}$ as well as u_i, which depends only on i can be used as a fanned-in input sequence. □

Note that for the R-stay linear systolic array shown in Figure 1, if b_{ij} is the current input to the j^{th} processor, then $b_{i-1,j}$ is the previous input to the j^{th} processor and $b_{i,j-1}$ is the previous input to the $(j-1)^{st}$ processor. It is understandable that in order to avoid the violation of the adjacent neighbor interconnection structure, b_{ij} can only depend on $b_{i-1,j}$ and $b_{i,j-1}$ as well as the data that can be preloaded and the data that can be fanned in, which is what Procedure 2 is about. In general, the systolic array shown in Figure 3 has two sets of input data. One of them consists of three fanned-in data sequences, P_i, u_i, and $b_{i,-1}$, which depend only on the i index, and the other set consists of three preloaded data sequences, Q_j, v_j and $b_{-1,j}$, which depend only on the j index, where u_i, v_j, $b_{i,-1}$ and $b_{-1,j}$ are used to generate all the b_{ij}'s. For each processor, four registers are required, namely Q_p, V_p, B and R, where registers Q_p and V_p are used to store the preloaded data Q_j and v_j respectively. Initially register B is loaded as $b_{-1,j}$ and register R is set to be $R_j^{[0]}$, both of which will be updated as the systolic array start operation. The reason to include so many data sequences is to take care of the general cases. However, it is expected that in many applications, not all of these fanned-in and preloaded data sequences are required. It is often the case that the fan-in generation process of b_{ij} simply depends on two or three data sequences which can either be fanned-in or preloaded. Similarly for the R-move linear systolic array, very similar results can be obtained as follows.

Procedure 3 : For the R-move linear systolic array, if b_{ij} can be determined through the following dependence equation

$$b_{ij} = T(b_{i-1,j};\ b_{i,j-1};\ u_i;\ v_j), \qquad (3)$$

where u_i is a variable which depends only on i, v_j is a variable which depends only on j, and T is a function of four variables, then b_{ij} can be generated by the fan-in scheme systolic array as shown in Figure 4 rather than being broadcast as shown in Figure 2. Also note that $b_{i,-1}$ as well as u_i, which depends only on i, can be preloaded in the i^{th} processor, and $b_{-1,j}$ as well as v_j, which depends only on j, can be used as a fanned-in input sequence. □

Note that for the R-move linear systolic array shown in Figure 2, if b_{ij} is the current input to the i^{th} processor, then $b_{i,j-1}$ is the previous input to the i^{th} processor and $b_{i-1,j}$ is the previous input to the $(i-1)^{st}$ processor. What procedure 3 says simply repeats the fact that in order to avoid the violation of adjacent neighbor interconnection structure, b_{ij} can only depend on $b_{i-1,j}$ and $b_{i,j-1}$ as well as the data that can be preloaded and the data that can be fanned in. In general, the systolic array shown in Figure 3 has two sets of input data. One of them consists of three fanned-in data sequences, Q_j, v_j, and $b_{-1,j}$, which depend only on the j index, and the other set consists of three preloaded data sequences, P_i, u_i, and $b_{i,-1}$, which depend only on the i index, where u_i, v_j, $b_{i,-1}$ and $b_{-1,j}$ are used to generate all the b_{ij}'s. For each processor, three registers are required, namely U_p, B and P, where registers P and U_p are used to store the preloaded data P_i and u_i. Initially register B is loaded as $b_{i,-1}$ and output data R_j is set to be $R_j^{[0]}$, both of which will be updated as the systolic array start operation.

The previous three procedures provide a rather systematic approach to design the systolic array architecture for the implementation of a given problem. At first, by checking the existence of the recurrence relationship as shown in equation (1), we are able to know if there exist any systolic arrays as shown in Figure 1 and 2. Next, by checking the dependence among the b_{ij}'s as shown in equations (2) and (3), we are able to know the existence of the fan-in type systolic arrays as shown in Figure 3 and 4 so that only small input bandwidth and storage are required. The key issue is in how to search for the recurrence function f and the dependence function T. It is expected that there may exist several different forms of functions due to different possible approaches to formulate a given problem. Various forms of these functions simply create many different but equivalent configurations of systolic arrays. Also note that in the previous discussion, P, Q, b, u, and v are somewhat treated as single variables, however it is clear that they can be set of variables and the same results still hold. This approach can be applied to design systolic arrays for many interesting problems in the real world. Various new configurations of systolic arrays can be derived. In the next section, we shall illustrate this design approach by considering the DFT algorithm.

IV. SYSTOLIC ARRAY ARCHITECTURE FOR DISCRETE FOURIER TRANSFORM

Given n discrete data a_i in the time domain, where $0 \le i \le n-1$, and n discrete frequencies $W_j = (e^{i2\pi/n})^j$ in the frequency domain, where $0 \le j \le n-1$, the discrete Fourier transform (DFT) is to compute

$$y_j = a_{n-1}W_j^{n-1} + a_{n-2}W_j^{n-2} + \ldots + a_1W_j + a_0.$$

Let

$$f(P, Q, b; R) = (R \times b) + P.$$

By induction, it can be shown that by letting

$$y_j^{[i+1]} = (y_j^{[i]} \times W_j) + a_{n-i-2} \qquad (4)$$

and $y_j^{[0]} = a_{n-1}$, then $y_j^{[n-1]} = y_j$, is the required output. The existence of a recurrence function f and the satisfaction of the recurrence relationship guarantee that there exists systolic arrays for the implementation of discrete Fourier transform as shown in Figure 5 and 6.

It can be seen from Figure 5 and 6 that the b_{ij}'s are not totally independent. Note that $P_i = a_{n-i-2}$ and $b_{ij} = W_j$. In order to see if b_{ij} can be fanned-in generated, let us examine the data

dependence among the b_{ij}'s. Many different forms of dependence function T exist. For example,

$$b_{ij} = T(b_{i-1,j}; b_{i,j-1}; u_i; v_j) = v_j. \quad (5)$$

where $v_j = W_j$. The pair of systolic arrays based on equations (4) and (5) are shown in Figure 7 and 8. The systolic array shown in Figure 8 is the well known systolic DFT [2], whose discovery appears to be heuristic rather than in a systematic manner as from our approach. For another example of T function, note that

$$b_{ij} = W_i = W_1^j = W_1^{j-1}W_1 = b_{i,j-1}W_1,$$

i.e.,

$$b_{ij} = T(b_{i-1,j}; b_{i,j-1}; u_i; v_j) = b_{i,j-1}u_i, \quad (6)$$

where $u_i = W_1$ and $b_{i,-1} = W_1^{-1}$, which can be either used as fanned-in sequences of the R-stay linear systolic array or preloaded in the i^{th} processor of the R-move linear systolic array. The pair of systolic arrays based on equations (4) and (6) are shown in Figure 9 and 10.

Another interesting issue is that the type of function f used in this example does not belong to the class of general matrix vector multiplication. This confirm the fact that the class of problems covered in the Procedure 1 really contains not only the class of general matrix vector multiplication. As well known, there are two different ways to consider the discrete Fourier transform. One shows that the DFT is a special case of the evaluation of a polynomial and the other shows that the DFT is a special case of general matrix vector multiplication. The first way was just considered in this example. Let us see what can be obtained by following the second way. Let

$$f(P, Q, b; R) = R + (P \times b).$$

By induction, it can be shown that by letting

$$y_j^{[i+1]} = y_j^{[i]} + (a_i \times W_j^i), \quad (7)$$

and $y_j^{[0]} = 0$, then $y_j^{[n]} = y_j$, is the required output. The existence of a new recurrence function f and the satisfaction of the recurrence relationship guarantee that there exists systolic arrays for the implementation of DFT as shown in Figure 11 and 12.

From Figure 11 and 12 it can also be seen that the b_{ij}'s are not totally independent. Note that $P_i = a_i$ and $b_{ij} = W_j^i$. Let us examine the data dependence among the b_{ij}'s. Note that

$$b_{ij} = W_j^i = W_j^{j \cdot i} = W_i^{j-1}W_i = W_{j-1}^i W_i = b_{i,j-1}W_i^i,$$

i.e.,

$$b_{ij} = T(b_{i-1,j}; b_{i,j-1}; u_i; v_j) = b_{i,j-1}u_i, \quad (8)$$

where $u_i = W_i$ and $b_{i,-1} = W_i^{-1}$, which can be either used as fanned-in sequences of the R-stay linear systolic array or preloaded in the i^{th} processor of the R-move linear systolic array. The pair of systolic arrays based on equations (7) and (8) are shown in Figure 13 and 14. Also note that

$$b_{ij} = W_j^i = W_j^{i-1}W_j = b_{i-1,j}W_j,$$

i.e.

$$b_{ij} = T(b_{i-1,j}; b_{i,j-1}; u_i; v_j) = b_{i-1,j}v_j, \quad (9)$$

where $v_j = W_j$ and $b_{-1,j} = W_j^{-1}$, which can be either preloaded in the j^{th} processor of the R-stay linear systolic array or used as fanned-in sequences of the R-move linear systolic array. The pair of systolic arrays based on equations (7) and (9) are shown in Figure 15 and 16.

This DFT example shows that under certain circumstances it is possible to formulate a given problem in several different ways to implement with various different but equivalent configurations of systolic arrays.

V. CONCLUDING REMARKS

A systematic approach is presented for designing systolic arrays and deriving their equivalent configurations for certain general classes of recursively formulated algorithms. This approach can be considered as a two-stage design procedure. In the first stage, the existence of recursiveness is investigated. If it exists, according to the same formulation the input data are classified into three parts, two of them, P_i and Q_j, depend only on one index, and another one of them, namely b_{ij} depends on both index i and j, so that the systolic arrays shown in Figure 1 and 2 apply. However, for certain applications, it is either infeasible or inefficient to store all of the b_{ij}'s. In the second stage, the dependence among the b_{ij}'s is then investigated to see if it can be used to fan-in generate the b_{ij}'s through the data sequence that can either be preloaded or fanned in. For a given problem, various formulations of the recursive property and the dependence among the b_{ij}'s are possible, which simply lead to many different but equivalent configurations of systolic arrays.

So far we mainly deal with the linear systolic arrays. However, the same technique can be easily generalized to the two dimensional mesh-connected systolic arrays, since the mesh-connected systolic arrays can be simply treated as the concatenation of many linear systolic arrays.

VI. ACKNOWLEDGEMENT

This work was partially supported by the NASA/Ames research contract NAG-2-304.

VII. REFERENCES

1. H. T. Kung and C. E. Leiserson, 'Systolic Arrays (for VLSI),' Proc. Symp. Sparse Matrix Computations and Their Applications, Nov. 2-3, 1978, pp. 256-282.
2. H. T. Kung, 'Why Systolic Architectures,' Computer, Jan. 1982, pp. 37-45.
3. H. T. Kung, 'Let's design algorithms for VLSI systems,' in Proc. Caltech Conf. on VLSI, pp. 65-90, Jan. 1979.
4. P. R. Cappello and K. Steiglitz, 'Unifying VLSI Array Design with Geometric Transformations,' Proc. Int. Conf. on Parallel Processing, pp. 448-457, Bellaire, Michigan, Aug. 1983.

5. D. I. Moldovan, 'On the Design of Algorithms for VLSI Systolic Arrays,' Proc. IEEE, V 71, N 1, pp. 113-120, Jan 1983.
6. W. L. Miranker and A. Winkler, 'Spacetime Representations of Computational Structures,' Computing, V 32, 1984.

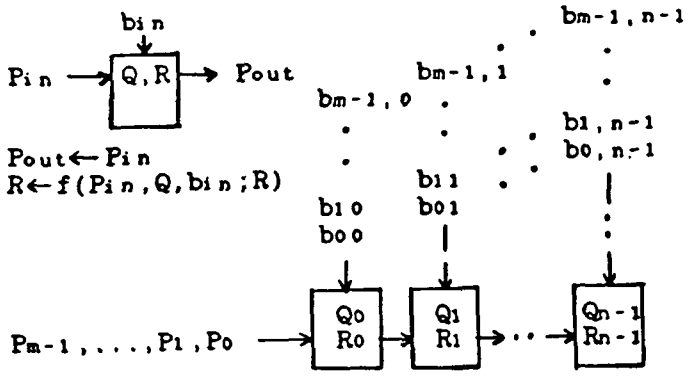

Figure 1: The R-stay linear systolic array.

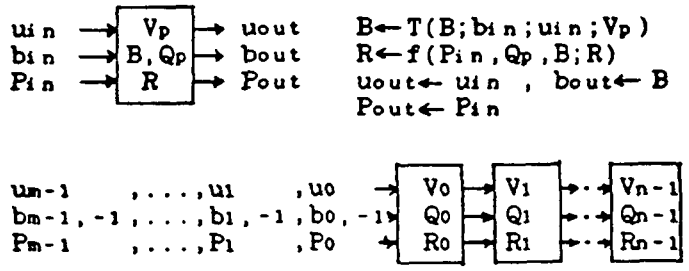

Figure 3: The fan-in scheme of R-stay linear systolic array. Note that the register B in the j^{th} processor is initially loaded with $b_{-1,j}$.

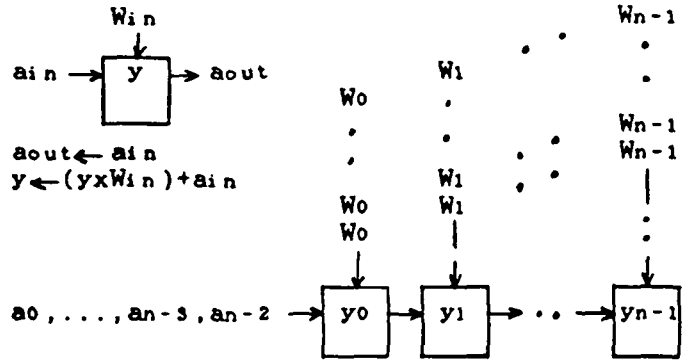

Figure 5: R-stay linear systolic array of discrete Fourier transform based on equation (4).

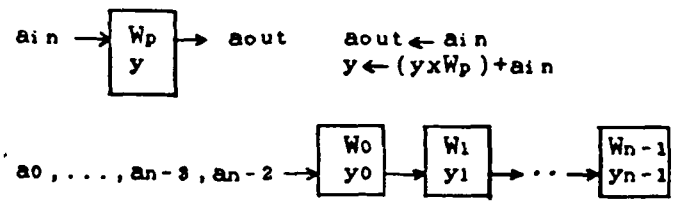

Figure 7: R-stay linear systolic array of discrete Fourier transform based on equations (4) and (5).

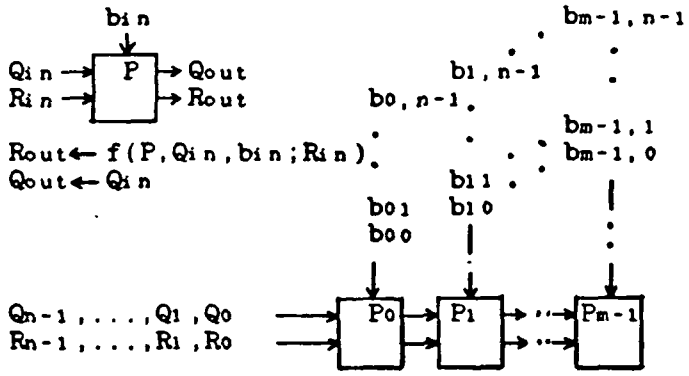

Figure 2: The R-move linear systolic array.

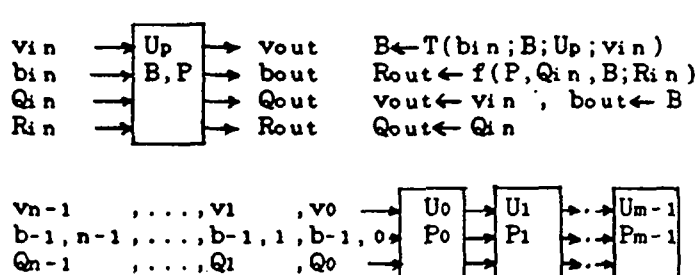

Figure 4: The fan-in scheme of R-move linear systolic array. Note that the register B in the i^{th} processor is initially loaded with $b_{i,-1}$.

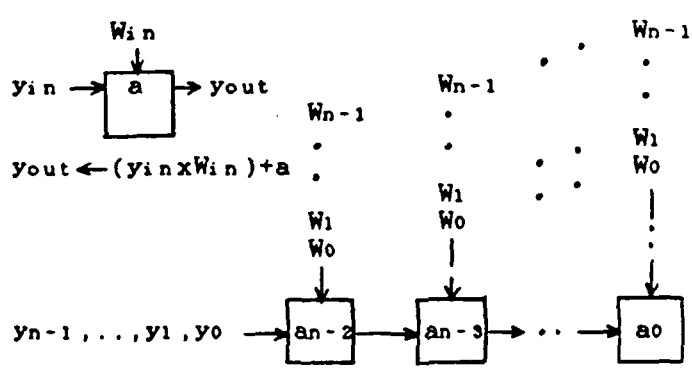

Figure 6: R-move linear systolic array of discrete Fourier transform based on equation (4).

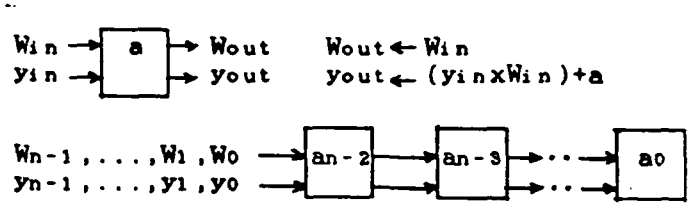

Figure 8: R-move linear systolic array of discrete Fourier transform based on equations (4) and (5)

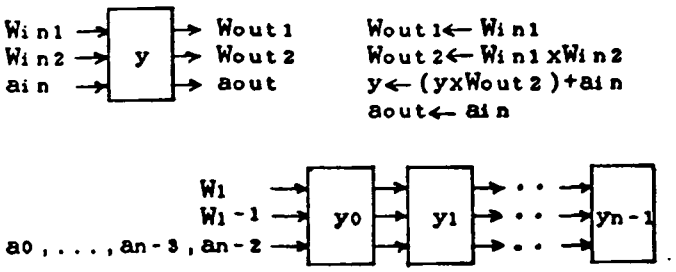

Figure 9: R-stay linear systolic array of discrete Fourier transform based on equations (4) and (6).

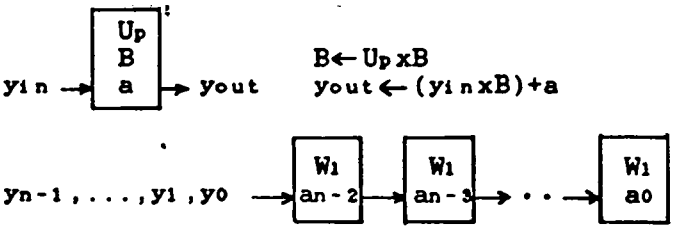

Figure 10: R-move linear systolic array of discrete Fourier transform based on equations (4) and (6). Note that register Up is preloaded with W_i and register B is initially loaded with W_i^{-1}.

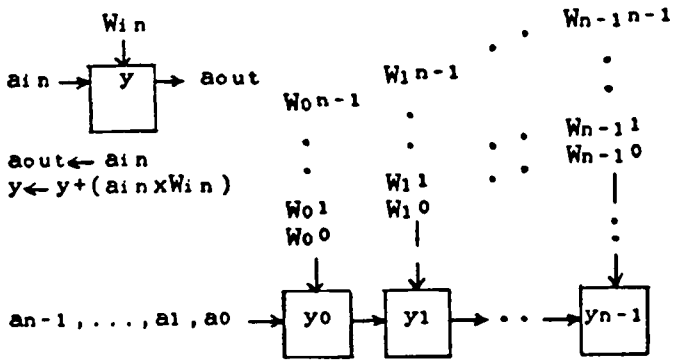

Figure 11: R-stay linear systolic array of discrete Fourier transform based on equation (7).

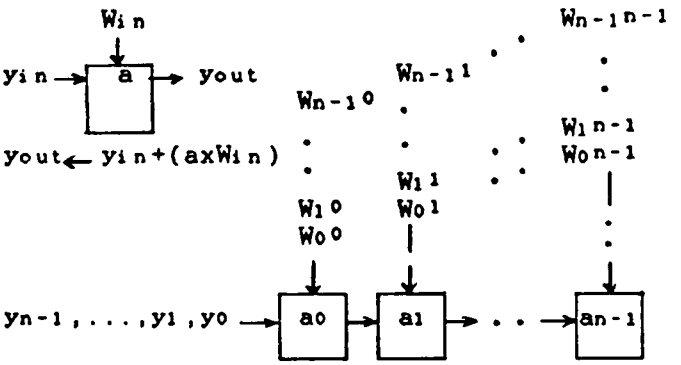

Figure 12 : R-move linear systolic array of discrete Fourier transform based on equation (7).

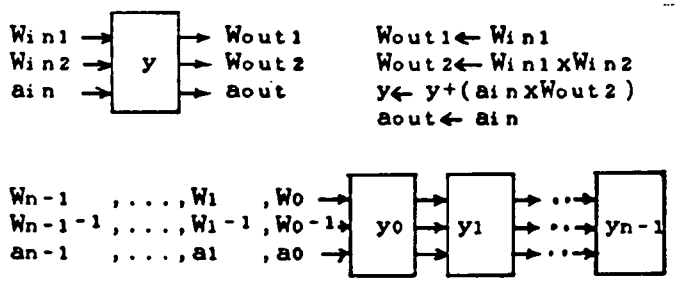

Figure 13: R-stay linear systolic array of discrete Fourier transform based on equation (7) and (8).

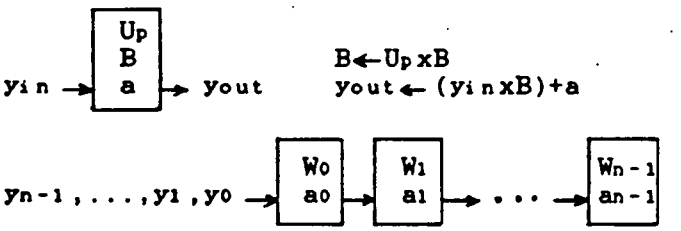

Figure 14: R-move linear systolic array of discrete Fourier transform based on equations (7) and (8). Note that in the i^{th} processor, register Up is preloaded with W_i and register B is initially loaded with W_i^{-1}.

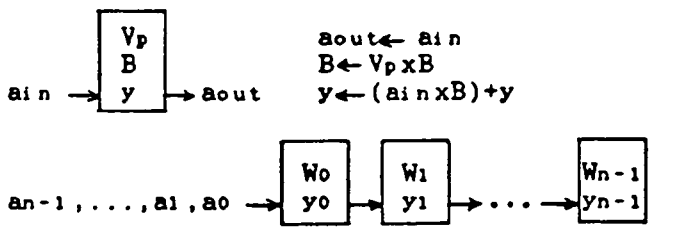

Figure 15: R-stay linear systolic array of discrete Fourier transform based on equations (7) and (9). Note that in the j^{th} processor, register Vp is preloaded with W_j and register B is initially loaded with W_j^{-1}.

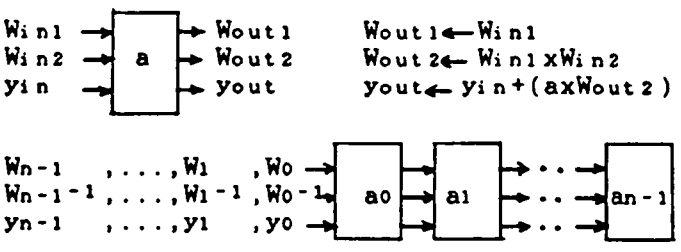

Figure 16: R-move linear systolic array of discrete Fourier transform based on equations (7) and (9).