SYSTOLIC ARCHITECTURES

FOR

PARALLEL IMPLEMENTATION

OF

DIGITAL FILTERS

by

Zhou Bing Bing

A thesis submitted to the

Australian National University

for the degree of Doctor of Philosophy

September, 1988

PREFACE

Some of the work of this thesis was carried out in collaboration with Professor Richard Brent. In particular, Chapters 4 and 5 contain results which were established jointly.

Elsewhere in the thesis, unless otherwise stated, the work described is my own.

Zhou Bing Bing

ACKNOWLEDGEMENTS

I am indebted to Professor Richard Brent for his invaluable guidance throughout the course of my research, and for his constructive criticisms of this thesis. In writing the thesis I was also greatly assisted by Dr. Iain Macleod who suggested many clarifications and corrections in the text. This work would have been impossible without their help.

I am grateful to Dr. Heiko Schröder for his helpful advice. I would also like to thank all the staff of the Computer Sciences Laboratory for their friendship during the completion of this work.

I would also like to record my deep appreciation for the support and encouragement of my wife Zheng Yong Min during my research.

Finally I acknowledge with gratitude the financial support received from the Chinese Government and the Australian National University.

ABSTRACT

This thesis addresses the design of efficient systolic architectures for real-time digital filters.

Many VLSI architectures for digital filters have been introduced in the literature. Most of these architectures are one-dimensional because the problems to be solved are 1-D. Although 1-D architectures have the advantage of low I/O bandwidth requirement, the throughput is limited by their word-serial nature. In many high-throughput applications, (two-level) pipelined and/or (2-D) parallel architectures have to be considered seriously.

Digital filters can be classified as non-recursive filters and recursive filters. In order to obtain efficient pipelined and/or parallel systolic architectures for these two types of filters, some difficult problems have to be solved. Suppose that N_1 and N_2 are the numbers of coefficients and inputs of a non-recursive filter. Since there is no recursion in the system and there is a high degree of inherent parallelism, it is easy to obtain an $N_1 \times N_2$ architecture for implementing that filter. However, this $N_1 \times N_2$ architecture is impractical for implementation in VLSI because it has too many input/output lines and requires too large an area for a large problem. Therefore, the key problem is to construct 2-D systolic architectures which not only can solve the problem efficiently, but have a reasonable size well suited for VLSI implementation.

Recursion implies sequential rather than parallel execution, which typically places an upper bound on the throughput with which a recursion can be implemented. Therefore, to obtain parallel algorithms/architectures with guaranteed

stability for recursive filters is a more difficult task than for non-recursive filters. This thesis aims to solve these problems and then to obtain efficient pipelined and/or parallel systolic architectures for non-recursive and recursive filters.

We introduce a 2-D systolic architecture for linear phase non-recursive filters and two 2-D systolic ring structures for linear convolution problems. These architectures are based on nearest neighbour interconnections and have the property of linear area versus period tradeoff for a given problem. Moreover, they are unidirectional structures. The technique of two-level pipelining may easily be applied to these structures. Thus the throughput can further be increased without a great increase in area.

Although look-ahead computation is a good method for deriving parallel algorithms for state-variable-form recursive filters, this conventional technique may cause numerical instability in direct-form recursive filters due to the effect of finite wordlength. We introduce a new method of Z domain derivation. Using this method, not only can parallel algorithms with guaranteed stability be derived, but the additional complexity required for this purpose can be minimized through a decomposition technique. A time domain derivation of parallel algorithms for direct-form recursive filters is also introduced. The derived algorithms are of particular interest for time-varying recursive systems.

Using the stabilized parallel algorithms for direct-form recursive filters, very efficient pipelined and/or parallel VLSI architectures can be constructed. We show that those algorithms lead directly to an efficient two-level pipelined structure. Two different parallel systolic structures are also derived based on those algorithms. One has the advantage of regularity while the other can achieve a linear

complexity in parallel size for cascaded second-order recursive filters. Using 2-D parallel processing in combination with two-level pipelining, efficient pipelined and parallel architectures can also be constructed. With the same degree of complexity, this combination of techniques enables a substantial increase in throughput compared to purely parallel architectures for a given problem.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

This thesis addresses the design of efficient architectures for high-speed digital signal processing.

Many signal processing applications have two characteristics which make them computationally intensive: (i) large amounts of processing (up to hundreds of elementary operations at each point); (ii) real-time response (at rates up to several MHz). Ever-increasing demands for sophistication and speed of processing clearly point to the need for dramatic increases in computational capability. Rapid improvements in digital integrated circuit technology have provided about a ten times increase in device speed every few years, but these improvements still cannot supply the required performance in areas of high-speed signal processing such as speech recognition, beamforming, and image analysis. Therefore, real-time computation needs accompanying architectural advances.

Many compute-intensive tasks are now handled by general-purpose supercomputers. These general-purpose machines have complicated system organizations, are very expensive and provide facilities such as 64-bit word and arithmetic units which are not fully utilized in signal processing operations. The advent of low-cost and high-density VLSI technology makes it possible to use special-purpose array processors for solving certain classes of compute-intensive problems faster and more economically than would be possible with a general-purpose system.

In VLSI, communication is expensive whereas logic is relatively cheap [31]. To minimize communication cost, one can build architectures which are regular ar-

rays or lattices of small processing elements. The only communication is between near neighbours and the array can be laid out simply and efficiently. Fortunately, most algorithms in modern signal processing possess some useful common properties, such as regularity, local data communication, and a high degree of potential parallelism. These properties may be utilized to exploit the power of VLSI and to circumvent its constraints. Because of its local communication, synchronous data flow, simple control and modular parallelism with throughput directly proportional to the number of cells, the systolic array [18,20] has been considered as a most promising VLSI architecture for real-time signal processing. In the last few years, there has been a dramatic worldwide growth in research on mapping various signal processing algorithms into systolic arrays.

Digital filtering is one of the most important techniques in modern signal processing. It is widely used in areas such as speech and image processing, radar signal processing and biomedical engineering. While this technique is compute-intensive, it is well suited to VLSI implementation and many examples have been reported in the literature.

This thesis concentrates on the design of efficient systolic architectures for high-throughput digital filters. The next two sections of this chapter briefly review systolic architectures and VLSI implementation of digital filters. The final section then gives an outline of the following chapters, together with a summary of the main results obtained.

1.1. Systolic Arrays

Systolic architectures are typically large, regular arrays with only a few different types of small processing elements, each capable of performing some simple operation. Data and control flow in the system is simple and regular. All operations involving a data item are applied to it as it passes through the array; communication with the outside world occurs only at the "boundary cells". This method of computation eliminates the need to retrieve a data item from external memory every time it is used. The regular pattern of local interconnections between cells implies that the systolic design can be made modular and extensible, so that a systolic array can easily be expanded to achieve high computation throughput without increased memory bandwidth. This property gives systolic architectures a major advantage over traditional architectures, which are limited by the "Von Neumann bottleneck". Moreover, the simplicity of systolic designs is particularly important for special-purpose applications, where design costs must be kept low.

Because of their low memory bandwidth, simplicity, regularity, modularity and local communication, systolic architectures are well suited to implementation in VLSI. Various systolic systems have been developed to solve compute-intensive problems in areas such as signal and image processing and matrix arithmetic, for example see [1,4,5,12-14,17-22,24-26,37,44,45,50,52,56,57]. However, it should be noted that clock skew (which may inhibit global synchronization for ultra-large-scale 2-D arrays) is one possible disadvantage of the systolic array approach. A simple solution to this problem is to adopt the principle of dataflow computing in systolic array processors, which leads to wavefront array processors [24-26].

In a wavefront array instructions cannot be executed until their operands have become available. The arrival of data from neighbouring processors will be interpreted as a change of state and will initiate some action. The wavefront arrays are named because of the analogy with wavefront propagation, and comprise a distributed and (globally) asynchronous array processing system. This approach replaces the requirement for correct timing by one of correct sequencing, and handles the data dependency locally. Thus, it eliminates the need for global control and global synchronization [25]. Since systolic arrays can easily be converted into wavefront array processors, in this thesis we regard the concept of systolic architectures as encompassing wavefront array processors.

Systolic architectures can be either one-dimensional (1-D) or two-dimensional (2-D) depending on the problems to be solved. Sometimes a given problem can have both 1-D and 2-D systolic array solutions. 1-D systolic arrays have the advantage of low I/O bandwidth requirement. However, throughput is limited by their word-serial nature. To achieve higher throughput, two-level pipelined and/or 2-D parallel structures have to be considered. The two-level pipelined systolic approach, first introduced by H. T. Kung and his colleagues [19,21], is a good method not only for achieving high throughput computation, but also for reducing the area required in VLSI implementation in comparison with other parallel approaches. Thus it is desirable that a given system is implemented by first using two-level pipelining to the maximum possible extent, and then using 2-D parallel processing in combination with pipelining if further increase in throughput is required.

1.2. VLSI Implementation of Digital Filters

In this thesis we study VLSI architectures for linear filters. Digital filters can be classified as non-recursive filters and recursive filters.

Non-recursive filters

One of the most important characteristics of non-recursive filters is that they can be designed to have exactly linear phase [38]. There have been many signal flow graph (SFG) networks introduced to compute linear phase non-recursive filters [38]. 1-D systolic arrays can easily be obtained by applying cut-set localization rules [24-26] to these SFG computing networks. (One example is described in [24].) Suppose that the number of coefficients for a given problem is N_1, which is assumed to be even. The derived 1-D systolic array will use $N_1/2$ multipliers. We shall derive a 2-D systolic array for solving the same problem by using N_1 multipliers. However, this 2-D architecture can achieve twice the throughput of the 1-D array. Thus our 2-D implementation is preferable in high-speed signal processing.

We also discuss 2-D systolic architectures for computing 1-D linear convolution equations. This is not only because the problem of non-recursive filtering can be represented by a linear convolution equation, but because other problems such as DFT, circular convolution and 2-D linear convolution can also be transformed into 1-D linear convolution [11,22,35]. Thus one structure can be used for several different kinds of problems.

Although various types of systolic architectures for computing linear convolution problems have been introduced in the literature [8,17,18,21,22,25,30,43,46],

only a few of them are 2-D. Suppose that N_1 and N_2 are respectively the numbers of coefficients and inputs of a given problem. Cappello and Steiglitz [8] derived several $N_1 \times N_2$ 2-D systolic arrays. In theory, these systolic arrays are AP^2 asymptotically optimal (where A is defined as Area and P as Period, which is the reciprocal of throughput), but they are impractical for implementation in VLSI because they have too many input/output lines and require too large an area for a large problem. Lu and others [30] derived an $L \times N_1$ 2-D systolic array for the case $L < N_2$. They split the linear convolution equation into a number of subequations. Each subequation contains L successive outputs. Because the subproblems have the form of matrix-vector multiplications, they can be implemented successively on a matrix-vector multiplier. In this design the throughput is increased by a factor of L in comparison with 1-D systolic arrays. However, a large proportion of the total area is consumed by the communication lines. Thus this implementation is not area efficient in VLSI. We shall introduce two $N_1 \times L$ systolic ring architectures. These $N_1 \times L$ architectures are based on nearest neighbour interconnections and can achieve the same throughput as the $L \times N_1$ array, but are more efficient in their use of area.

Recursive filters

Recursion implies sequential execution, which typically places an upper bound on the throughput with which a recursion can be implemented. Therefore, conventional serial algorithms for recursive filters cannot effectively make use of large numbers of processing elements. To overcome this problem, Gold and Jordan [16] proposed block (or parallel) recursion methods for implementing recursive filters. They demonstrated that rational transfer functions can be realized by finite con-

volutions employing blocks of data. Subsequently, Voelcker and Hartquist [47] showed that a rational transfer function can be realized with block feedback and finite convolution. Burrus [6,7] developed a different approach based on a matrix representation of convolution, which results in a state-variable description with block feedback. The relationship between the Gold and Jordan idea and the Burrus method was described by Gnanasekaran and Mitra [15], who also proposed several new block structures using a matrix representation of convolution [34]. Meyer and Burrus [32,33] discussed the use of block implementations for multirate and periodically time-varying digital filters. Note that block implementations for state-variable-form recursive filters have recently received a lot of attention [2,3,30,34,36,39-41,48,53]. These implementations are based on the technique of so-called look-ahead computation. (Although this terminology was first formally described in [39,41], the technique had been known previously for some time.) In the look-ahead technique, the given recursion is iterated as many times as desired to create the necessary concurrency; the concurrency created can then be used to obtain pipelined and/or parallel implementations of recursive systems [41].

Although the look-ahead computation technique has been applied successfully to the parallel implementation of state-variable-form recursive filters, it may cause numerical instability in direct-form recursive filters due to the effect of finite wordlength. We shall introduce new methods for deriving parallel algorithms for direct-form recursive filters. In comparison with the original serial algorithm, not only is the degree of parallelism increased, but the stability is also improved. Moreover, these algorithms lead to very efficient systolic/wavefront architectures. Therefore, they are most suitable for VLSI.

1.3. Thesis Outline and Main Results

In Chapter 2, the signal-flow graph (SFG) notations to be used throughout this thesis are described. A very powerful architecture transformation technique based on cut-set localization rules (or systolization rules) and introduced by S. Y. Kung [24-26] is described. Then we introduce a commutative rule. Using cut-set localization rules in combination with this commutative rule, one may obtain more efficient systolic arrays from certain feedforward SFG computing networks.

Chapter 3 describes 2-D systolic architectures for non-recursive digital filters. Suppose that N_1 and N_2 are respectively the numbers of coefficients and inputs of a linear filtering problem. It is easy to obtain $N_1 \times N_2$ architectures for implementing that problem because there is no recursion in the system. However, as mentioned in the previous section these architectures are impractical for VLSI implementation. Therefore, the key problem is to construct 2-D systolic architectures which not only can solve the problem efficiently, but also have a reasonable size well suited for implementation in VLSI. We derive a 2-D systolic array for linear phase non-recursive filters. By using twice the number of multipliers that 1-D systolic arrays normally require, this 2-D systolic architecture can achieve twice the throughput of 1-D arrays for a given problem. We introduce two $N_1 \times L$ 2-D systolic ring structures, where $L < N_2$. These 2-D structures are based on nearest neighbour interconnections and can achieve L times the throughput of 1-D systolic arrays for solving the same linear convolution problem. Moreover, the most efficient 1-D systolic array for linear convolution is just a special case of a 2-D systolic ring array with $L = 1$. The complexity measure AP^2 is also analyzed.

Chapter 4 derives stabilized parallel algorithms for direct-form recursive fil-

ters. The conventional look-ahead computation may cause numerical instability in the direct-form implementation. Thus we introduce a new method for deriving, in the Z domain, a class of stabilized parallel algorithms for direct-form recursive filters. In order to obtain this stabilized algorithm extra zeros and poles are introduced, so the complexity is greatly increased. We then introduce a decomposition technique to minimize this increase in complexity. We also introduce a time-domain derivation of parallel algorithms, which have the same form as those derived in the Z domain. This is of particular interest for time-varying recursive systems. There are many different ways to achieve the same goal. The condition for the unique solution of the stabilized parallel algorithm is discussed.

Chapter 5 introduces efficient pipelined and/or parallel architectures associated with the stabilized parallel algorithms derived in Chapter 4. We show that the stabilized parallel algorithms lead directly to an efficient two-level pipelined structure. Using these algorithms, we can also derive two different parallel structures. The first one has the advantage of regularity while the second one can achieve a linear complexity in parallel size for cascaded second-order recursive filters. Combining parallel processing with pipelining we finally describe two efficient pipelined and parallel architectures for direct-form recursive filters. This combination of techniques enables a substantial increase in throughput compared to previously known architectures.

CHAPTER 2

CUT–SET LOCALIZATION RULES

2.1. Introduction

A non-systolic system can be transformed into a systolic array by using architecture transformation techniques. The original work was done by Leiserson and Saxe [27,28]; they introduced various rules such as retiming and slowdown and showed how these rules can be used to obtain an efficient systolic array from a given synchronous system. H. T. Kung and Lam [19] derived a cut theorem to show how to add delays to a given system without affecting its function. Because the main purpose of this theorem is to construct a two-level pipelined systolic array from a uni-directional structure, it is required that all the lines in the cut must point in the same direction. Thus further applications to more complicated systems are limited. Cut-set localization rules (or "systolization" rules) were introduced by S. Y. Kung [24]. By using these simpler, but more powerful rules, one can easily transform a signal flow graph (SFG) computing network into a systolic array. It has also been proved in [24] that all computable SFG arrays can be temporally localized by using these rules.

Although an SFG computing network (or synchronous system) can be converted into a systolic array by using the methods metioned above, we cannot guarantee that the derived system is the most efficient systolic array for solving the given problem. The reason is that the chosen SFG computing network may be inefficient. A method for converting a dependence graph into an efficient SFG computing network was introduced in [25]. Since a dependence graph may as-

sociate with many different SFGs, we may choose the most efficient one among them [25].

In the following, we first describe the SFG notations and the systolization rules. Then we introduce a commutative rule for obtaining efficient systolic arrays from certain SFG computing networks without feedback. We give two examples to show that by applying the commutative law of addition, the direction of some lines in this kind of system may be changed without affecting the overall function. More efficient systolic arrays can thus be obtained after the modification.

2.2. SFG Notations and Cut-Set Localization Rules

SFG notations

The SFG notations used herein are similar to those in [24]. They are described as follows: A node is denoted by a square (or a circle) representing an arithmetic or logic function performed with zero delay. On the other hand, a line denotes a delay. When a line is labeled with a number t (a non-negative integer), it represents a time delay operator with a delay time t. We omit the number when it is zero. Therefore, a line without a number represents a zero delay operator. For convenience we sometimes use another notation for a time delay operator; a time delay operator with a delay time t ($t \neq 0$) is denoted by a small square (smaller than a node) with a number t in it. The two equivalent representations are depicted in Fig. 2.1.

Fig. 2.1. Equivalent notations for a time delay operator

Cut-set localization rules

There are only two basic rules:

1. Time-scaling: All delays in a system may be scaled by a single positive integer α. Correspondingly, the input and output rates also have to be scaled by a factor α.

2. Delay-transfer: Given any cut-set of the SFG, we can group the lines in the cut-set into in-bound lines and out-bound lines, depending on the directions assigned to the lines. Rule 2 allows an advance of k time units on all the out-bound lines and a delay of k time units on the in-bound lines, and vice versa, provided the resulting delays are non-negative.

The proof of cut-set localization rules and many examples of deriving systolic array structures from SFG computing networks can be found in [24-26].

2.3. Commutative Rule

From cut-set localization rules we see that the throughput rate of the derived systolic array is inversely proportional to the scaling factor α. The optimal localization algorithm [26] is then proposed to search non-rescaling (NR) cut-sets so that α can be minimized. An NR cut-set is a "good" cut-set in which all the lines (excluding input lines) in the opposite direction to the zero-delay target line to be localized, have delays of at least two time units. (A "bad" cut-set is one in which this condition does not hold.) Once an NR cut is determined, one can simply apply the delay-transfer operation along the cut and localize the target line(s). If there exist no NR cut-sets in the original SFG computing network, time-scaling has to be applied, which will decrease the throughput of the system. In this section we give two examples to show how to avoid application of time-scaling to SFGs without feedback by using the commutative rule.

2.3.1. Linear convolution

A well known SFG computing network for linear convolution problems [18] is depicted in Fig. 2.2. It is easy to see that there exist no NR cut-sets in the structure. Time-scaling has to be applied and the throughput of the resulting systolic array is then decreased.

To avoid use of time-scaling, we use the following transformation procedure. First we apply delay transfer to the SFG, as shown in Fig. 2.4(a). We see that each of the computation steps has now been divided into two stages; first all nodes or cells perform multiplications, and then these multiplication terms are accumulated from right to left as one of the outputs. Note that the order of additions is

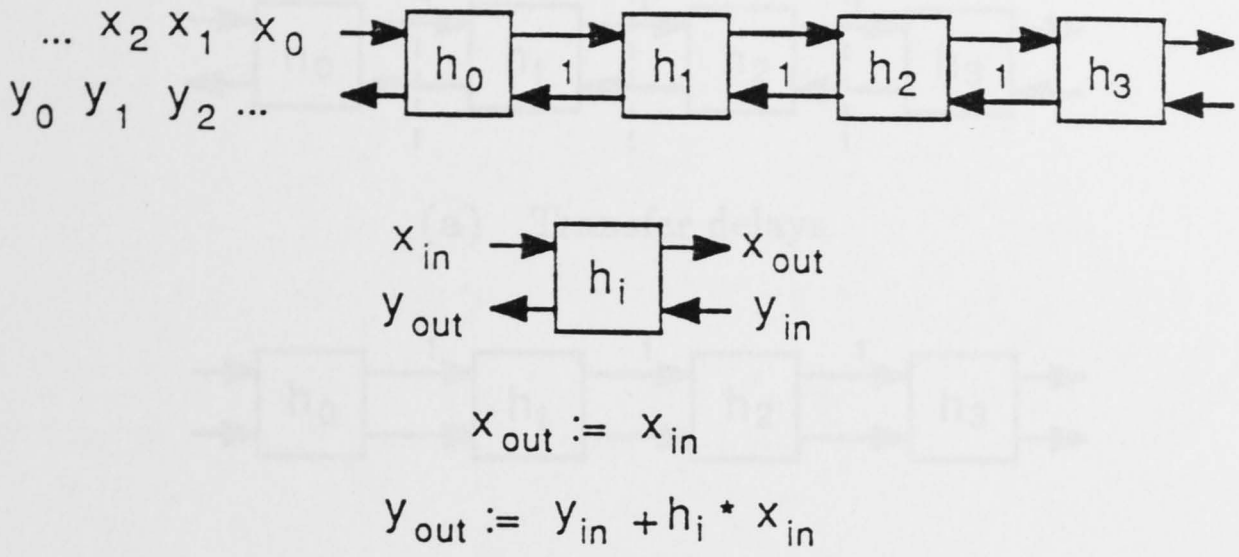
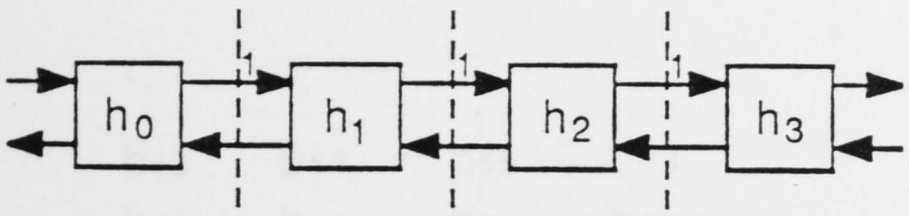


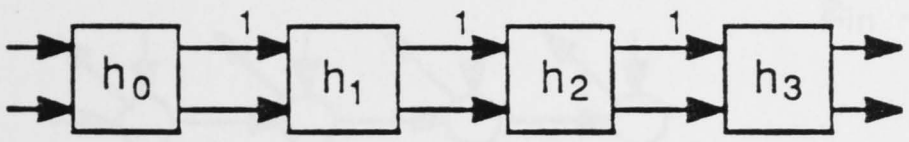
Fig. 2.2. SFG array for linear convolution

Fig. 2.3. Systolic array for linear convolution in [27]

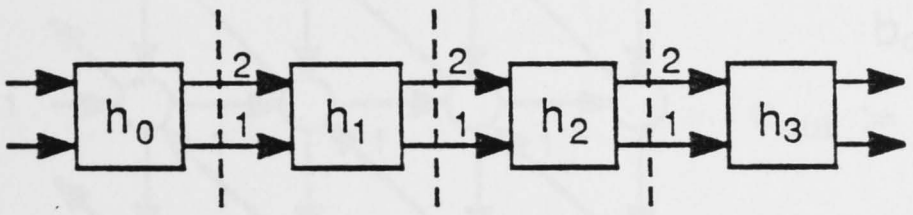
changeable according to the commutative law. Therefore, the multiplication terms can also be accumulated from left to right. This means that we can change all left-pointing lines into right-pointing lines without affecting the function of the system. Thus we next change the direction of the left-pointing lines in Fig. 2.4(a). Since all the lines are now pointing in one direction, we can introduce a set of new cuts and add one delay to each of the lines in the cuts. The final result is the same as that described in [18] and has been proved to be the most efficient among 1-D systolic arrays for solving linear convolution problems [29].



(a) Transfer delays



(b) Change the direction of left-pointing lines



(c) Add delays

Fig. 2.4. Transformation steps for obtaining an efficient systolic array from Fig. 2.2

2.3.2. Multiplication of two band matrices

Consider the problem of multiplying two band matrices as follows:

$$\begin{pmatrix} c_{11} & c_{12} & c_{13} & c_{14} & 0 \\ c_{21} & c_{22} & c_{23} & c_{24} & \\ c_{31} & c_{32} & c_{33} & c_{34} & \\ c_{41} & c_{42} & \cdot & & \\ & & & \cdot & \\ 0 & & & & \cdot \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & & & 0 \\ a_{21} & a_{22} & a_{23} & & \\ a_{31} & a_{32} & a_{33} & & \\ & a_{42} & \cdot & & \\ & & & \cdot & \\ 0 & & & & \cdot \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} & b_{13} & & 0 \\ b_{21} & b_{22} & b_{23} & b_{24} & \\ & b_{32} & b_{33} & b_{34} & \\ & & \cdot & & \\ & & & \cdot & \\ 0 & & & & \cdot \end{pmatrix}$$

Fig. 2.5 depicts an SFG network for computing the above matrix multiplication. This structure was originally reported by Chern and Murata [9]. The corresponding systolic array, as shown in Fig. 2.6, was derived from Fig. 2.5 by using cut-set localization rules [24]. The derived systolic array is not very efficient

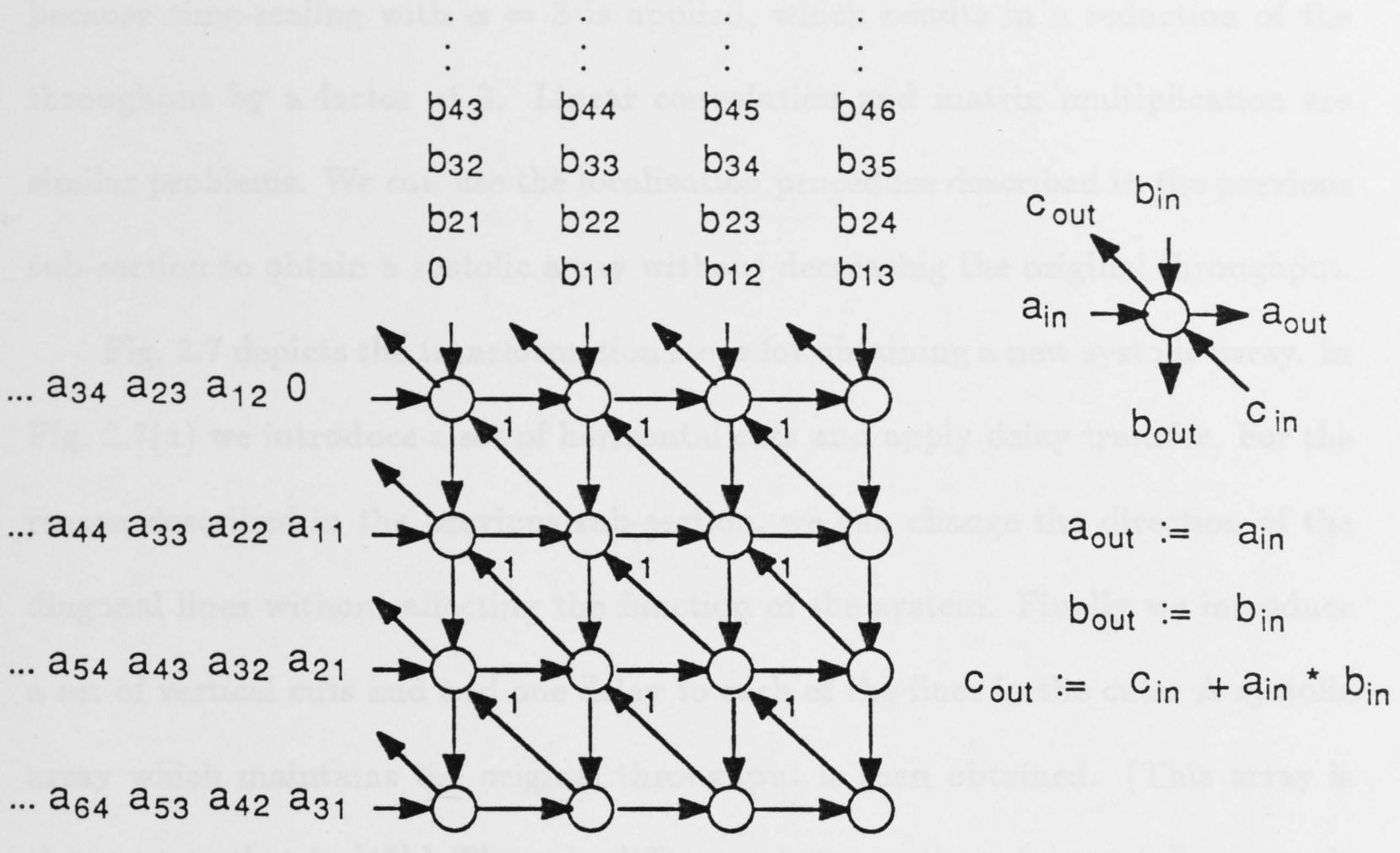


Fig. 2.5. SFG array for multiplication of two band matrices

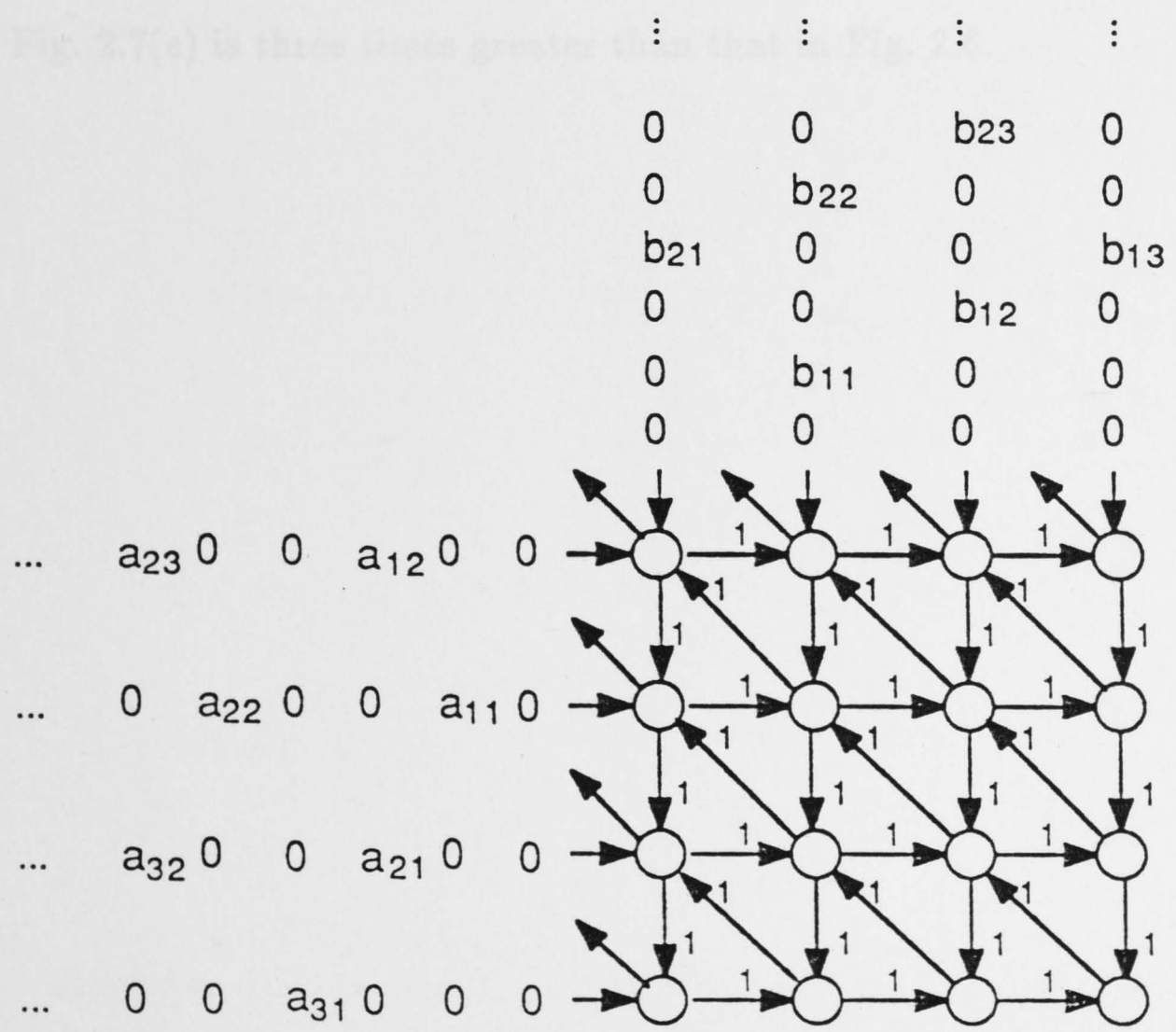
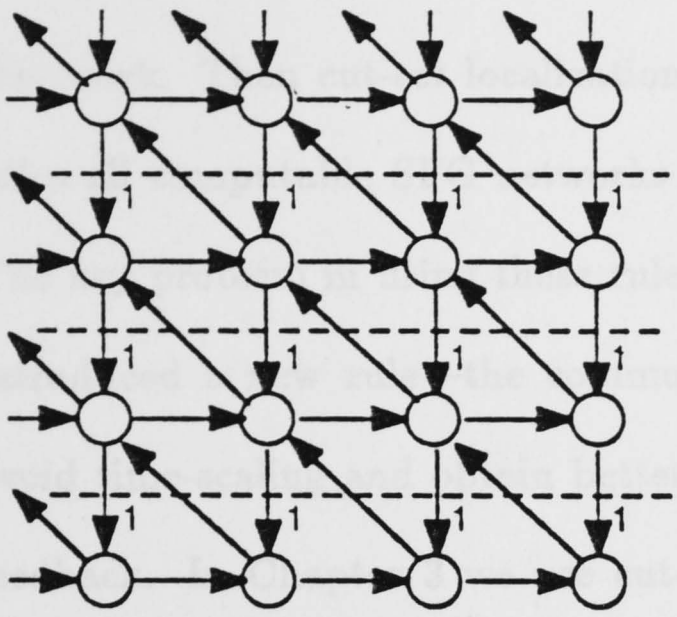


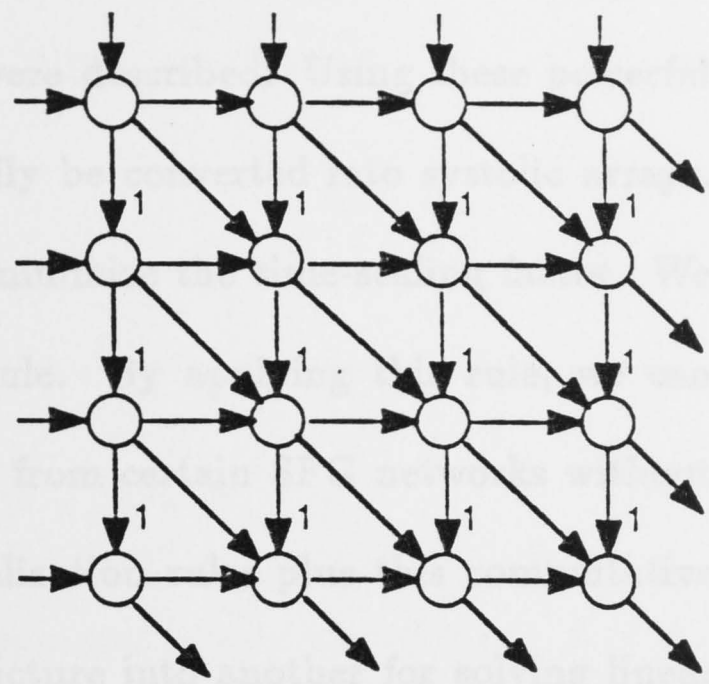
Fig. 2.6. Systolic array for multiplication of two band matrices in [24]

because time-scaling with $\alpha = 3$ is applied, which results in a reduction of the throughput by a factor of 3. Linear convolution and matrix multiplication are similar problems. We can use the localization procedure described in the previous sub-section to obtain a systolic array without decreasing the original throughput.

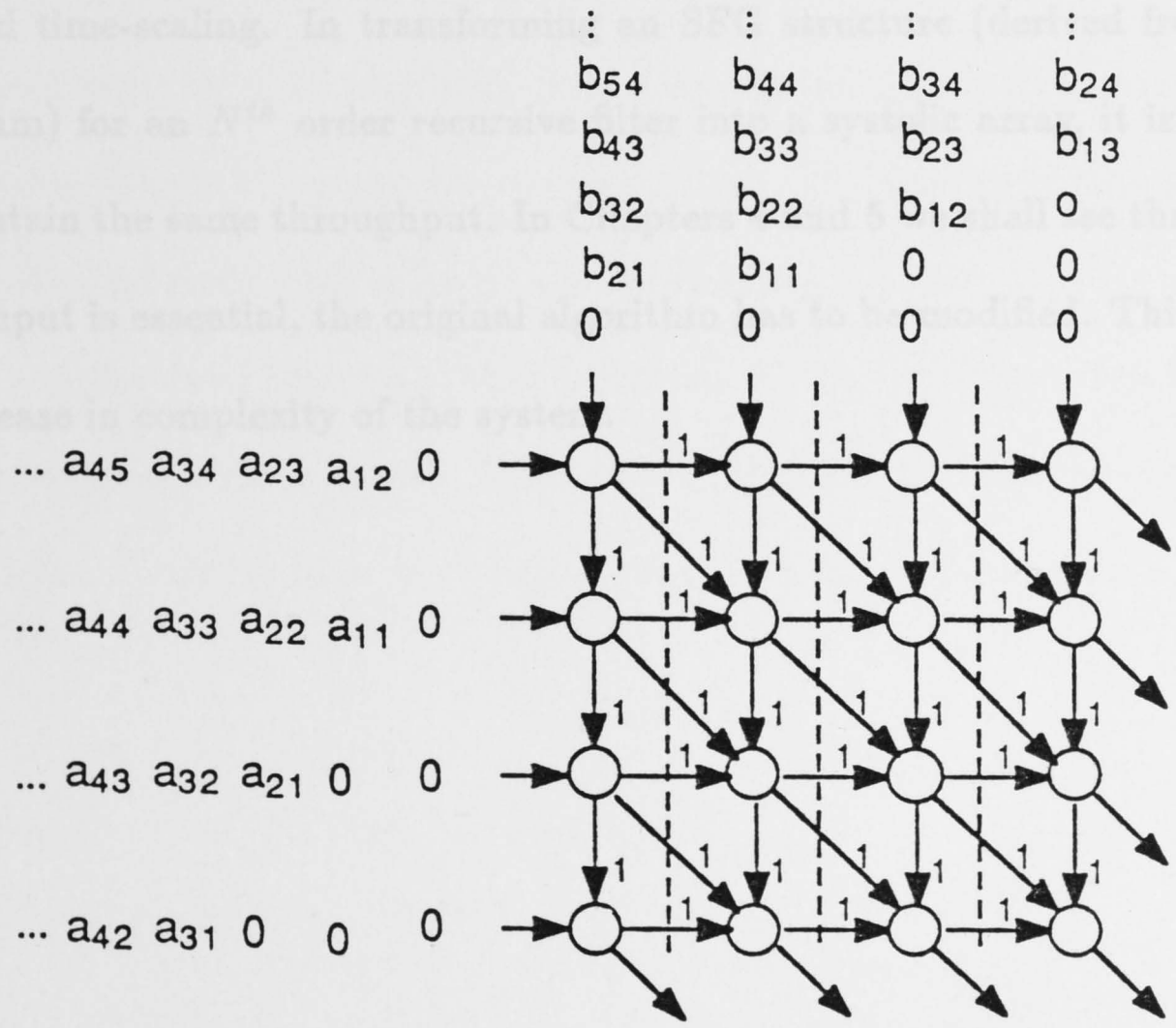
Fig. 2.7 depicts the transformation steps for obtaining a new systolic array. In Fig. 2.7(a) we introduce a set of horizontal cuts and apply delay transfer. For the reason described in the previous sub-section, we can change the direction of the diagonal lines without affecting the function of the system. Finally we introduce a set of vertical cuts and add one delay to each of the lines in the cuts. A systolic array which maintains the original throughput is then obtained. (This array is the same as that in [49].) The only difference between these two systolic arrays is that all diagonal lines are reversed in direction, but the throughput of the array in Fig. 2.7(c) is three times greater than that in Fig. 2.6.



(a) step 1



(b) step 2



(c) step 3

Fig. 2.7. Transformation steps for obtaining an efficient systolic array from Fig. 2.5

2.4. Discussion

In this chapter we first introduced the SFG notation to be used throughout this work. Then cut-set localization rules were described. Using these powerful rules all computable SFG networks can easily be converted into systolic arrays. The key problem in using these rules is to minimize the time-scaling factor. We introduced a new rule—the commutative rule. By applying this rule, we can avoid time-scaling and obtain better results from certain SFG networks without feedback. In Chapter 3 we use cut-set localization rules plus this commutative rule to transform one 2-D systolic ring structure into another for solving linear convolution problems. For SFG networks with feedback, however, it is not so easy to avoid time-scaling. In transforming an SFG structure (derived from a serial algorithm) for an N^{th} order recursive filter into a systolic array, it is impossible to maintain the same throughput. In Chapters 4 and 5 we shall see that, if a high throughput is essential, the original algorithm has to be modified. This will cause an increase in complexity of the system.

CHAPTER 3

2-D SYSTOLIC IMPLEMENTATIONS FOR NON-RECURSIVE DIGITAL FILTERS

3.1. Introduction

A non-recursive filter, or a linear convolution problem, can be expressed as

$$y_n = \sum_{i=0}^{N_1-1} h_i x_{n-i}, \quad 0 \le n \le N_1 + N_2 - 2 \tag{3.1}$$

where N_1 and N_2 are the numbers of coefficients and inputs respectively. This is a very important computational problem in modern signal processing. To perform this computation at high speed, various kinds of systolic architectures have been introduced [8,17,18,21,22,24,43,46]. Since most of these architectures are 1-D, the throughput is limited by the word-serial nature. To achieve higher throughput, 2-D word-parallel structures have to be considered.

One of the most important characteristics of non-recursive filters is that they can be designed to have exactly linear phase [38]. The finite impulse response for a causal non-recursive filter with linear phase has the property that

$$h_n = h_{N_1-1-n} \tag{3.2}$$

Therefore, equation (3.1) can be rewritten as

$$y_n = \sum_{j=0}^{N_1/2-1} h_j (x_{n-j} + x_{n-(N_1-1-j)}) \tag{3.3}$$

where we assume that N_1 is even.

A 1-D systolic array can easily be obtained by applying cut-set localization rules to SFG structures corresponding to equation (3.3). (One example is described

in [24].) The derived systolic structure uses $N_1/2$ multipliers. In section 3.2 we derive a 2-D systolic array for solving the same problem. By using N_1 multipliers, our 2-D systolic array can achieve twice the throughput of the 1-D array. Thus this 2-D implementation is preferable for high speed signal processing.

The concept of a systolic ring was first introduced by H. T. Kung and Lam [19]. Section 3.3 introduces two $N_1 \times L$ 2-D systolic ring structures for solving linear convolution problems in (3.1), where $L < N_2$. These $N_1 \times L$ 2-D systolic ring structures are based on nearest neighbour interconnections and can achieve the same throughput rate as the $L \times N_1$ structure described in [30]. The complexity measure AP^2 for this systolic ring is inversely proportional to L, but AP is independent of L. By varying L we can then trade off area versus period for a given problem. We shall see that the most efficient 1-D systolic array for solving linear convolution problems is just the special case of a 2-D systolic ring structure with $L = 1$.

3.2. Linear Phase Non-Recursive Filters

In this section we derive a 2-D systolic array with twice the throughput of the 1-D systolic version for linear phase non-recursive filters. We only consider the case when N_1 is even. The derivation for the filter with N_1 odd is similar.

The basic idea for constructing this 2-D systolic array is as follows. The equation in (3.3) can be arranged into two groups, one containing the outputs with even subscripts and the other those with odd subscripts—

$$\begin{cases} y_{2p} = \sum_{i=0}^{N_1/2-1} h_i(x_{2p-i} + x_{2p-(N_1-1-i)}) \\ y_{2p+1} = \sum_{i=0}^{N_1/2-1} h_i(x_{2p+1-i} + x_{2p+1-(N_1-1-i)}) \end{cases} \tag{3.4}$$

It is easy to see that the above two equations can be computed independently. Therefore, we may use two sub-systems to solve these two problems in parallel. If each sub-system can achieve the same throughput as that of the 1-D systolic version, the throughput of the entire system for the complete problem is then doubled.

Since the two equations in (3.4) are similar, we can visualize that the structures for solving these equations are also similar. Therefore, we just give a detailed description for the structure for computing y_{2p}.

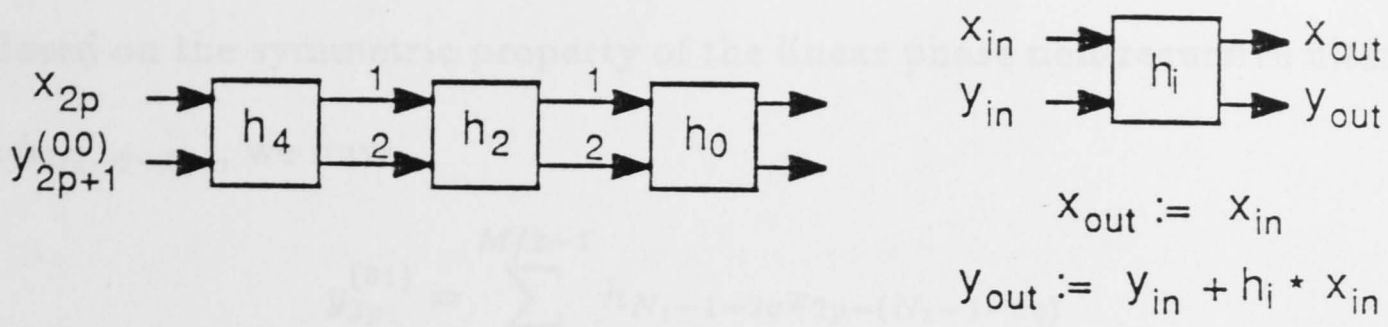


Fig. 3.1. Linear systolic array for computing $y_{2p}^{(00)}$ $(N_1 = 6)$

3.2.1. Sub-system for computing y_{2p}

We divide the equation for y_{2p} into two sub-equations as

$$\begin{cases} y_{2p}^{(0)} = \sum_{q=0}^{M/2-1} h_{2q}(x_{2p-2q} + x_{2p-(N_1-1-2q)}) \\ y_{2p}^{(1)} = \sum_{q=0}^{M/2-1} h_{2q+1}(x_{2p-(2q+1)} + x_{2p-(N_1-1-(2q+1))}) \end{cases} \tag{3.5}$$

where $M = N_1/2$.

The superscript 0 (or 1) in (3.5) indicates that the equation contains only those coefficients with even (or odd, respectively) subscripts.

Again the two sub-equations in (3.5) have similar structures, so we only consider $y_{2p}^{(0)}$.

$y_{2p}^{(0)}$ in (3.5) consists of two parts

$$\begin{cases} y_{2p}^{(00)} = \sum_{q=0}^{M/2-1} h_{2q} x_{2p-2q} \\ y_{2p}^{(01)} = \sum_{q=0}^{M/2-1} h_{2q} x_{2p-(N_1-1-2q)} \end{cases} \tag{3.6}$$

where the second component of the superscript being 0 (or 1) denotes that only the inputs with even (or odd) subscripts are required for solving that problem.

It is easy to see that the equation for $y_{2p}^{(00)}$ is a linear convolution problem which can be implemented in a 1-D systolic array, as shown in Fig. 3.1.

Based on the symmetric property of the linear phase non-recursive filter (i.e., $h_{2q} = h_{N_1-1-2q}$), we have

$$y_{2p}^{(01)} = \sum_{q=0}^{M/2-1} h_{N_1-1-2q} x_{2p-(N_1-1-2q)} \tag{3.7}$$

Thus the equation for $y_{2p}^{(01)}$ is also a linear convolution problem. Instead of using the array depicted in Fig. 3.1, however, we use another well known 1-D systolic array [18] to compute $y_{2p}^{(01)}$, as shown in Fig. 3.2(a). Using the symmetric property again, we obtain Fig. 3.2(b). The reasoning is as follows: If $y_{2p}^{(01)}$ is implemented as in Fig. 3.1, the coefficients should be placed in the opposite order. These two arrays cannot easily be combined into one array. We would then have to use $N_1/2$ multipliers for computing the equations for $y_{2p}^{(00)}$ and $y_{2p}^{(01)}$. But the equation for $y_{2p}^{(0)}$ in (3.5) shows that only $N_1/4$ multipliers are required for the problem.

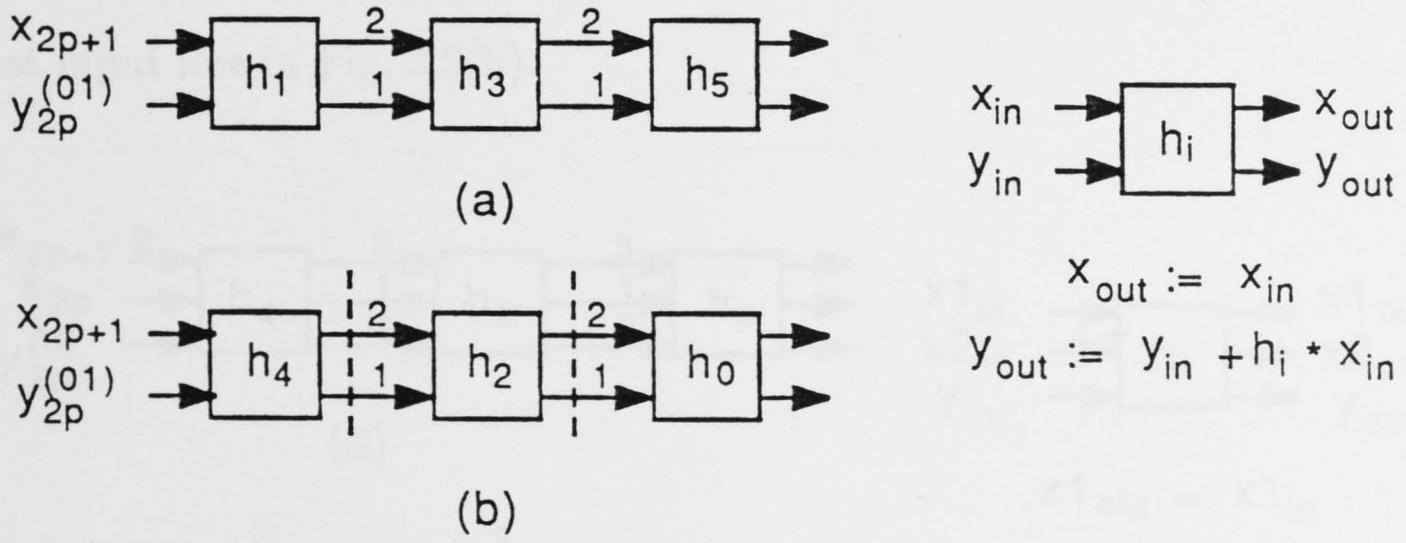


Fig. 3.2. Two equivalent systolic arrays for computing $y_{2p}^{(01)}$ $(N_1 = 6)$

Although the coefficients in Fig. 3.1 and Fig. 3.2(b) are in the same order, there are two problems to be solved before they can be combined into one array with $N_1/4$ multipliers for computing $y_{2p}^{(0)}$. One problem is that the outputs moving from one cell to the next in the two arrays do not travel at the same speed. The

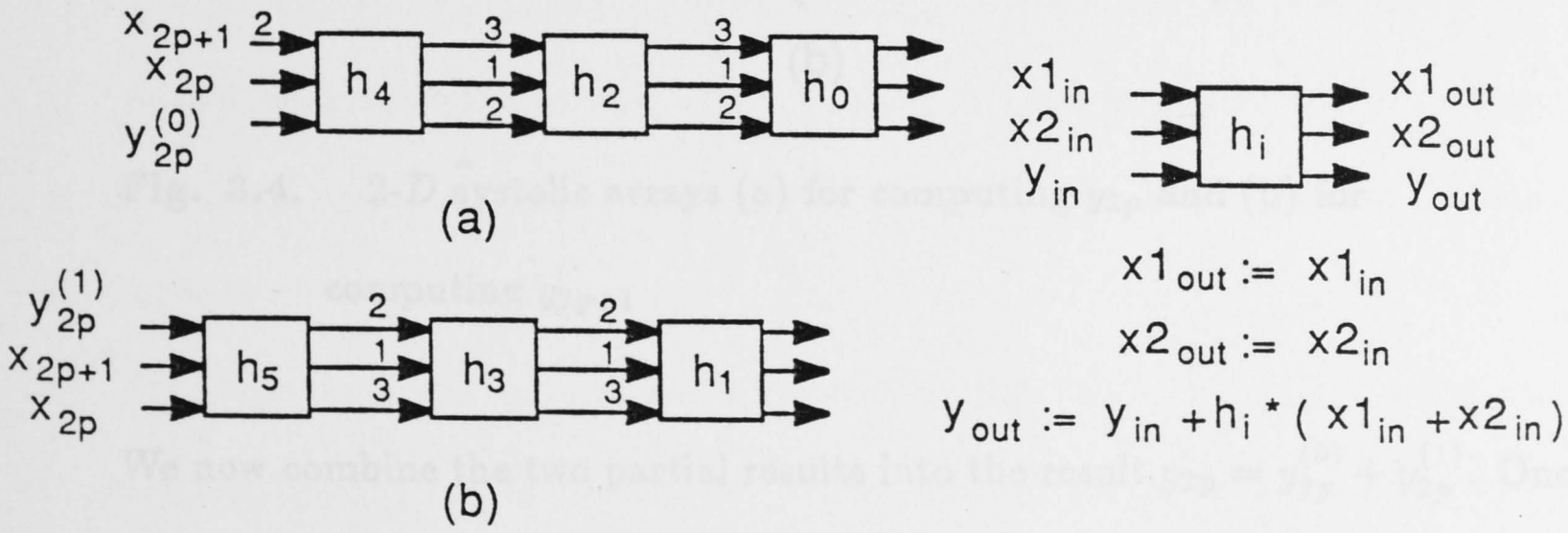
other is that the inputs to these arrays do not accord with the demands of the equation for $y_{2p}^{(0)}$ in (3.5).

From Fig. 3.1 we see that $y_{2p}^{(00)}$ needs two time units for travelling from one cell to the next, while $y_{2p}^{(01)}$ in Fig. 3.2(b) requires only one time unit. Therefore, to solve the first problem we apply a set of cuts in Fig. 3.2(b) and add one delay to each line in the cuts.

To solve the second problem, we consider the difference between the two inputs in (3.5). Let D be the difference. Then

$$\begin{aligned} D &= 2p - 2q - (2p - (N_1 - 1 - 2q) \\ &= N_1 - 1 - 4q \end{aligned} \tag{3.8}$$

Since the outputs in the arrays travel at the same speed after the first modification, we need only to consider D in the leftmost cells. Substituting $q = M/2 - 1$ into (3.8), we then obtain $D = 3$. This can be done by adding extra two delays to the leftmost input line in Fig. 3.2(b).



(b)

Fig. 3.3. Systolic arrays (a) for computing $y_{2p}^{(0)}$ and (b) for $y_{2p}^{(1)}$

After the above two modifications, we can combine these two arrays into one for computing $y_{2p}^{(0)} = y_{2p}^{(00)} + y_{2p}^{(01)}$ with $N_1/4$ multipliers. The combined array is

depicted in Fig. 3.3(a).

Since the equation for $y_{2p}^{(1)}$ has a similar structure to the one for $y_{2p}^{(0)}$, we can use the same method described above to construct an array for computing $y_{2p}^{(1)}$. (See Fig. 3.3(b).)

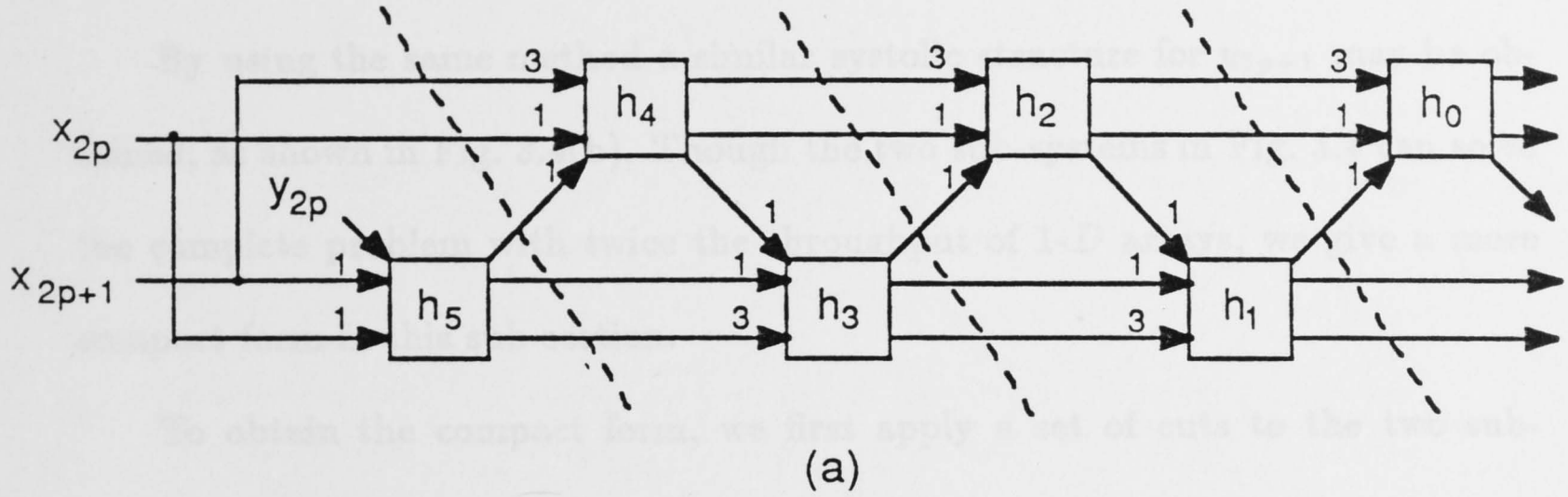


Fig. 3.4. 2-*D* systolic arrays (a) for computing y_{2p} and (b) for computing y_{2p+1}

We now combine the two partial results into the result $y_{2p} = y_{2p}^{(0)} + y_{2p}^{(1)}$. One way of doing this is to place an additional adder at the right ends of the two arrays in Fig. 3.3. Observing that the outputs in each array take two time units for moving from one cell to the immediately next, we can also accumulate the output y_{2p} by letting it move diagonally between these two arrays. Therefore, not

only one adder, but also about $N_1/2$ shift registers can be saved. The array is depicted in Fig. 3.4(a). It is clear that this array requires only $N_1/2$ multipliers for computing y_{2p}.

3.2.2. Compact form

By using the same method a similar systolic structure for y_{2p+1} may be obtained, as shown in Fig. 3.4(b). Though the two sub-systems in Fig. 3.4 can solve the complete problem with twice the throughput of 1-D arrays, we give a more compact form in this sub-section.

To obtain the compact form, we first apply a set of cuts to the two sub-systems in Fig. 3.4. It can be seen that all the lines in the cuts are pointing in one direction. Thus we can add one delay to each line in the cuts. The modified version is depicted in Fig. 3.5. The blank square in the figure denotes a null cell. It passes data through it, but has no other function. It is easy to see, from Fig. 3.5, that the inputs x_{2p} and x_{2p+1} move through both sub-systems in exactly the same manner and the basic processing elements in Fig. 3.5(a) take the position of the null cells in Fig. 3.5(b), and vice versa. If we put them together, the functions of the two sub-systems will not be changed. Therefore, the final compact form is obtained, as shown in Fig. 3.6.

Although the idea described above can be extended to achieve a higher throughput, regular systolic structures cannot be constructed because long communication lines in the system are required. This problem requires further study.

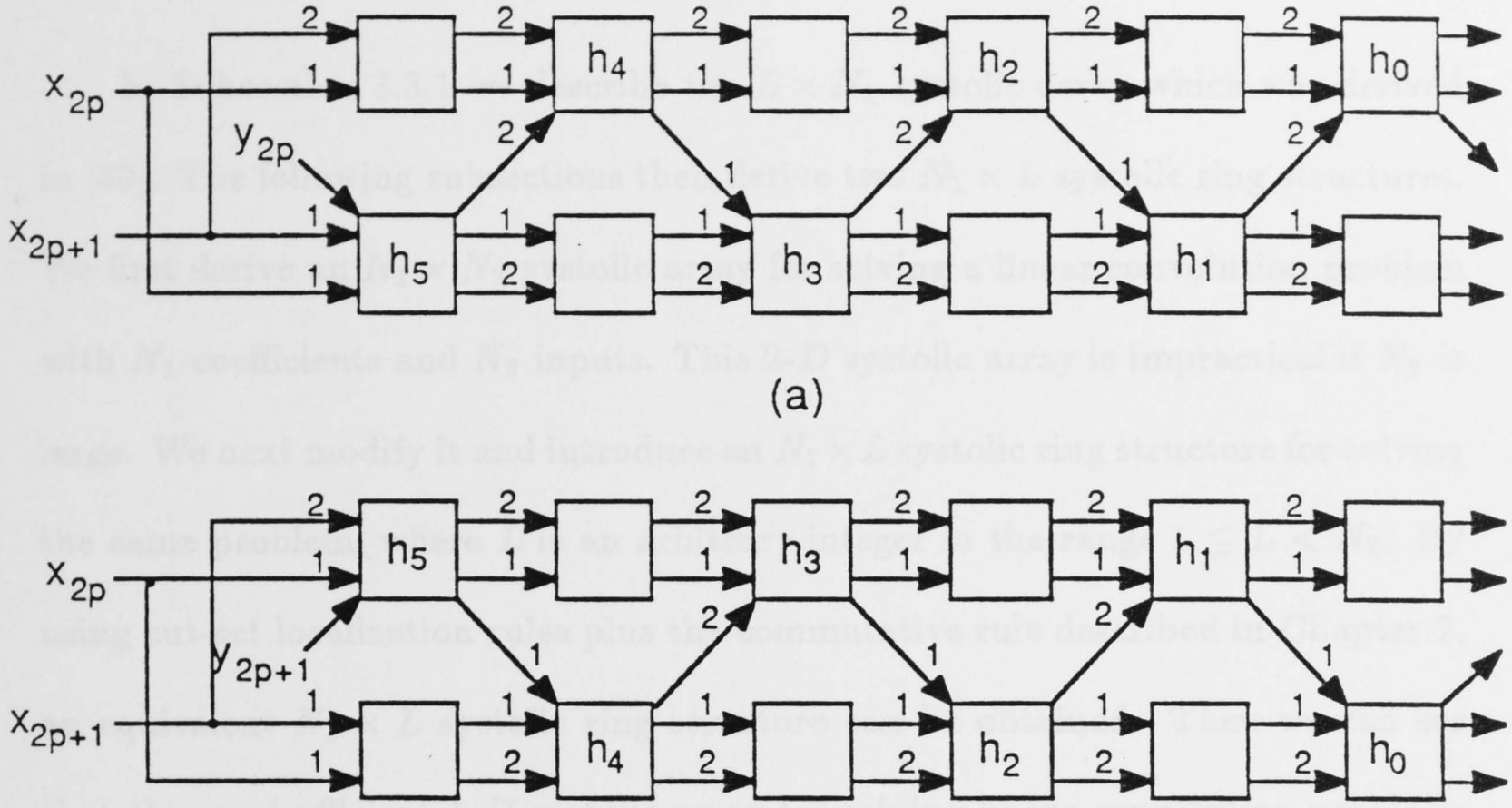


Fig. 3.5. The modified version of Fig. 3.4

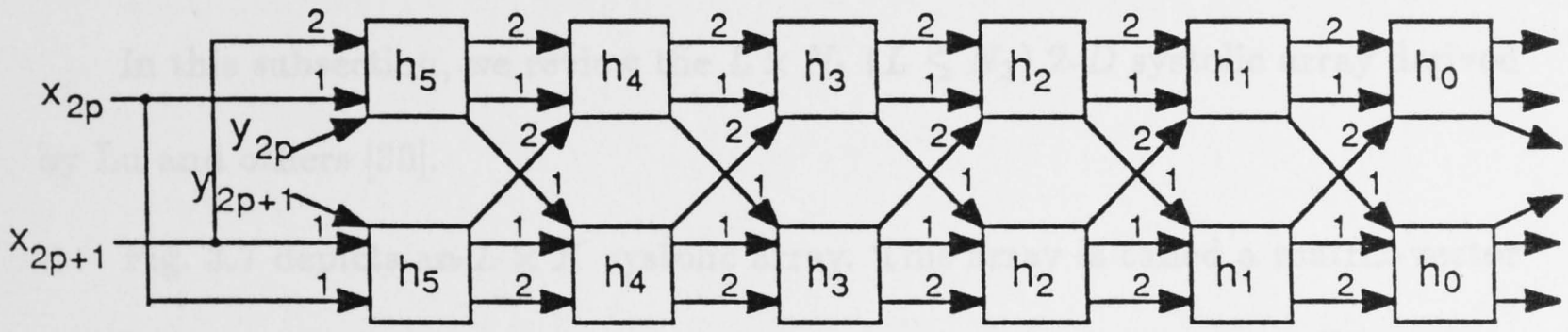


Fig. 3.6. 2-*D* systolic array for a linear phase filter

3.3. Linear Convolution Problems

In Subsection 3.3.1 we describe the $L \times N_1$ systolic array which was derived in [30]. The following subsections then derive two $N_1 \times L$ systolic ring structures. We first derive an $N_1 \times N_2$ systolic array for solving a linear convolution problem with N_1 coefficients and N_2 inputs. This 2-D systolic array is impractical if N_2 is large. We next modify it and introduce an $N_1 \times L$ systolic ring structure for solving the same problem, where L is an arbitrary integer in the range $1 \leq L < N_2$. By using cut-set localization rules plus the commutative rule described in Chapter 2, an equivalent $N_1 \times L$ systolic ring structure can be obtained. Then we can see that the most efficient 1-D systolic array for solving linear convolution problems is just a special case of this 2-D array with $L = 1$. An upper bound on AP^2 for these ring structures is also presented in the following discussion.

3.3.1. $L \times N_1$ 2-D systolic array

In this subsection, we review the $L \times N_1$ $(L \leq N_2)$ 2-D systolic array derived by Lu and others [30].

Fig. 3.7 depicts an $L \times K$ systolic array. This array is called a matrix-vector multiplier because it can perform a matrix-vector multiplication $\mathbf{y} = \mathbf{Hx}$, where $\mathbf{H}$ is an $L \times K$ matrix and $\mathbf{y}$ and $\mathbf{x}$ are $L \times 1$ and $K \times 1$ vectors respectively. The matrix $\mathbf{H}$ is prestored in the array. While the input data travel vertically from top to bottom and remain unchanged, the outputs accumulate their terms horizontally and obtain the final results at the right side of the array. If the same $\mathbf{H}$ can be used for computing a number of problems, then the period for this array becomes equal to one.

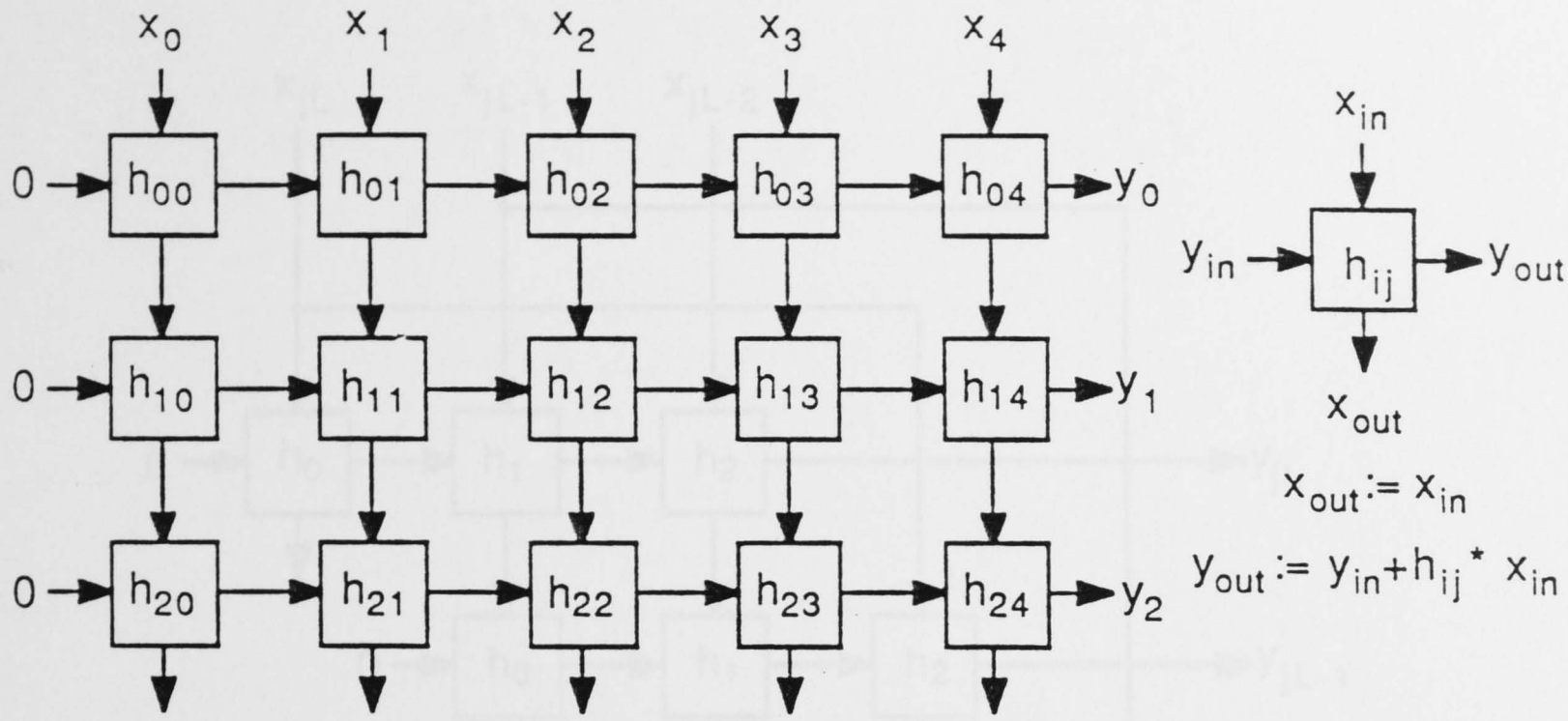


Fig. 3.7. The matrix-vector multiplier

Use the notation $\mathbf{x}_n^{(K)}$ to denote a $K \times 1$ vector, that is,

$$\mathbf{x}_n^{(K)} = \begin{pmatrix} x_n \\ x_{n-1} \\ \vdots \\ x_{n-K+1} \end{pmatrix}. \tag{3.9}$$

The linear convolution problem in (3.1) can then be expressed as

$$y_n = \mathbf{h}^T \mathbf{x}_n^{(N_1)} \tag{3.10}$$

where $\mathbf{h}^T$ is the transpose of the coefficient vector and $\mathbf{x}_n^{(N_1)}$ is a vector of N_1 successive input data items.

To achieve a high throughput rate, we can compute L outputs simultaneously by using the following equation

$$\mathbf{y}_{jL}^{(L)} = \mathbf{H}\mathbf{x}_{jL}^{(N_1+L-1)} \tag{3.11}$$

where $j = 0, 1, 2, 3 \cdots$ and $\mathbf{H}$ is an $L \times (N_1 + L - 1)$ banded Toeplitz matrix. For

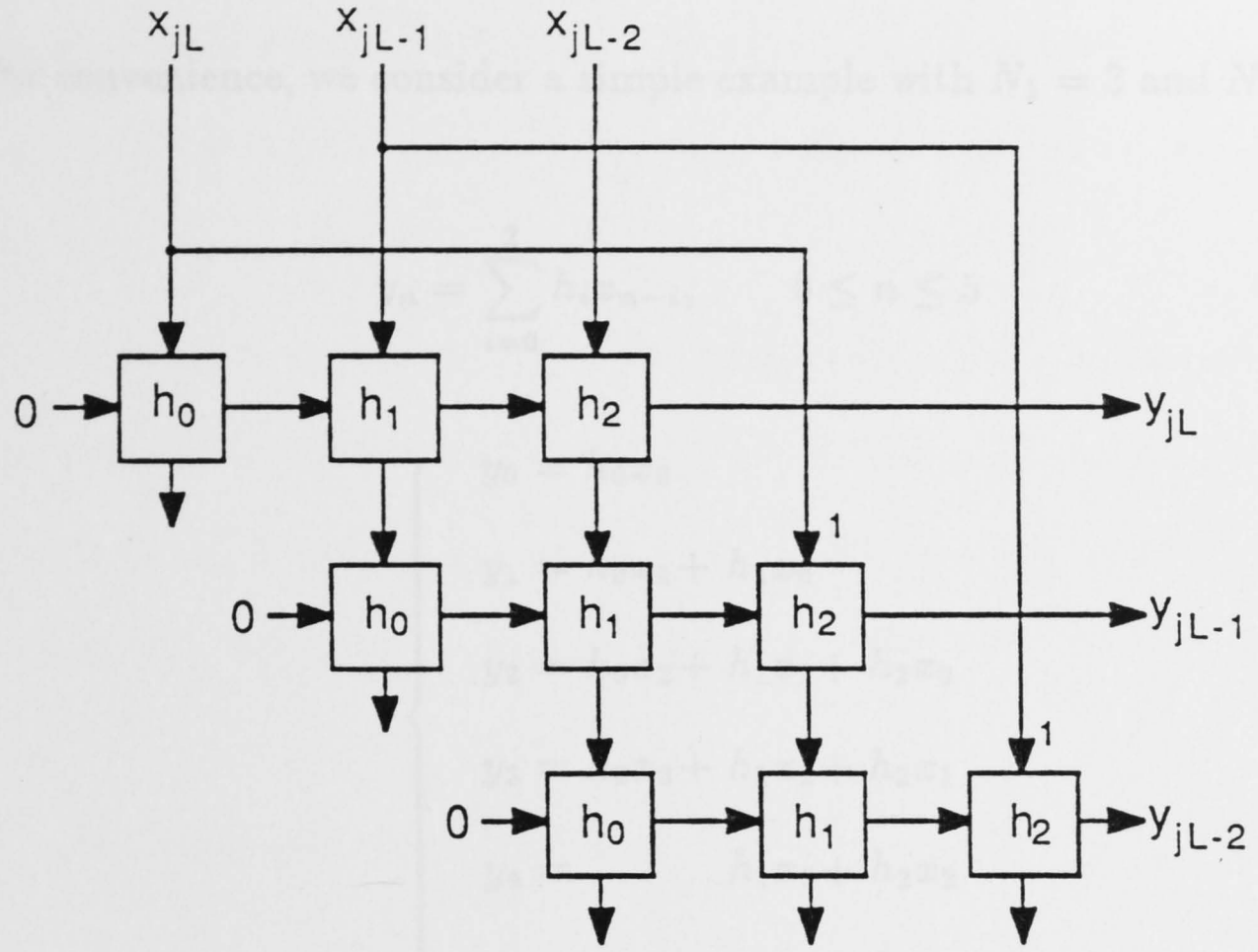


Fig. 3.8. The $L \times N_1$ systolic array for solving linear convolution problems

example, when $L = 3$ and $N_1 = 3$, the equation has the form

$$\begin{pmatrix} y_{jL} \\ y_{jL-1} \\ y_{jL-2} \end{pmatrix} = \begin{pmatrix} h_0 & h_1 & h_2 & 0 & 0 \\ 0 & h_0 & h_1 & h_2 & 0 \\ 0 & 0 & h_0 & h_1 & h_2 \end{pmatrix} \begin{pmatrix} x_{jL} \\ x_{jL-1} \\ x_{jL-2} \\ x_{jL-3} \\ x_{jL-4} \end{pmatrix}. \tag{3.12}$$

The above equation describes a matrix-vector multiplication. It can then be computed by using the matrix-vector multiplier depicted in Fig. 3.7. Because some input data has to be reused for computing different blocks of output (which is easily seen by taking two successive blocks of output into account), extra communication lines have to be introduced. From Fig. 3.8 we can see that the area taken by the communication lines is very large. Therefore, this implementation is not area efficient in VLSI.

3.3.2. $N_1 \times N_2$ 2-D systolic array

For convenience, we consider a simple example with $N_1 = 3$ and $N_2 = 4$, that is

$$y_n = \sum_{i=0}^{2} h_i x_{n-i}, \qquad 0 \le n \le 5 \tag{3.13.a}$$

or

$$\begin{cases} y_0 = h_0 x_0 \\ y_1 = h_0 x_1 + h_1 x_0 \\ y_2 = h_0 x_2 + h_1 x_1 + h_2 x_0 \\ y_3 = h_0 x_3 + h_1 x_2 + h_2 x_1 \\ y_4 = \qquad h_1 x_3 + h_2 x_2 \\ y_5 = \qquad\qquad h_2 x_3 \end{cases} \tag{3.13.b}$$

By inspection of the equations (3.13) a 2-D systolic structure can easily be obtained, as shown in Fig. 3.9.

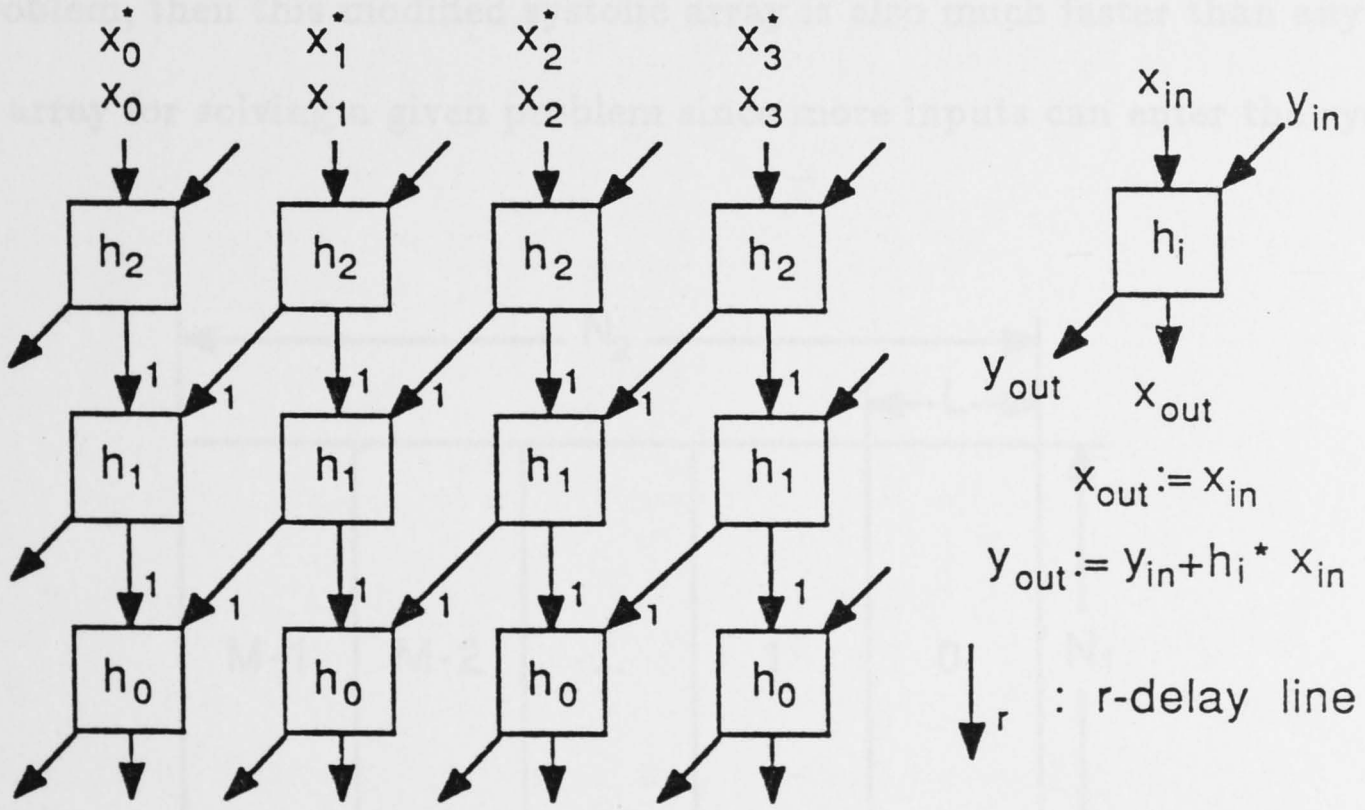


Fig. 3.9. An $N_1 \times N_2$ systolic array for solving linear convolution problems

In this systolic array, the coefficients are pre-stored in the cells. Cells in the same row contain the same coefficient. While the input data travel in parallel from top to bottom, the partial results move diagonally downward to accumulate the terms for each sum in (3.13).

From Fig. 3.9 it can be seen that the computation for a subsequent problem can start immediately after the inputs of the previous problem have entered the array and that the area is proportional to the product of N_1 and N_2. Therefore, $AP^2 = O(N_1 N_2)$, which is asymptotically optimal [8].

3.3.3. $N_1 \times L$ systolic ring structure

The 2-D systolic array derived above is impractical because it has too many input/output lines and requires too large an area for a large problem. If we use only one part of the array and do some modifications so that it can still solve the same problem, then this modified systolic array is also much faster than any 1-D systolic array for solving a given problem since more inputs can enter the system.

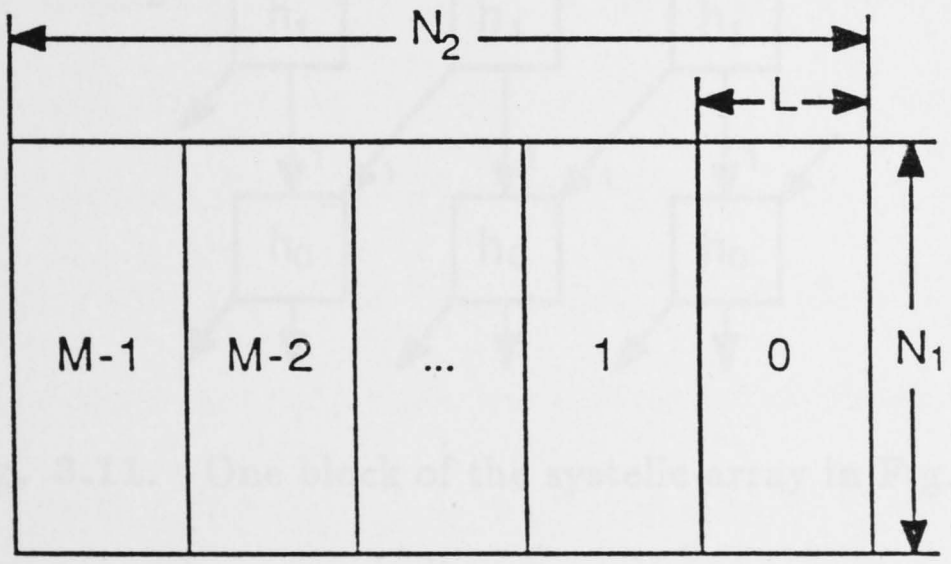


Fig. 3.10. A block-diagram of the $N_1 \times N_2$ systolic array

Fig. 3.10 gives a block-diagram of the $N_1 \times N_2$ systolic array. In this figure the array is divided into M blocks with L columns of cells per block, that is $N_2 = ML$. As mentioned above, only one block of the array is to be used for solving the given problem. The input has also to be organized as an $M \times L$ data matrix. Data enters this one-block array row by row. An example is given in Fig. 3.11, in which $N_2 = 9$, $N_1 = 5$ and $L = 3$, so $M = N_2/L = 3$.

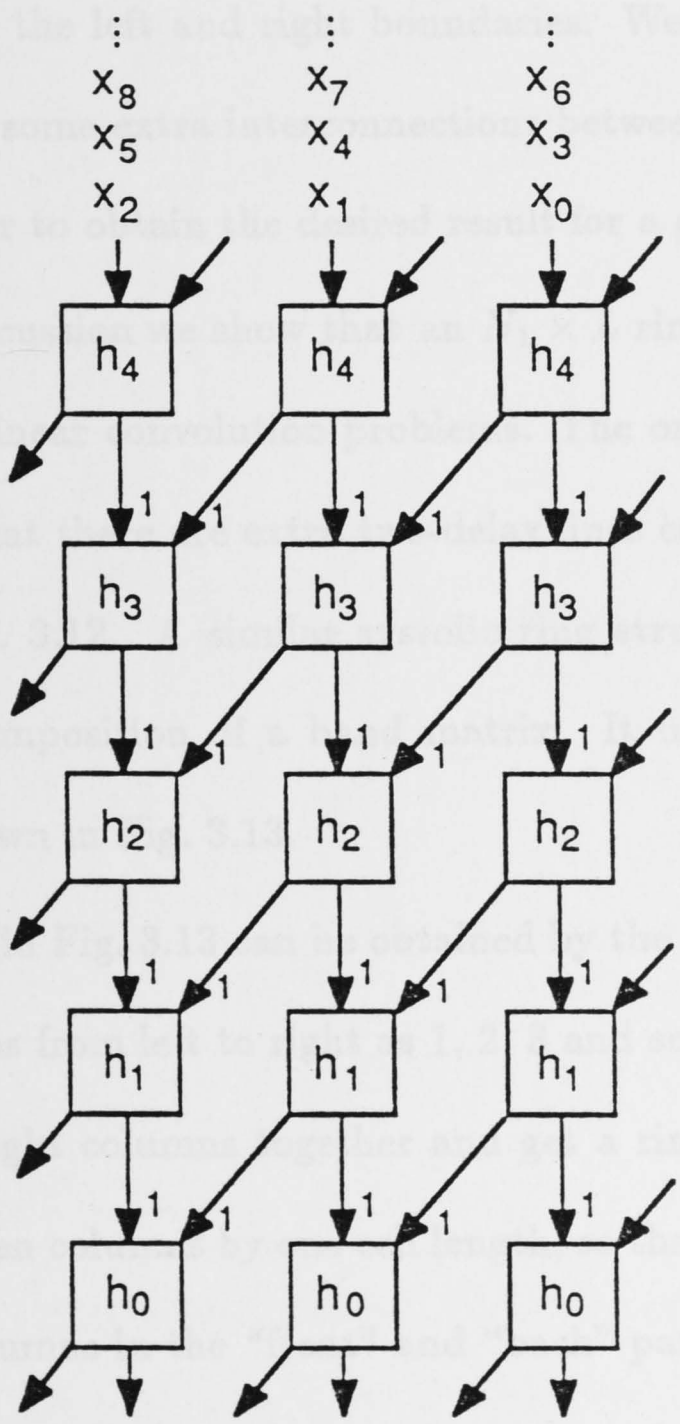


Fig. 3.11. One block of the systolic array in Fig. 3.10

If N_2 is not a multiple of L, we can easily add some zeros to the left or right sides of the input. (Correspondingly, we can put some extra columns of cells on

the left or right boundaries of Fig. 3.10.) This does not affect the correctness of the computation.

From Subsection 3.3.2 it is known that the period is one for this systolic array. The corresponding partial results produced by different rows of the input in Fig. 3.11 cannot be accumulated to form the final output. By observing the block-diagram in Fig. 3.10, it can be seen that each block communicates only with adjacent blocks on the left and right boundaries. We can visualize that in Fig. 3.11 there must be some extra interconnections between the block's left and right boundaries in order to obtain the desired result for a given problem.

In the following discussion we show that an $N_1 \times L$ ring structure, as shown in Fig. 3.12, can solve linear convolution problems. The only difference between Figs. 3.11 and 3.12 is that there are extra two-delay lines connecting the left and right boundaries in Fig. 3.12. A similar systolic ring structure was introduced in [19] for the LU-decomposition of a band matrix. It uses nearest neighbour interconnections, as shown in Fig. 3.13.

The systolic layout in Fig. 3.13 can be obtained by the following method. We first number the columns from left to right as 1, 2, 3 and so on. For the first row, we bring the left and right columns together and get a ring structure. We then expand the space between columns by one cell length, so that if we flatten out this ring the consecutive columns in the "front" and "back" parts will be interleaved. For the remaining rows, we apply odd-even column interchanges.

Before proving that the $N_1 \times L$ systolic ring structure can solve linear convolution problems, we set up Cartesian coodinates to locate the input x_k in the data matrix and the coefficients h_i in the array. The coordinate axes are given in

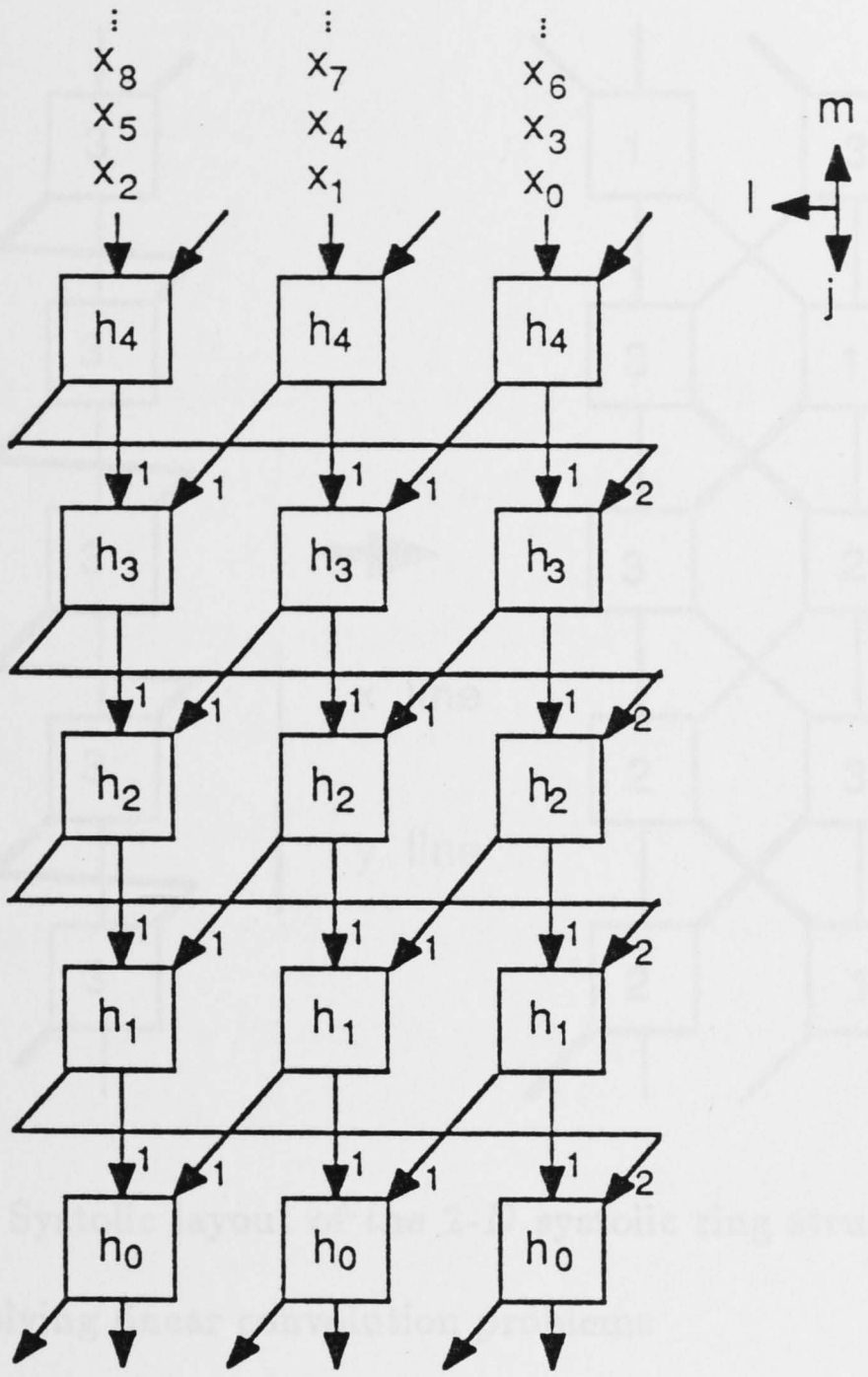


Fig. 3.12. An $N_1 \times L$ ring structure for solving linear convolution problems

Fig. 3.12.

The input x_k can be represented by x_{lm}. The relation between k and the two indices, l and m, is

$$k = l + mL \tag{3.14}$$

where $0 \leq k \leq N_2 - 1$, $0 \leq l \leq L - 1$ and $0 \leq m \leq M - 1$.

A cell in the systolic array is defined by its coordinates (j, l). For example, the

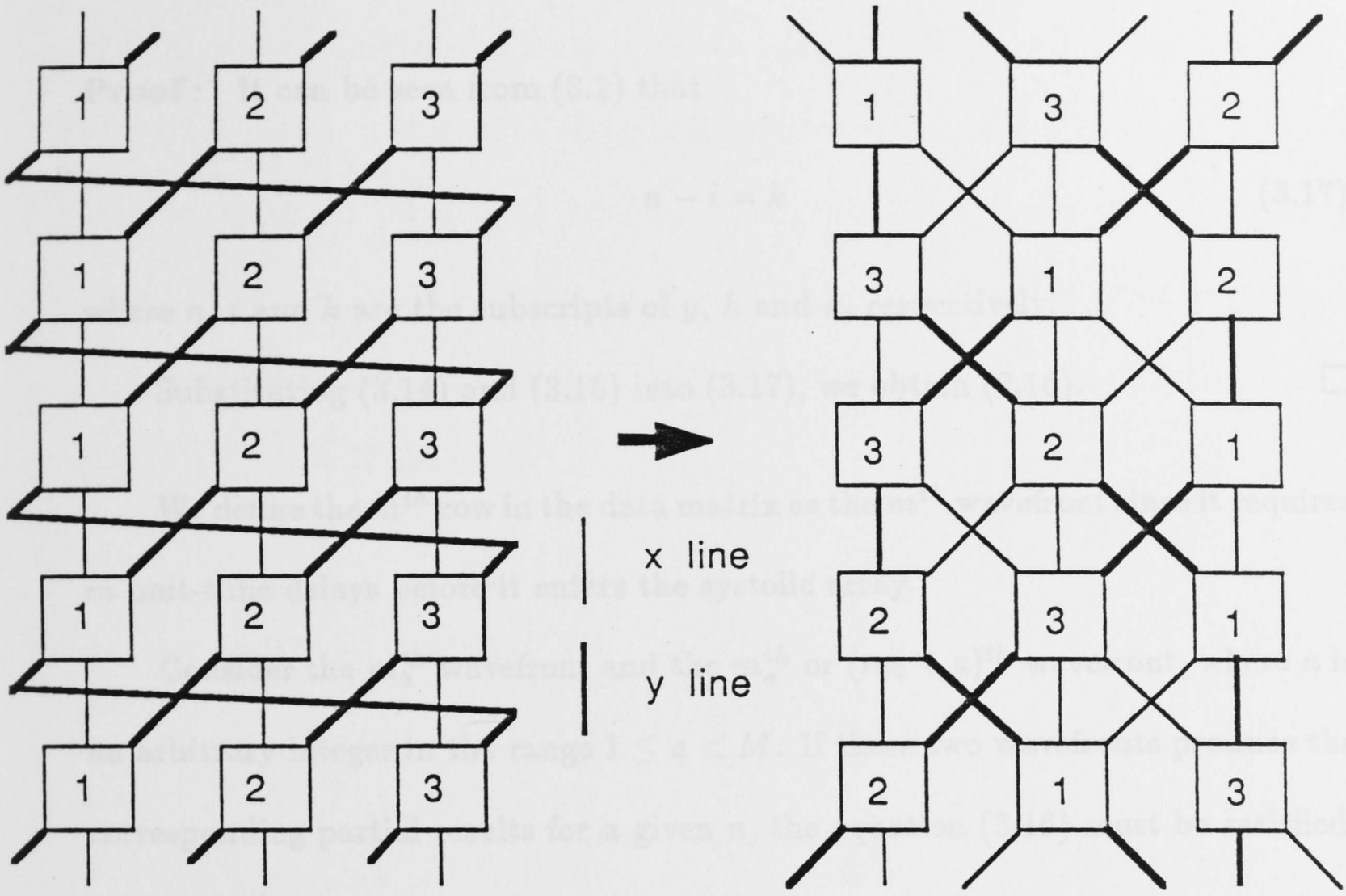


Fig. 3.13. Systolic layout of the 2-D systolic ring structure for solving linear convolution problems

top-right cell in Fig. 3.12 is cell $(0,0)$ and the bottom-left cell is cell $(N_1-1, L-1)$. Since cells on the same row store the same coefficient, it is easy to locate the coefficients in the array by using the j axis as

$$i = N_1 - 1 - j \tag{3.15}$$

where i is the subscript of the coefficients.

Lemma 3.1 : The relation between the output y_n and the coordinates j, l and m must satisfy

$$n = N_1 - 1 + mL + l - j \tag{3.16}$$

where n is the subscript of the output y_n.

Proof: It can be seen from (3.1) that

$$n - i = k \tag{3.17}$$

where n, i and k are the subscripts of y, h and x, respectively.

Substituting (3.14) and (3.15) into (3.17), we obtain (3.16). □

We define the m^{th} row in the data matrix as the m^{th} wavefront since it requires m unit-time delays before it enters the systolic array.

Consider the m_0^{th} wavefront and the m_a^{th} or $(m_0 + a)^{th}$ wavefront, where a is an arbitrary integer in the range $1 \leq a < M$. If these two wavefronts produce the corresponding partial results for a given n, the equation (3.16) must be satisfied. For the m_0^{th} wavefront we then have

$$n = N_1 - 1 + m_0 L + l_0 - j_0 \tag{3.18}$$

and for the $(m_0 + a)^{th}$ wavefront,

$$n = N_1 - 1 + (m_0 + a)L + l_a - j_a. \tag{3.19}$$

Combining (3.18) and (3.19) yields

$$l_0 - j_0 = aL + l_a - j_a. \tag{3.20}$$

Since the difference between Figs. 3.11 and 3.12 is the interconnections between two boundaries, we only consider $l_0 = L - 1$ (left boundary) and $l_a = 0$ (right boundary). The equation (3.20) becomes

$$j_0 + (a - 1)L + 1 = j_a. \tag{3.21}$$

The above equation shows that for a given n the partial result from the m_0^{th} wavefront accumulates the final term at cell $(j_0, L-1)$, while the $(m_0+a)^{th}$ wavefront produces its first term for that n at cell $(j_0+(a-1)L+1, 0)$. It can be seen from Fig. 3.14 that the partial result at cell $(j_0, L-1)$ can only arrive at cell $(j_0+(a-1)L+1, 0)$. Therefore, we need to show that the partial result produced by the m_0^{th} wavefront arrives at cell $(j_0+(a-1)L+1, 0)$ at the same time as the $(m_0+a)^{th}$ wavefront arrives.

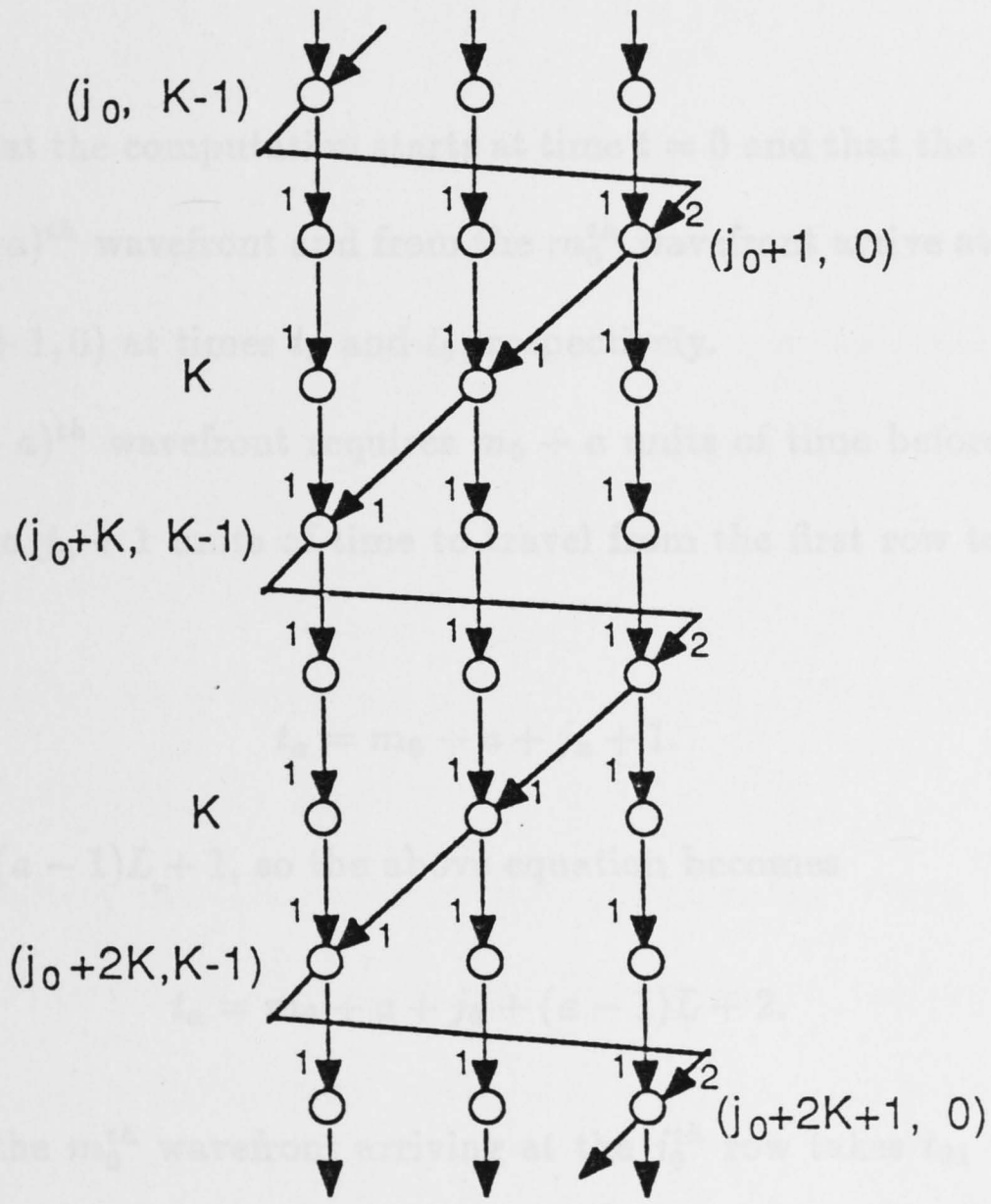


Fig. 3.14. The route for one output in the ring structure

Lemma 3.2 : For a partial result to move downward L rows takes $L+1$ units of time.

Proof : In Fig. 3.14, a partial result moving from cell $(j_0, L-1)$ to cell $(j_0 + L, L-1)$ first goes through a two-delay line to cell $(j_0+1, 0)$ on the right boundary and then moves diagonally downward L rows to cell $(j_0 + L, L-1)$, which takes $L-1$ units of time. Therefore the total time required is

$$t = 2 + L - 1 = L + 1.$$

It is easy to see that this is true not only for the first column on the left boundary, but also for other columns. □

Suppose that the computation starts at time $t = 0$ and that the partial results from the $(m_0 + a)^{th}$ wavefront and from the m_0^{th} wavefront arrive at cell $(j_a, 0)$ or $(j_0 + (a-1)L + 1, 0)$ at times t_a and t_0, respectively.

The $(m_0 + a)^{th}$ wavefront requires $m_0 + a$ units of time before entering the system and then $j_a + 1$ units of time to travel from the first row to the j_a^{th} row. The time t_a is

$$t_a = m_0 + a + j_a + 1. \tag{3.22}$$

Now $j_a = j_0 + (a-1)L + 1$, so the above equation becomes

$$t_a = m_0 + a + j_0 + (a-1)L + 2. \tag{3.23}$$

Similarly, the m_0^{th} wavefront arriving at the j_0^{th} row takes t_{01} units of time, that is,

$$t_{01} = m_0 + j_0 + 1. \tag{3.24}$$

The partial result produced by the m_0^{th} wavefront then moves zigzag downward $(a-1)L$ rows to cell $(j_0+(a-1)L, L-1)$, which requires $(a-1)L+a-1$ units

of time according to Lemma 3.2. Finally this partial result has to pass through a two-delay line before arriving at cell $(j_0 + (a-1)L + 1, 0)$ on the right boundary. Fig. 3.14 gives an example with $a = 3$. The time for the partial result to move from cell $(j_0, L-1)$ to cell $(j_0 + (a-1)L + 1, 0)$ is

$$t_{02} = (a-1)L + a - 1 + 2 = (a-1)L + a + 1. \tag{3.25}$$

Therefore, combining (3.24) and (3.25), we obtain

$$t_0 = t_{01} + t_{02} = m_0 + a + j_0 + (a-1)L + 2 = t_a. \tag{3.26}$$

Since we assumed that a is arbitrary in the above discussion, all the partial results for the given output can properly be accumulated in the system. Therefore, we conclude that the $N_1 \times L$ systolic ring structure can indeed solve linear convolution problems.

3.3.4. An upper bound on AP^2

In this subsection we derive an upper bound on AP^2 for the systolic ring structure described in Section 3.3.3.

We first consider the area taken by the systolic array. From Fig. 3.13, it is easy to see that the area is dominated by cells. Therefore, the area taken by an $N_1 \times L$ systolic ring is proportional to the product of N_1 and L, that is,

$$A = O(LN_1). \tag{3.27}$$

Since linear convolution problems can be computed by using the $N_1 \times L$ systolic ring, it is obvious that two product terms of $h_i x_{in}$ produced by x_{i_a} and x_{i_b} are accumulated by one output y_n in the $N_1 \times L$ systolic ring if and only if

$|i_a - i_b| \leq N_1 - 1$. Therefore, in order to avoid two successive linear convolution problems interfering with each other, the first problem must be followed by $N_1 - 1$ zeros. However, since L inputs can enter the systolic array simultaneously at one time, the period is

$$P = \frac{N_1 + N_2 - 1}{L}. \tag{3.28}$$

Combining (3.27) and (3.28), we obtain the upper bound

$$AP^2 = O(\frac{N_1(N_1 + N_2)^2}{L}).$$

From (3.27) and (3.28), it can also be seen that AP is independent of L. Therefore, by varying L we can trade off area versus period for a given linear convolution problem.

3.3.5. The second systolic ring structure

Using cut-set localization rules plus the commutative rule described in Chapter 2, we can transform the systolic array in Fig. 3.12 into the array in Fig. 3.15.

First we apply a set of horizontal cuts and subtract one delay from each line in these cuts; the delays on the input (x_i) lines become zero. Since x_i remains unchanged during the computation, we can change the direction of the input lines.

We then apply a set of diagonal cuts and delay transfer for the lines in the cuts so that the delays on all diagonal lines become zero. We can change the direction of the diagonal lines without affecting the final result since this is a non-feedback system so that the commutative law for addition can be applied.

Finally we apply another set of horizontal cuts and add one delay to each line on these cuts. The array in Fig. 3.12 is then transformed into the array in Fig. 3.15.

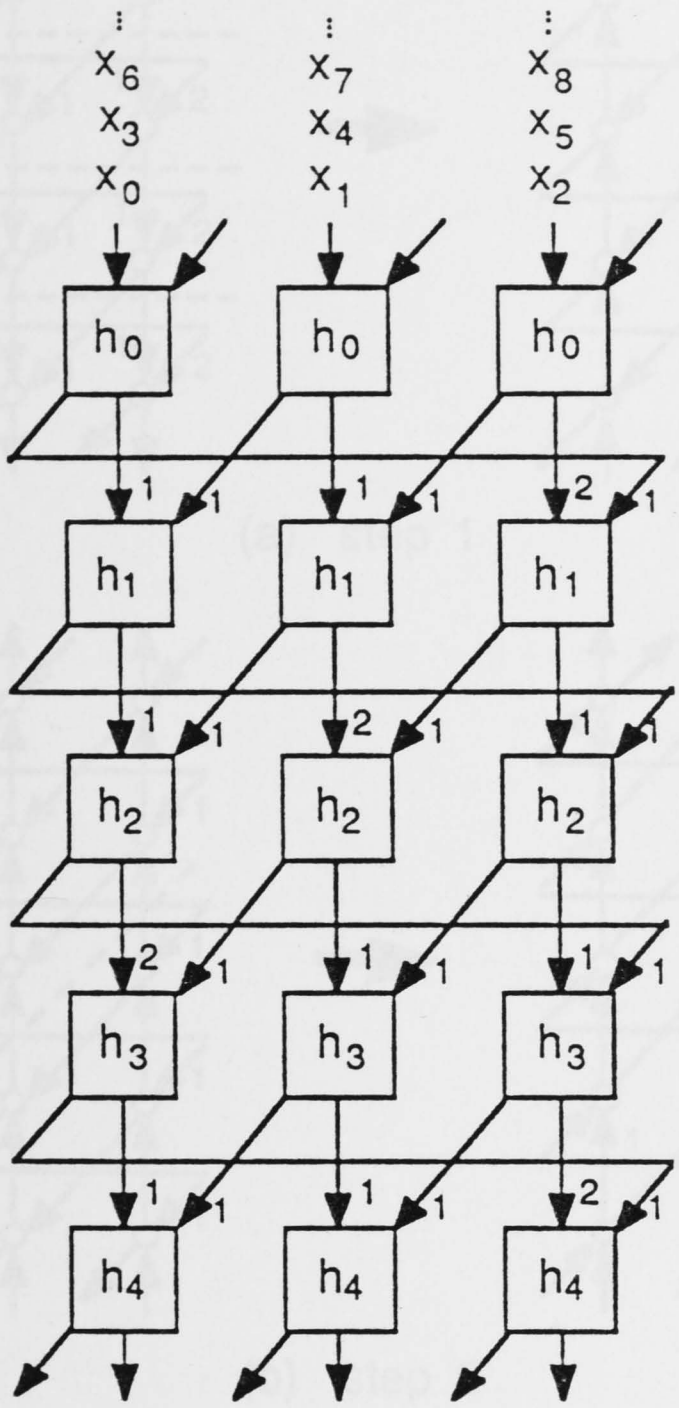
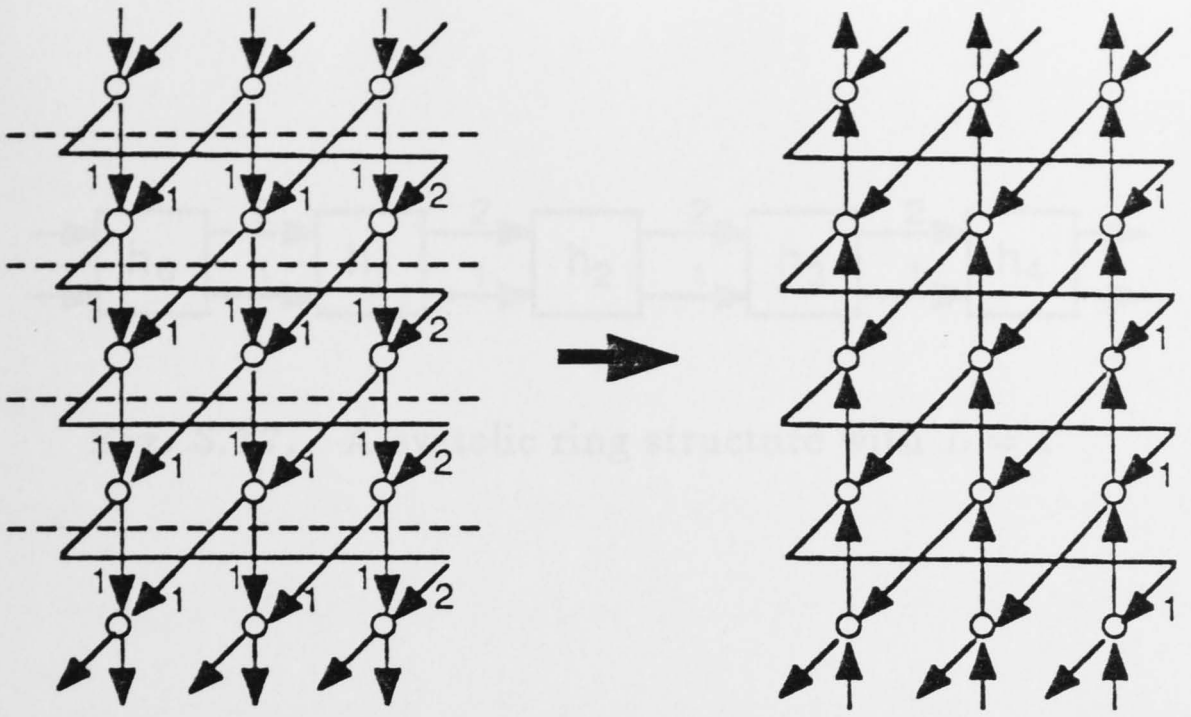


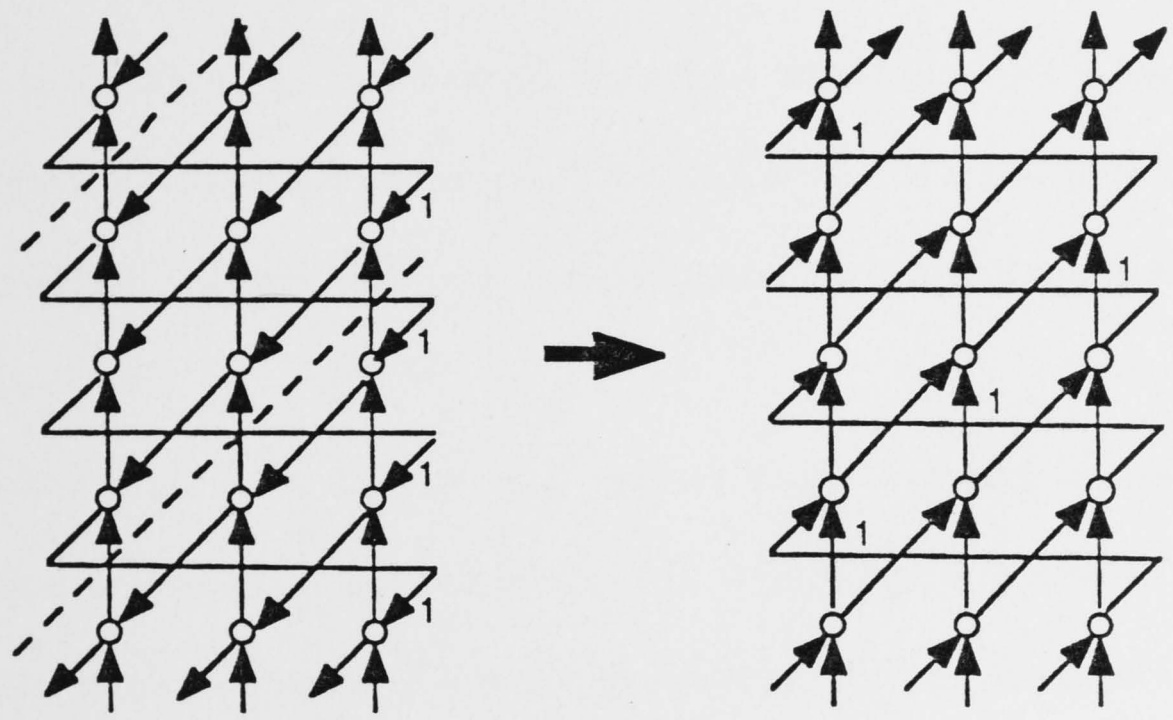
Fig. 3.15. A systolic array equivalent to that of Fig. 3.12

The above three transform steps are given in Fig. 3.16.

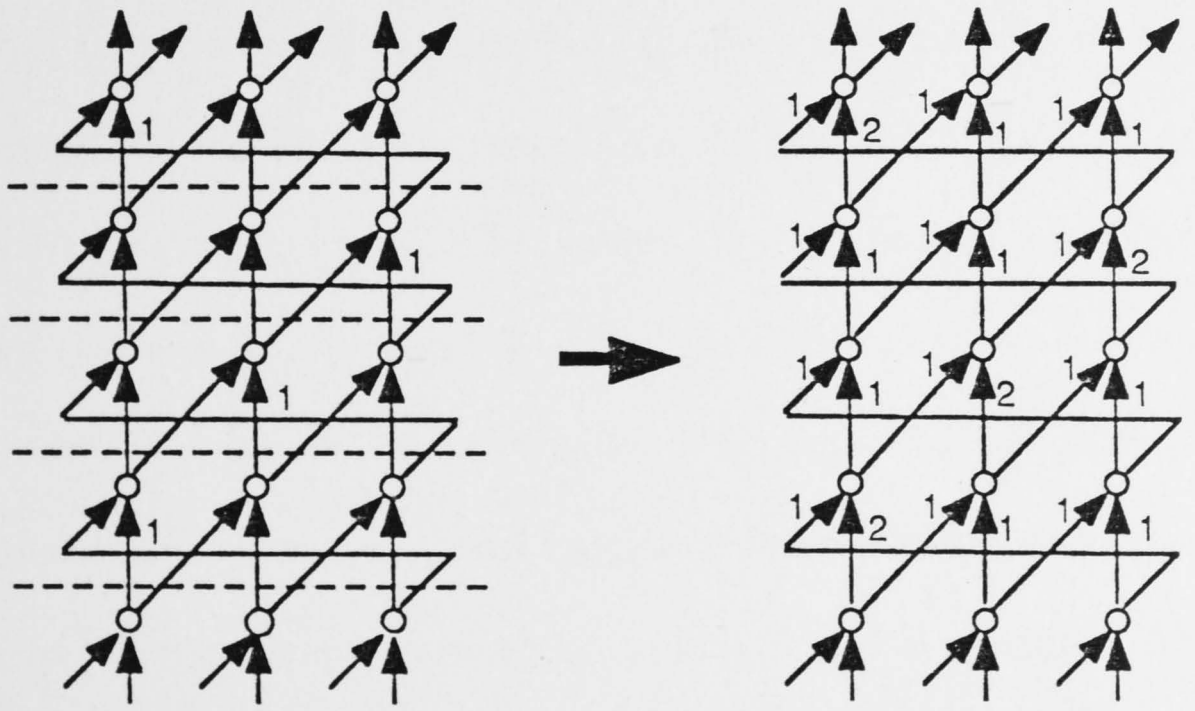
For $L = 1$, the 2-D systolic ring in Fig. 3.15 becomes a 1-D systolic array, as shown in Fig. 3.17. This 1-D systolic array is the most efficient 1-D systolic array for solving linear convolution problems among those reported in the literature [29].



(a) step 1



(b) step 2



(c) step 3

Fig. 3.16. Steps for transforming Fig. 3.12 into Fig. 3.15

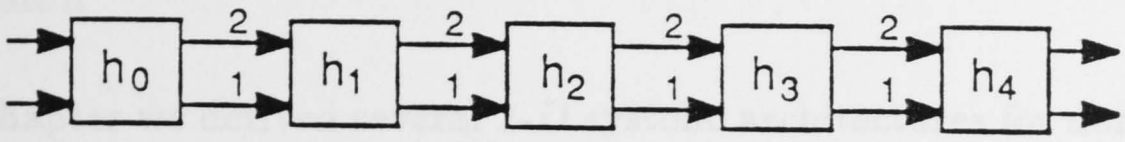


Fig. 3.17. A systolic ring structure with $L = 1$

3.4. Discussion

In this chapter we derived several 2-D systolic architectures for non-recursive digital filters.

We derived a 2-D systolic array for 1-D linear phase non-recursive filters. In this design we first divided the problem into several sub-problems. All these sub-problems are implemented in 1-D systolic arrays. Then we applied cut-set localization rules to put those 1-D arrays together to form a regular 2-D system. By using N_1 multipliers, this 2-D systolic array can achieve twice the throughput of the 1-D systolic arrays which use $N_1/2$ multipliers for a given problem. The disadvantage of this design is that long communication lines are required to achieve a higher throughput.

We also introduced two 2-D $N_1 \times L$ systolic ring structures for solving linear convolution problems. The method used in this design is different from the one for parallel implementation of linear phase filters. We first constructed a 2-D $N_1 \times N_2$ system for linear convolution problems with N_1 coefficients and N_2 inputs, and then partitioned this system into an $N_1 \times L$ ring structure. A systolic array for solving the same problem was obtained after applying odd-even column interchanges to the ring structure. The derived $N_1 \times L$ systolic ring structures are more efficient than the $L \times N_1$ array described in [30] because to achieve the same throughput our arrays require less area in VLSI implementation. It is interesting to note that the property of the nearest neighbour interconnection in these systolic ring structures is always guaranteed when the length of L is changed. With $L = 1$ a 2-D ring structure becomes a 1-D array which has been proved to be the most efficient 1-D systolic array for solving linear convolution problems. Because AP is

independent of L, we can trade off area versus time for a given problem by varying L.

When we were constructing the 2-D systolic arrays for non-recursive filters, we also demonstrated the power and generality of cut-set localization rules. There are already several well known applications of these rules, such as to localize a computing network with global communications, to transform one structure into other different ones with the same functions and to apply second-level pipelining to a given system so that the throughput of that system is greatly increased without a great increase in complexity. In this chapter we showed that cut-set localization rules are also useful in combining several sub-systems to form an efficient system with high throughput.

With some minor modifications, the systolic ring structures can also solve DFT, circular convolution and 2-D linear convolution problems because these problems can easily be transformed into 1-D linear convolution problems [11,22,43]. We shall see in Chapter 5 that the 2-D ring structures can be applied as a linear part in an efficient parallel structure for direct-form recursive filters.

CHAPTER 4

STABILIZED PARALLEL ALGORITHMS FOR DIRECT-FORM RECURSIVE FILTERS

4.1. Introduction

Recursive filtering is one of the most important techniques in digital signal processing. A lot of effort has been made in the past few years to achieve high-throughput implementation of this type of filtering [2,30,34,36,39-41,43,51,53-55]. However, these methods were mainly based on the conventional look-ahead computation. Although this technique has been applied successfully to the parallel implementation of state-variable-form recursive filters, it may cause numerical instability when applied to direct-form recursive filters, due to the effect of finite wordlength. Thus this chapter introduces new methods for deriving stabilized parallel algorithms for direct-form recursive filters. These algorithms lead to very efficient pipelined and/or parallel structures, which will be described in Chapter 5. The derived structures belong to the category of systolic/wavefront arrays and are suitable for VLSI.

In Section 4.2, we describe the conventional look-ahead computation technique and show that it may cause numerical instability in the direct-form implementation of recursive filters. Section 4.3 introduces a new method for deriving, in the Z domain, a stabilized parallel algorithm for direct-form recursive filters. The degree of parallelism and the stability of this algorithm are also analyzed. In order to obtain this stabilized algorithm, extra zeros and poles have been introduced. The complexity is then greatly increased. In Section 4.4 we introduce a technique

called decomposition to minimize this increase in complexity. Since the algorithm is derived in the Z domain, it cannot be used for time-varying recursive systems. In Section 4.5, we consider a time domain derivation of the parallel algorithms for direct-form recursive filters.

4.2. Conventional Look-Ahead Computation

In the look-ahead technique, the given recursion is iterated as many times as desired to create the necessary concurrency and then the concurrency created can be used to obtain pipelined and/or parallel implementation of recursive systems [41]. In the following we give an example to show that this conventional technique may cause numerical instability in the practical implementation of direct-form recursive filters due to the effect of finite wordlength.

An N^{th} order direct-form recursive filter can be expressed as

$$y_n = \sum_{j=1}^{N} r_j y_{n-j} + \sum_{j=0}^{N} w_j x_{n-j} \tag{4.1}$$

Because y_n in (4.1) depends on the availability of the immediately previous output y_{n-1}, it is not obvious that any two outputs can be computed in parallel. Conventional look-ahead computation is applied to increase the degree of parallelism as follows.

Using (4.1), we write y_{n-1} explicitly as

$$y_{n-1} = \sum_{j=1}^{N} r_j y_{n-1-j} + \sum_{j=0}^{N} w_j x_{n-1-j} \tag{4.2}$$

Let $j' = j + 1$. Then

$$y_{n-1} = \sum_{j'=2}^{N+1} r_{j'-1} y_{n-j'} + \sum_{j'=1}^{N+1} w_{j'-1} x_{n-j'} \tag{4.3}$$

Substituting (4.3) into (4.1), we obtain

$$\begin{aligned} y_n &= r_1 y_{n-1} + \sum_{j=2}^{N} r_j y_{n-j} + \sum_{j=0}^{N} w_j x_{n-j} \\ &= r_1 \Big(\sum_{j=2}^{N+1} r_{j-1} y_{n-j} + \sum_{j=1}^{N+1} w_{j-1} x_{n-j} \Big) + \sum_{j=2}^{N} r_j y_{n-j} + \sum_{j=0}^{N} w_j x_{n-j} \qquad (4.4) \\ &= \sum_{j=2}^{N+1} r_j^{(1)} y_{n-j} + \sum_{j=0}^{N+1} w_j^{(1)} x_{n-j} \end{aligned}$$

where

$$w_j^{(1)} = \begin{cases} w_0 & j = 0 \\ w_j + r_1 w_{j-1} & 1 \le j \le N \\ r_1 w_N & j = N+1 \end{cases}$$

and

$$r_j^{(1)} = \begin{cases} r_j + r_1 r_{j-1} & 2 \le j \le N \\ r_1 r_N & j = N+1 \end{cases}$$

Since y_n is independent of y_{n-1} after the modification, two outputs can be computed simultaneously. We can prove that, after $L-1$ iterations, y_n will become

$$y_n = \sum_{j=L}^{N+L-1} r_j^{(L-1)} y_{n-j} + \sum_{j=0}^{N+L-1} w_j^{(L-1)} x_{n-j} \tag{4.5}$$

In the above equation, the coefficients can be computed by the following iterative algorithm.

for $i \le l \le L$ **do**

begin

for $0 \le j \le N+l-1$ **do**

begin { to compute $w_j^{(l-1)}$ }

if $0 \le j < l-1$ **then**

$w_j^{(l-1)} := w_j^{(l-2)}$

else if $l-1 \le j < N+l-1$ **then**

$w_j^{(l-1)} := w_j^{(l-2)} + r_{l-1}^{(l-2)} * w_{j-(l-1)}$

else $w_j^{(l-1)} := r_{l-1}^{(l-2)} * w_N$ (4.6)

end; { end of computing $w_j^{(l-1)}$ }

for $0 \le j \le N+l-1$ **do**

begin { to compute $r_j^{(l-1)}$ }

if $0 \le j < l$ **then**

$r_j^{(l-1)} := 0$

else if $l \leq j < N + l - 1$ **then**

$r_j^{(l-1)} := r_j^{(l-2)} + r_{l-1}^{(l-2)} * r_{j-(l-1)}$

else $r_j^{(l-1)} := r_{l-1}^{(l-2)} * r_N$

end { end of computing $r_j^{(l-1)}$ }

end.

(The proofs of (4.5) and (4.6) are given in Appendix 4.1.)

To analyze the stability of the modified algorithm, we transform it into the Z domain and then obtain the impulse response function as

$$H(z) = \frac{\sum_{j=0}^{N+L-1} w_j^{(L-1)} z^{-j}}{1 - \sum_{L}^{N+L-1} r_j^{(L-1)} z^{-j}} \tag{4.7}$$

We prove, in Appendix 4.2, that $H(z)$ in (4.7) can be rewritten as

$$H(z) = \tilde{H}(z) \frac{D(z)}{D(z)} \tag{4.8}$$

In the above equation, $\tilde{H}(z)$ is the impulse response function before the modification and $D(z)$ is an $(L-1)^{th}$ order polynomial in z^{-1}, which is defined as

$$D(z) = 1 + \sum_{j=1}^{L-1} r_j^{(j-1)} z^{-j} \tag{4.9}$$

where $r_j^{(j-1)}$ can be computed by the iterative algorithm in (4.6).

From (4.8) we see that, to obtain a parallel algorithm, we have multiplied both numerator and denominator of the original impulse response function by a factor $D(z)$. Because of the zero and pole cancellation, the impulse response function after the modification is theoretically equivalent to the original one. However, we cannot guarantee that all roots of $D(z)$ are within the unit circle. Thus the modification may cause numerical instability due to the effect of finite wordlength. We give an example below.

Consider a second-order recursive filter. The denominator of $\tilde{H}(z)$ is given as $1 - 1.5z^{-1} + 0.56z^{-2}$. Since the two roots of this denominator are 0.7 and 0.8, the system is stable. We now derive a parallel algorithm by using the above conventional technique. After one iteration, the root of $D(z) = 1 + r_1 z^{-1}$ is -1.5, which is outside the unit circle. After two iterations, the roots of $D(z) = 1 + r_1 z^{-1} + r_2^{(1)} z^{-2}$ are $-0.75 \pm 1.5i$. They are also outside the unit circle. Therefore, the modified system is definitely unstable.

4.3. Stabilized Parallel Algorithm

In the previous section, we have shown that applying the conventional look-ahead computation to the implementation of direct-form recursive filters may cause numerical instability because the extra poles introduced cannot be guaranteed to be inside the unit circle. In this section we derive a new parallel algorithm which is guaranteed to be stable if the original (serial) algorithm is stable.

4.3.1. Algorithm

The impulse response of an N^{th} order recursive filter can be expressed as

$$\tilde{H}(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{j=0}^{N} w_j z^{-j}}{1 - \sum_{j=1}^{N} r_j z^{-j}} \tag{4.10}$$

where $X(z)$ and $Y(z)$ represent the Z transforms of input and output, respectively.

To obtain our parallel algorithm, we first introduce a well known $N \times N$ matrix, which is called a companion matrix,

$$\mathbf{B} = \begin{pmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ -b_0 & -b_1 & -b_2 & \dots & -b_{N-1} \end{pmatrix} \tag{4.11}$$

In this matrix, the elements on the first superdiagonal are all equal to one and the j^{th} element on the last row is $-b_{j-1}$, but all other elements are equal to zero. It is known that

$$\begin{aligned} \det(z\mathbf{I} - \mathbf{B}) &= z^N + b_{N-1} z^{N-1} + \cdots + b_1 z + b_0 \\ &= z^N + \sum_{j=1}^{N} b_{N-j} z^{N-j} \end{aligned} \tag{4.12}$$

where $\det(\mathbf{X})$ denotes the determinant of the matrix $\mathbf{X}$ and $\mathbf{I}$ is an $N \times N$ identity

matrix. Let $-b_j = r_{N-j}$. Then

$$\mathbf{B} = \begin{pmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ r_N & r_{N-1} & r_{N-2} & \dots & r_1 \end{pmatrix} \tag{4.13}$$

and

$$\det(z\mathbf{I} - \mathbf{B}) = z^N - \sum_{j=1}^{N} r_j z^{N-j} \tag{4.14}$$

Multiplying both sides of (4.14) by z^{-N}, we obtain

$$\det(\mathbf{I} - \mathbf{B}z^{-1}) = 1 - \sum_{j=1}^{N} r_j z^{-j} \tag{4.15}$$

From the above equation, we see that by using the companion matrix $\mathbf{B}$, the denominator of the impulse response function in (4.10) can be expressed in matrix form. Thus we can rewrite $\tilde{H}(z)$ as

$$\tilde{H}(z) = \frac{\sum_{j=0}^{N} w_j z^{-j}}{\det(\mathbf{I} - \mathbf{B}z^{-1})} \tag{4.16}$$

We next multiply both numerator and denominator of (4.16) by a factor $\det(\sum_{j=0}^{L-1} \mathbf{B}^j z^{-j})$, where $\mathbf{B}^0 = \mathbf{I}$ and $\mathbf{B}^j$ is a product of j matrices $\mathbf{B}$. (An efficient method for computing $\mathbf{B}^j$ is given in Appendix 4.3.) The impulse response function then becomes

$$\begin{aligned} H(z) &= \tilde{H}(z) \frac{\det(\sum_{j=0}^{L-1} \mathbf{B}^j z^{-j})}{\det(\sum_{j=0}^{L-1} \mathbf{B}^j z^{-j})} \\ &= \frac{(\sum_{j=0}^{N} w_j z^{-j}) \det(\sum_{j=0}^{L-1} \mathbf{B}^j z^{-j})}{\det(\mathbf{I} - \mathbf{B}z^{-1}) \det(\sum_{j=0}^{L-1} \mathbf{B}^j z^{-j})} \\ &= \frac{(\sum_{j=0}^{N} w_j z^{-j}) \det(\sum_{j=0}^{L-1} \mathbf{B}^j z^{-j})}{\det((\mathbf{I} - \mathbf{B}z^{-1})(\sum_{j=0}^{L-1} \mathbf{B}^j z^{-j}))} \end{aligned} \tag{4.17}$$

Now

$$(\mathbf{I} - \mathbf{B}z^{-1})(\sum_{j=0}^{L-1} \mathbf{B}^j z^{-j}) = \mathbf{I} - \mathbf{B}^L z^{-L} \tag{4.18}$$

Substituting (4.18) into (4.17), we finally obtain our modified impulse response function as

$$H(z) = \frac{(\sum_{j=0}^{N} w_j z^{-j})\det(\sum_{j=0}^{L-1} \mathbf{B}^j z^{-j})}{\det(\mathbf{I} - \mathbf{B}^L z^{-L})} \tag{4.19}$$

which implies a parallel algorithm. The rest of this section will analyze the degree of parallelism and the stability of this algorithm.

4.3.2. Degree of parallelism

We may divide (4.19) into two parts:

$$H(z) = \frac{Y(z)}{U(z)}\frac{U(z)}{X(z)} = H_2(z)H_1(z) \tag{4.20}$$

In the above equation, $U(z)$ is an intermediate variable, and $H_2(z)$ and $H_1(z)$ are the recursive part and linear part, respectively, defined as follows

$$H_2(z) = \frac{Y(z)}{U(z)} = \frac{1}{\det(\mathbf{I} - \mathbf{B}^L z^{-L})} \tag{4.21}$$

and

$$H_1(z) = \frac{U(z)}{X(z)} = (\sum_{j=0}^{N} w_j z^{-j})\det(\sum_{j=0}^{L-1} \mathbf{B}^j z^{-j}) \tag{4.22}$$

It is recursion that limits the parallel implementation of recursive filters. To analyze the degree of parallelism, therefore, we only consider the recursive part $H_2(z)$.

Lemma 4.1. Suppose that $\mathbf{B}$ is an $N \times N$ matrix. Then $\det(\mathbf{I} - \mathbf{B}^L z^{-L})$ can be expressed as an NL^{th} order polynomial in z^{-1} with only $N+1$ terms, that is,

$$\det(\mathbf{I} - \mathbf{B}^L z^{-L}) = 1 - \sum_{j=1}^{N} b_{jL} z^{-jL} \tag{4.23}$$

where b_{jL} is a combination of some elements in B^L.

Proof: Let $\lambda = z^L$. Then

$$\det(\mathbf{I} - \mathbf{B}^L z^{-L}) = \det(\mathbf{I} - \mathbf{B}^L \lambda^{-1}) \tag{4.24}$$

Since $\mathbf{B}$ is an $N \times N$ matrix, so is $\mathbf{B}^L$. Thus $\det(\mathbf{I} - \mathbf{B}^L \lambda^{-1})$ is an N^{th} order polynomial in z^{-1} and can be expressed as

$$\det(\mathbf{I} - \mathbf{B}^L \lambda^{-1}) = 1 - \sum_{j=1}^{N} b_{jL} \lambda^{-j} \tag{4.25}$$

where b_{jL} is a combination of some elements in $\mathbf{B}^L$.

Substituting z^L for λ in (4.25), we then obtain (4.23). □

Using Lemma 4.1, we can rewrite $H_2(z)$ as

$$H_2(z) = \frac{Y(z)}{U(z)} = \frac{1}{1 - \sum_{j=1}^{N} b_{jL} z^{-jL}} \tag{4.26}$$

Converting (4.26) into the time domain, we obtain

$$y_n = \sum_{j=1}^{N} b_{jL} y_{n-jL} + u_n \tag{4.27}$$

Because y_n in (4.27) depends only on y_{n-jL} for $j = 1$ to N, L outputs can be computed simultaneously. That is why we call our modified algorithm a parallel algorithm.

In the following, we give an example of $N = 2$.

From Appendix 4.3, we can obtain that $\mathbf{B}^L = \begin{pmatrix} r_L^{(L-2)} & r_{L-1}^{(L-2)} \\ r_{L+1}^{(L-1)} & r_L^{(L-1)} \end{pmatrix}$, where $r_j^{(l)}$ is computed by using (4.6). Then

$$\mathbf{I} - \mathbf{B}^L z^{-L} = \begin{pmatrix} 1 - r_L^{(L-2)} z^{-L} & -r_{L-1}^{(L-2)} z^{-L} \\ -r_{L+1}^{(L-1)} z^{-L} & 1 - r_L^{(L-1)} z^{-L} \end{pmatrix} \tag{4.28}$$

We have

$$\begin{aligned}
\det(\mathbf{I}-\mathbf{B}^L z^{-L}) &= (1-r_L^{(L-2)}z^{-L})(1-r_L^{(L-1)}z^{-L}) - r_{L-1}^{(L-2)}r_{L+1}^{(L-1)}z^{-2L} \\
&= 1-(r_L^{(L-2)}+r_L^{(L-1)})z^{-L} - \\
&\quad -(r_{L-1}^{(L-2)}r_{L+1}^{(L-1)} - r_L^{(L-2)}r_L^{(L-1)})z^{-2L} \\
&= 1-\mathrm{tr}(\mathbf{B}^L)z^{-L} - (-\det(\mathbf{B}^L))z^{-2L} \\
&= 1-\sum_{j=1}^{2} b_{jL}z^{-jL}
\end{aligned} \tag{4.29}$$

where $\mathrm{tr}(\mathbf{B}^L)$ denotes the trace of $\mathbf{B}^L$, $b_{1L}=\mathrm{tr}(\mathbf{B}^L)$ and $b_{2L}=-\det(\mathbf{B}^L)$.

Since $H_2(z)=\frac{Y(z)}{U(z)}=\frac{1}{1-\sum_{j=1}^{2}b_{jL}z^{-jL}}$, then

$$y_n = b_{1L}y_{n-L} + b_{2L}y_{n-2L} + u_n \tag{4.30}$$

4.3.3. Stability

Since we have assumed that $\mathbf{B}$ is an $N\times N$ matrix,

$$\det(\mathbf{I}-\mathbf{B}^L z^{-L}) = z^{-NL}\det(\mathbf{I}z^L-\mathbf{B}^L) \tag{4.31}$$

and

$$\det(\sum_{j=0}^{L-1}\mathbf{B}^j z^{-j}) = z^{-N(L-1)}\det(\sum_{j=0}^{L-1}\mathbf{B}^j z^{L-1-j}) \tag{4.32}$$

We can rewrite $H(z)$ into another form as

$$H(z) = \frac{(\sum_{j=0}^{N}w_j z^{N-j})\det(\sum_{j=0}^{L-1}\mathbf{B}^j z^{L-1-j})}{\det(\mathbf{I}z^L-\mathbf{B}^L)} \tag{4.33}$$

Similarly, $\tilde{H}(z)$ in (4.16) can be rewritten as

$$\tilde{H}(z) = \frac{\sum_{j=0}^{N}w_j z^{N-j}}{\det(\mathbf{I}z-\mathbf{B})} \tag{4.34}$$

Suppose that the original algorithm before the modification is stable. Then the roots of $\det(\mathbf{I}z - \mathbf{B})$ are all in the unit circle. This means that the eigenvalues z_i of $\mathbf{B}$ are all in the unit circle. It is clear that the eigenvalues z_i^L of $\mathbf{B}^L$ are also in the unit circle and closer to the origin than their corresponding z_i's. Thus, the stability of our modified algorithm is obvious.

4.4. Reduction of Complexity

In this section we consider the number of multipliers required for implementing the modified algorithm in the 1-D case. (It will be seen, in the next chapter, that the number of multipliers in the 2-D case is only increased by a factor of L.) We shall see that the direct implementation of the modified algorithm is not very practical because the number of multipliers for that algorithm is very large. A technique called decomposition is introduced to reduce the number of multipliers.

Lemma 4.2. Suppose $\mathbf{B}$ is an $N \times N$ matrix, then $\det(\sum_{j=0}^{L-1} \mathbf{B}^j z^{-j})$ is an $N(L-1)^{th}$ polynomial in z^{-1}.

Proof: From Lemma 4.1, $\det(\mathbf{I} - \mathbf{B}^L z^{-L})$ is an NL^{th} order polynomial in z^{-1}. We also know that $\det(\mathbf{I} - \mathbf{B}z^{-1})$ is an N^{th} order polynomial in z^{-1}. However, we have $\det(\mathbf{I} - \mathbf{B}^L z^{-L}) = \det(\mathbf{I} - \mathbf{B}z^{-1})\det(\sum_{j=0}^{L-1} \mathbf{B}^j z^{-j})$. Thus the order of $\det(\sum_{j=0}^{L-1} \mathbf{B}^j z^{-j})$ must be $NL - N = N(L-1)$. □

Since $\det(\sum_{j=0}^{L-1} B^j z^{-j})$ is an $N(L-1)^{th}$ order polynomial, $N(L-1)$ multipliers are required for computing the associated convolution. Therefore, $N(L-1)$ extra multipliers have been introduced in the modified algorithm, which dominates the total number of multipliers. In the following, we apply the decomposition technique to reduce this number of multipliers.

Lemma 4.3. If $L = l_1 l_2$, where l_1 and l_2 are positive integers, then $\sum_{j=0}^{L-1} \mathbf{B}^j z^{-j}$ can be expressed as

$$\sum_{j=0}^{L-1} \mathbf{B}^j z^{-j} = (\sum_{j=0}^{l_2-1} (\mathbf{B}z^{-1})^{jl_1})(\sum_{j=0}^{l_1-1} (\mathbf{B}z^{-1})^j) \tag{4.35}$$

where $\mathbf{B}^0 = \mathbf{I}$.

Proof : We arrange L terms of $\sum_{j=0}^{L-1} \mathbf{B}^j z^{-j}$ into l_2 groups with l_1 terms each as follows

$$\begin{aligned}\sum_{j=0}^{L-1} \mathbf{B}^j z^{-j} = &[\mathbf{I} + \mathbf{B}z^{-1} + \cdots + (\mathbf{B}z^{-1})^{l_1-1}] + \\ &+ [(\mathbf{B}z^{-1})^{l_1} + (\mathbf{B}z^{-1})^{l_1+1} + \cdots + (\mathbf{B}z^{-1})^{2l_1-1}] + \cdots \\ &+ [(\mathbf{B}z^{-1})^{(l_2-1)l_1} + (\mathbf{B}z^{-1})^{(l_2-1)l_1+1} + \cdots + (\mathbf{B}z^{-1})^{l_2 l_1-1}]\end{aligned} \tag{4.36}$$

Let

$$\mathbf{Q} = \mathbf{I} + \mathbf{B}z^{-1} + \cdots + (\mathbf{B}z^{-1})^{l_1-1} = \sum_{j=0}^{l_1-1} (\mathbf{B}z^{-1})^j \tag{4.37}$$

For the i^{th} group in (4.37) we have

$$(\mathbf{B}z^{-1})^{(i-1)l_1} + (\mathbf{B}z^{-1})^{(i-1)l_1+1} + \cdots + (\mathbf{B}z^{-1})^{il_1-1} = (\mathbf{B}z^{-1})^{(i-1)l_1}\mathbf{Q} \tag{4.38}$$

We then obtain

$$\begin{aligned}\sum_{j=0}^{L-1} \mathbf{B}^j z^{-j} &= \mathbf{Q} + (\mathbf{B}z^{-1})^{l_1}\mathbf{Q} + \cdots + (\mathbf{B}z^{-1})^{(l_2-1)l_1}\mathbf{Q} \\ &= [\mathbf{I} + (\mathbf{B}z^{-1})^{l_1} + \cdots + (\mathbf{B}z^{-1})^{(l_2-1)l_1}]\mathbf{Q} \\ &= (\sum_{j=0}^{l_2-1} (\mathbf{B}z^{-1})^{jl_1})\mathbf{Q} \\ &= (\sum_{j=0}^{l_2-1} (\mathbf{B}z^{-1})^{jl_1})(\sum_{j=0}^{l_1-1} (\mathbf{B}z^{-1})^j)\end{aligned}$$

□

Lemma 4.4. If $L = \prod_{k=1}^{K} l_k$, where l_k is a positive integer, then

$$\sum_{j=0}^{L-1} \mathbf{B}^j z^{-j} = \prod_{k=1}^{K} (\sum_{j=0}^{l_k-1} (\mathbf{B}z^{-1})^{j \prod_{i=1}^{k-1} l_i}) \tag{4.39}$$

where $\mathbf{B}^0 = \mathbf{I}$ and the l_i are not necessarily distinct.

Proof : By induction on K from Lemma 4.3. □

We give an example with $L = 12$. Since 12 can be expressed as a product of three prime numbers 3, 2 and 2, we have

$$\begin{aligned}\sum_{j=0}^{11} \mathbf{B}^j z^{-j} &= (\mathbf{I} + \mathbf{B}z^{-1} + \mathbf{B}^2 z^{-2}) + (\mathbf{B}^3 z^{-3} + \mathbf{B}^4 z^{-4} + \mathbf{B}^5 z^{-5}) + \\ &\quad + (\mathbf{B}^6 z^{-6} + \mathbf{B}^7 z^{-7} + \mathbf{B}^8 z^{-8}) + (\mathbf{B}^9 z^{-9} + \mathbf{B}^{10} z^{-10} + \mathbf{B}^{11} z^{-11}) \\ &= (\mathbf{I} + \mathbf{B}z^{-1} + \mathbf{B}^2 z^{-2})((\mathbf{I} + \mathbf{B}^3 z^{-3}) + (\mathbf{B}^6 z^{-6} + \mathbf{B}^9 z^{-9})) \\ &= (\mathbf{I} + \mathbf{B}z^{-1} + \mathbf{B}^2 z^{-2})(\mathbf{I} + \mathbf{B}^3 z^{-3})(\mathbf{I} + \mathbf{B}^6 z^{-6})\end{aligned} \tag{4.40}$$

From the above example, it is easy to see that a large polynomial has been decomposed into a product of three small polynomials. Then $\sum_{j=0}^{11} \mathbf{B}^j z^{-j}$ can be implemented in a three-stage cascaded structure with only $4N$ multipliers, instead of $11N$ multipliers, where we suppose $\mathbf{B}$ is an $N \times N$ matrix. This reduction of the number of multipliers can be formally expressed by the following two lemmas.

Lemma 4.5. Suppose that $\mathbf{B}$ is an $N \times N$ matrix and q and p are constants, then

$$\det(\sum_{j=0}^{q-1} (\mathbf{B}z^{-1})^{jp}) = 1 + \sum_{j=1}^{(q-1)N} d_{jp} z^{-jp} \tag{4.41}$$

where $\mathbf{B}^0 = \mathbf{I}$ and d_{jp} is a combination of some elements in $\mathbf{B}^p, \cdots \mathbf{B}^{(q-1)p}$.

Proof: Let $z^p = \lambda$. Then

$$\det(\sum_{j=0}^{q-1} (\mathbf{B}z^{-1})^{jp}) = \det(\sum_{j=0}^{q-1} \mathbf{B}^{jp} \lambda^{-j}) \tag{4.42}$$

From Lemma 4.2, we know that $\det(\sum_{j=0}^{q-1} \mathbf{B}^{jp} \lambda^{-j})$ is an $N(q-1)^{th}$ order polynomial in λ^{-1}. It can then be expressed as

$$\det(\sum_{j=0}^{q-1} \mathbf{B}^{jp} \lambda^{-j}) = \sum_{j=0}^{(q-1)N} d_{jp} \lambda^{-j} \tag{4.43}$$

where d_{jp} is a combination of some elements in $\mathbf{B}^p, \cdots \mathbf{B}^{(q-1)p}$.

Since the coefficient of λ^0 is unity both in $\det(\mathbf{I}-\mathbf{B}^L\lambda^{-L})$ and in $\det(\mathbf{I}-\mathbf{B}\lambda^{-1})$, the coefficient of λ^0 in (4.43) must also be unity. Then

$$\det(\sum_{j=0}^{q-1}\mathbf{B}^{jp}\lambda^{-j}) = 1 + \sum_{j=1}^{(q-1)N} d_{jp}\lambda^{-j} \tag{4.44}$$

Replacing λ by z^p in (4.44), we then obtain

$$\det(\sum_{j=0}^{q-1}(\mathbf{B}z^{-1})^{jp}) = 1 + \sum_{j=1}^{(q-1)N} d_{jp}z^{-jp}$$

□

We see, from Lemma 4.5, that there are only $N(q-1)$ multipliers required to compute the associated convolution although $\det(\sum_{j=0}^{q-1}(\mathbf{B}z^{-1})^{jp})$ is an $Np(q-1)^{th}$ order polynomial. By extending this result, we have Lemma 4.6.

Lemma 4.6. If $L = \prod_{k=1}^{K} l_k$, there are $N(\sum_{k=1}^{K} l_k - K)$ multipliers required for computing the convolution associated with $\det(\sum_{j=0}^{L-1}\mathbf{B}^j z^{-j})$ by using the decomposition technique, where $\mathbf{B}$ is an $N \times N$ matrix.

Proof: From Lemma 4.4, we may have

$$\begin{aligned}\det(\sum_{j=0}^{L-1}\mathbf{B}^j z^{-j}) &= \det(\prod_{k=1}^{K}(\sum_{j=0}^{l_k-1}(\mathbf{B}z^{-1})^j \textstyle\prod_{i=1}^{k-1} l_i)) \\ &= \prod_{k=1}^{K}\det(\sum_{j=0}^{l_k-1}(\mathbf{B}z^{-1})^j \textstyle\prod_{i=1}^{k-1} l_i)\end{aligned} \tag{4.45}$$

We see, from the above equation, that $\det(\sum_{j=0}^{L-1}\mathbf{B}^j z^{-j})$ can be expressed as a product of K small polynomials. It can then be implemented in a K-stage cascaded structure. From Lemma 4.5, however, there are $N(l_k - 1)$ multipliers required in the k^{th} stage. For K stages, the total number of multipliers is then

$$\sum_{k=1}^{K} N(l_k - 1) = N(\sum_{k=1}^{K} l_k - K) \tag{4.46}$$

Since $\sum_{k=1}^{K} l_k - K$ may be much smaller than $\prod_{k=1}^{K} l_k - 1$, the number of multipliers may be greatly decreased after the decomposition. If $L = 2^K$ (or $K = \log_2 L$), for example, then the number in (4.46) becomes

$$\sum_{k=1}^{K} N(l_k) = NK = N\log_2 L \tag{4.47}$$

4.5. Time Domain Derivation

In this section we consider a time domain derivation of the parallel algorithms for direct-form recursive filters. This is of particular interest for time-varying recursive systems.

From Section 4.3 we see that the stabilized parallel algorithm derived in the Z domain has the form

$$y_n = \sum_{j=1}^{N} b_{jL} y_{n-jL} + u_n \tag{4.48}$$

Thus our goal is to derive, in the time domain, a parallel algorithm with the same form. In Section 4.5.1, we use a second-order recursive filter as an example to demonstrate two methods of achieving this form from the original (serial) algorithm. There are many other ways to achieve this form and one could ask if all these methods produce the same solution. Section 4.5.2 shows that uniqueness holds under one condition. If this condition is satisfied, all the methods give a unique solution and this solution is stable (if the original algorithm is stable), because it is the same as the one derived in Section 4.3, which has been proved to be stable. If the condition for uniqueness is not satisfied, we need to analyze the stability of the derived algorithm. This problem is discussed in Section 4.5.3.

4.5.1. Methods of derivation

In this subsection, we use a second-order recursive filter as an example to demonstrate two methods for obtaining parallel algorithms with the same form as that in (4.48). We assume that the condition for the unique solution (which will be discussed in the next subsection) holds, so all steps of the following derivations are valid.

A second-order recursive filter is expressed as

$$y_n = r_1 y_{n-1} + r_2 y_{n-2} + v_n \tag{4.49}$$

where $v_n = w_0 x_n + w_1 x_{n-1} + w_2 x_{n-2}$. To increase the degree of parallelism by a factor of $L = 6$, a modification is made so that the following form is obtained

$$\begin{aligned} y_n &= b_{1L} y_{n-L} + b_{2L} y_{n-2L} + u_n \\ &= b_6 y_{n-6} + b_{12} y_{n-12} + u_n \end{aligned} \tag{4.50}$$

To achieve this, we introduce two methods. The first one needs $(L-1)N$ steps to complete the derivation. Thus it is called the derivation without decomposition. The second one is called the derivation with decomposition because it takes only $(\sum_{k=1}^{K} l_k - K)N$ steps, where $L = \prod_{k=1}^{K} l_k$.

1	$(1^{\times}$ 2	$3)^{1}$	[0 1 2]
2	$(2^{\times}$ 3	$4)^{1}$	[0 2 3]
3	$(3^{\times}$ 4	$5)^{1}$	[0 3 4]
4	$(4^{\times}$ 5	$6)^{1}$	[0 4 5]
5	$(5^{\times}$ 6	$7)^{1}$	[0 5 6]
6	(6 $7^{\times}$	$8)^{1}$	[0 6 7]
7	(6 $8^{\times}$	$9)^{2}$	[0 6 8]
8	(6 $9^{\times}$	$10)^{3}$	[0 6 9]
9	(6 $10^{\times}$	$11)^{4}$	[0 6 10]
10	(6 $11^{\times}$	$12)^{5}$	[0 6 11]
11			[0 6 12]

Fig. 4.1. The derivation without decomposition ($N = 2$ and $L = 6$)

4.5.1.1. Derivation without decomposition

We are concerned with the derived form, but not with the exact values of coefficients. For simplicity, therefore, we use index notation to describe the procedure of our derivation, as shown in Fig. 4.1. The number in the first column denotes the step of the derivation. The numbers in the other columns represent an output. For example, 2 stands for y_{n-2} and 3 for y_{n-3}. The numbers between the square brackets at the i^{th} row represent an equation, which is the result from the immediately previous step and is called the equation at step i. For example, [0 6 7] on the sixth row denotes

$$y_n = b_6^{(5)} y_{n-6} + b_7^{(5)} y_{n-7} + v_n^{(5)} \tag{4.51}$$

where $b_j^{(5)}$ is the j^{th} coefficient of the equation, which is derived from step 5. A

group of numbers between parentheses with a superscript i denotes that an output, the corresponding number of which has a superscript "$\times$", is written explicitly by using the equation at step i. Since the equation at step 1 is expressed as

$$y_n = b_1^{(0)} y_{n-1} + b_2^{(0)} y_{n-2} + v_n^{(0)} \tag{4.52}$$

where $b_1^{(0)} = r_1$, $b_2^{(0)} = r_2$ and $v_n^{(0)} = v_n$, then $(6 \quad 7 \quad 8)^1$ can be written as

$$y_{n-6} = b_1^{(0)} y_{n-7} + b_2^{(0)} y_{n-8} + v_{n-6}^{(0)} \tag{4.53}$$

Thus $(6 \quad 7^{\times} \quad 8)^1$ on the sixth row is expressed as

$$y_{n-7} = \frac{1}{b_1^{(0)}} y_{n-6} - \frac{b_2^{(0)}}{b_1^{(0)}} y_{n-8} - \frac{1}{b_1^{(0)}} v_{n-6}^{(0)} \tag{4.54}$$

In the above we have assumed that $b_1^{(0)} \neq 0$.

The operation at step i is described as follows: First an output is written explicitly according to the coefficients between the parentheses. Next the expression of this output is substituted into the equation at step i, which is determined by the coefficients between the square brackets and is derived from step $i-1$. An equation for the next step is then obtained. An example follows. At step 6 in Fig. 4.1, we write y_{n-7} explicitly, using the equation at step 1. The result is given in (4.54). Next we substitute it into (4.51), the equation at step 6. Then we obtain

$$\begin{aligned} y_n &= (b_6^{(5)} + \frac{b_7^{(5)}}{b_1^{(0)}}) y_{n-6} + (-\frac{b_7^{(5)} b_2^{(0)}}{b_1^{(0)}}) y_{n-8} + (-\frac{b_7^{(5)}}{b_1^{(0)}}) v_{n-6}^{(0)} + v_n^{(5)} \\ &= b_6^{(6)} y_{n-6} + b_8^{(6)} y_{n-8} + v_n^{(6)} \end{aligned} \tag{4.55}$$

It is easy to verify, from Fig. 4.1, that the final result after ten iterations can be expressed as

$$y_n = b_6 y_{n-6} + b_{12} y_{n-12} + u_n \tag{4.56}$$

where

$$u_n = v_n + \sum_{j=1}^{10} d_j v_{n-j} \tag{4.57}$$

Because ten extra multipliers are introduced for computing u_n, the complexity after the modification is greatly increased. This number of multipliers is just $(L-1)N$. That is why we call this method derivation without decomposition. In the following, we describe a more efficient method with decomposition.

4.5.1.2. Derivation with decomposition

1	$(1^{\times}\ 2\quad 3)^{1}$	[0 1 2]
2	$(2^{\times}\ 3\quad 4)^{1}$	[0 2 3]
3	$(3\quad 4^{\times}\ 5)^{1}$	[0 3 4]
4	$(3\quad 5^{\times}\ 6)^{2}$	[0 3 5]
5	$(3^{\times}\ 6\quad 9)^{5}$	[0 3 6]
6	$(6\quad 9^{\times}\ 12)^{5}$	[0 6 9]
7		[0 6 12]

Fig. 4.2. Derivation with decomposition ($N = 2$ and $L = 6$)

The index notation of this derivation is depicted in Fig. 4.2. Since we are considering the decomposition, the detailed derivation at each step (as listed in Appendix 4.4) must be studied carefully.

From Appendix 4.4 it is easily verified that u_n can be computed in two stages, that is,

$$\tilde{u}_n = v_n + \sum_{j=1}^{4} d_j^{(1)} v_{n-j} \tag{4.58}$$

and

$$u_n = \tilde{u}_n + \sum_{j=1}^{2} d_{3j}^{(2)} \tilde{u}_{n-3j} \tag{4.59}$$

Therefore, only six multipliers are required for computing u_n.

1	$(1^{\times}$	2	3	$4)^{1}$	[0 1 2 3]
2	(2	$3^{\times}$	4	$5)^{1}$	[0 2 3 4]
3	(2	4	$5^{\times}$	$6)^{2}$	[0 2 4 5]
4	$(2^{\times}$	4	6	$8)^{4}$	[0 2 4 6]
5	(4	$6^{\times}$	8	$10)^{4}$	[0 4 6 8]
6	(4	8	$10^{\times}$	$12)^{5}$	[0 4 8 10]
7	$(4^{\times}$	8	12	$16)^{7}$	[0 4 8 12]
8	(8	$12^{\times}$	16	$20)^{7}$	[0 8 12 16]
9	(8	16	$20^{\times}$	$24)^{8}$	[0 8 16 20]
10					[0 8 16 24]

Fig. 4.3. Derivation with decomposition ($N = 3$ and $L = 8$)

The idea described above can easily be extended to more complicated cases. Fig. 4.3 depicts an example of $N = 3$ and $L = 8$. It is easy to prove that u_n in this example can be computed in three stages, using only nine multipliers (which is $N\log_2 L$).

4.5.2. Condition for unique solution

In section 4.5.1, we described two methods for obtaining a time domain derivation. However, there are many other methods of achieving the same goal. Two of them are depicted in Fig. 4.4. In this subsection we give the condition for the unique solution of the stabilized parallel algorithm.

The problem is defined as follows: Suppose that

$$(1-\sum_{j=1}^{N} r_j z^{-j})(\sum_{j=0}^{(L-1)N} a_j z^{-j}) = \sum_{j=0}^{N} b_{jL} z^{-jL} \tag{4.60}$$

where $b_0 = 1$ and $1-\sum_{j=1}^{N} r_j z^{-j}$ is the determinant of the Z transform of a stable recursive filter. The question is whether there exists a unique solution for determining $LN+1$ unknowns a_j for $0 \le j \le (L-1)N$ and b_{jL} for $1 \le j \le N$.

We first modify (4.60). Multiplying both sides of (4.60) by $\frac{1}{1-\sum_{j=1}^{N} r_j z^{-j}}$, then

$$\sum_{j=0}^{(L-1)N} a_j z^{-j} = \frac{\sum_{j=0}^{N} b_{jL} z^{-jL}}{1-\sum_{j=1}^{N} r_j z^{-j}} \tag{4.61}$$

By a power series expansion, we have

$$\begin{aligned} \frac{1}{1-\sum_{j=1}^{N} r_j z^{-j}} &= 1+\sum_{j=1}^{N} r_j z^{-j} + (\sum_{j=1}^{N} r_j z^{-j})^2 + \cdots \\ &= \sum_{k=0}^{\infty} (\sum_{j=1}^{N} r_j z^{-j})^k \\ &= \sum_{k=0}^{\infty} s_k z^{-k} \end{aligned} \tag{4.62}$$

where $s_0 = 1$ and s_k is a combination of some r_j. (The calculation of s_k for $1 \le k \le (L-1)N$ can be found in Appendix 4.5.) The above expansion is convergent for z^{-1} in the unit circle because the original system is stable.

1	$(1 \quad 2^{\times} \quad 3)^{1}$	$[0 \quad 1 \quad 2]$
2	$(1^{\times} \quad 2 \quad 3)^{1}$	$[0 \quad 1 \quad 3]$
3	$(2^{\times} \quad 3 \quad 5)^{2}$	$[0 \quad 2 \quad 3]$
4	$(3 \quad 5^{\times} \quad 6)^{3}$	$[0 \quad 3 \quad 5]$
5	$(3^{\times} \quad 6 \quad 9)^{5}$	$[0 \quad 3 \quad 6]$
6	$(6 \quad 9^{\times} \quad 12)^{5}$	$[0 \quad 6 \quad 9]$
7		$[0 \quad 6 \quad 12]$

(a)

1	$(1^{\times} \quad 2 \quad 3)^{1}$	$[0 \quad 1 \quad 2]$
2	$(2 \quad 3^{\times} \quad 4)^{1}$	$[0 \quad 2 \quad 3]$
3	$(2^{\times} \quad 4 \quad 6)^{3}$	$[0 \quad 2 \quad 4]$
4	$(4^{\times} \quad 6 \quad 8)^{3}$	$[0 \quad 4 \quad 6]$
5	$(6 \quad 8^{\times} \quad 10)^{3}$	$[0 \quad 6 \quad 8]$
6	$(6 \quad 10^{\times} \quad 12)^{4}$	$[0 \quad 6 \quad 10]$
7		$[0 \quad 6 \quad 12]$

(b)

Fig. 4.4. Two other methods for the time domain derivation ($N = 2$ and $L = 6$)

Using (4.61) and (4.62), we can rewrite (4.60) as

$$\sum_{j=0}^{(L-1)N} a_j z^{-j} = (\sum_{j=0}^{N} b_{jL} z^{-jL})(\sum_{k=0}^{\infty} s_k z^{-k}) \tag{4.63}$$

From (4.63) it is easy to verify that, once b_{jL} for $0 \leq j \leq N$ is determined, there is an unique solution for a_j for $0 \leq j \leq (L-1)N$. Therefore, our problem is reduced

to finding the condition for the unique solution of N unknowns b_{jL} for $1 \leq j \leq N$.

Lemma 7. Consider an equation of the form (4.60) or (4.63). There exists a unique solution for b_{jL} for $1 \leq j \leq N$ if and only if the following inequality is satisfied

$$\det \begin{pmatrix} s_{(L-1)N+1-L} & s_{(L-1)N+1-2L} & \cdots & s_{L-N+1} \\ s_{(L-1)N+2-L} & s_{(L-1)N+2-2L} & \cdots & s_{L-N+2} \\ \vdots & \vdots & \ddots & \vdots \\ s_{LN-1-L} & s_{LN-1-2L} & \cdots & s_{L-1} \end{pmatrix} \neq 0 \qquad (4.64)$$

where $s_k = 0$ for $k < 0$.

Proof: We define $a_j = 0$ for $j > (L-1)N$. Then we can write

$$\sum_{j=0}^{(L-1)N} a_j z^{-j} = \sum_{j=0}^{\infty} a_j z^{-j} \qquad (4.65)$$

Substituting (4.65) into (4.63), we have

$$\sum_{j=0}^{\infty} a_j z^{-j} = (\sum_{j=0}^{N} b_{jL} z^{-jL})(\sum_{k=0}^{\infty} s_k z^{-k}) \qquad (4.66)$$

Except for the first $(L-1)N+1$ linear equations, which are used to determine a_j for $0 \leq j \leq (L-1)N$, from (4.66) we can write an infinite set of linear equations as

$$\begin{aligned} 0 &= s_{(L-1)N+1}\, b_0 + s_{(L-1)N+1-L}\, b_L + \cdots s_{L-N+1}\, b_{(N-1)L} \\ 0 &= s_{(L-1)N+2}\, b_0 + s_{(L-1)N+2-L}\, b_L + \cdots s_{L-N+2}\, b_{(N-1)L} \\ &\vdots \qquad\qquad \vdots \qquad\qquad \vdots \\ 0 &= s_{LN}\, b_0 + s_{LN-L}\, b_L + \cdots s_L\, b_{(N-1)L} + s_0\, b_{NL} \\ &\vdots \qquad\qquad \vdots \qquad\qquad \vdots \qquad\qquad \vdots \end{aligned} \qquad (4.67)$$

where $s_k = 0$ for $k < 0$.

Although there is a countably infinite number of linear equations in (4.67), only at most N of them are independent. This is proved as follows: Substituting

(4.65) into (4.60), then

$$(1-\sum_{j=1}^{N} r_j z^{-j})(\sum_{j=0}^{\infty} a_j z^{-j}) = \sum_{j=0}^{N} b_{jL} z^{-jL} \tag{4.68}$$

Since $\sum_{j=0}^{N} b_{jL} z^{-jL}$ is an NL^{th} order polynomial in z^{-1}, the coefficient for any term with an order higher than NL must be zero. Thus we can write

$$\begin{aligned}
&a_{NL+1} + r_1 a_{NL} + r_2 a_{NL-1} + \cdots + r_N a_{(L-1)N+1} = 0 \\
&a_{NL+2} + r_1 a_{NL+1} + r_2 a_{NL} + \cdots + r_N a_{(L-1)N+2} = 0 \\
&a_{NL+3} + r_1 a_{NL+2} + r_2 a_{NL+1} + \cdots + r_N a_{(L-1)N+3} = 0 \\
&\quad\vdots \qquad\qquad\qquad\quad \vdots \qquad\qquad \vdots
\end{aligned} \tag{4.69}$$

From the above equations it is easy to see that once the first N coefficients a_j for $(L-1)N+1 \le j \le NL$ are known, a_{NL+1} can be determined by the first equation, then a_{NL+2} is determined by the second and then a_{NL+3} by the third, and so on. Therefore, only at most N linear equations among those equations in (4.67) are independent.

To compute b_{jL} for $1 \le j \le N$, we choose the first N linear equations from (4.67). Since $b_0 = s_0 = 1$, we then have

$$\begin{aligned}
0 &= s_{(L-1)N+1} + s_{(L-1)N+1-L}\, b_L + \cdots + s_{L-N+1}\, b_{(N-1)L} \\
0 &= s_{(L-1)N+2} + s_{(L-1)N+2-L}\, b_L + \cdots + s_{L-N+2}\, b_{(N-1)L} \\
&\vdots \qquad\qquad \vdots \qquad\qquad\qquad \vdots \\
0 &= s_{LN} \qquad + s_{LN-L}\, b_L \qquad + \cdots + s_L\, b_{(N-1)L} \qquad + b_{NL}
\end{aligned} \tag{4.70}$$

or in a matrix form,

$$\begin{pmatrix}
s_{(L-1)N+1-L} & \cdots & s_{L-N+1} & 0 \\
s_{(L-1)N+2-L} & \cdots & s_{L-N+2} & 0 \\
\vdots & \ddots & \vdots & \vdots \\
s_{LN-1-L} & \cdots & s_{L-1} & 0 \\
s_{LN-L} & \cdots & s_L & 1
\end{pmatrix}
\begin{pmatrix} b_L \\ b_{2L} \\ \vdots \\ b_{(N-1)L} \\ b_{NL} \end{pmatrix}
= -
\begin{pmatrix} s_{(L-1)N+1} \\ s_{(L-1)N+2} \\ \vdots \\ s_{LN-1} \\ s_{LN} \end{pmatrix} \tag{4.71}$$

where $s_k = 0$ for $k < 0$. From (4.71), it is easy to see that the necessary and sufficient condition for the unique solution of b_{jL} for $1 \leq j \leq N$ is

$$\det \begin{pmatrix} s_{(L-1)N+1-L} & s_{(L-1)N+1-2L} & \cdots & s_{L-N+1} & 0 \\ s_{(L-1)N+2-L} & s_{(L-1)N+2-2L} & \cdots & s_{L-N+2} & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ s_{LN-1-L} & s_{LN-1-2L} & \cdots & s_{L-1} & 0 \\ s_{LN-L} & s_{LN-2L} & \cdots & s_L & 1 \end{pmatrix} \neq 0 \tag{4.72}$$

or

$$\det \begin{pmatrix} s_{(L-1)N+1-L} & s_{(L-1)N+1-2L} & \cdots & s_{L-N+1} \\ s_{(L-1)N+2-L} & s_{(L-1)N+2-2L} & \cdots & s_{L-N+2} \\ \vdots & \vdots & \ddots & \vdots \\ s_{LN-1-L} & s_{LN-1-2L} & \cdots & s_{L-1} \end{pmatrix} \neq 0$$

where $s_k = 0$ for $k < 0$. □

4.5.3. Stability

In the previous subsection we derived the condition for the unique solution of the stabilized parallel algorithm. In certain problems, however, this condition does not hold. Since the solutions in these cases are not the same, we need to analyze the stability of the derived algorithm. The following is an example with $N = 2$ and $L = 6$.

From (4.64) we can express the condition of the unique solution for $N = 2$ and $L = 6$ as

$$s_5 \neq 0 \tag{4.73}$$

By using (4.A5.13) and (4.6), S_5 can be written as

$$s_5 = r_5^{(4)} = r_1(r_1^2 + 3r_2)(r_1^2 + r_2) \tag{4.74}$$

Therefore, the solutions will not be unique if either r_1, $r_1^2 + 3r_2$ or $r_1^2 + r_2$ is equal to zero.

To investigate stability, we consider one of the above three cases, that is,

$$r_1^2 + 3r_2 = 0 \tag{4.75}$$

Since $b_j = 0$ for $j \neq iL$ in (4.A5.2), we can obtain a linear homogeneous system of $N(L-1)$ equations. Using these equations plus the constraint (4.75), we can calculate the coefficient a_j and then obtain

$$\sum_{j=0}^{10} a_j z^{-j} = (1 + r_1 z^{-1} - r_2 z^{-2})(1 - r_2 z^{-2})(1 + a_6 z^{-6}) \tag{4.76}$$

Suppose that the original system is stable. The two poles z_1 and z_2 of the system are inside the unit circle. It is easy to verify that the roots of $1 + r_1 z^{-1} - r_2 z^{-2}$ are $-z_1$ and $-z_2$ and the roots of $1 - r_2 z^{-2}$ are $\pm(z_1 z_2)^{1/2}$. They are also inside the unit circle. To obtain a stabilized parallel algorithm, therefore, the absolute value of a_6 in (4.76) must be smaller than unity, that is,

$$|a_6| < 1 \tag{4.77}$$

Consider one extreme case when $a_6 = 0$. Then

$$\begin{aligned} \sum_{j=0}^{2} b_{jL} z^{-jL} &= (1 - r_1 z^{-1} - r_2 z^{-2})(\sum_{j=0}^{10} a_j z^{-j}) \\ &= (1 - r_1 z^{-1} - r_2 z^{-2})(1 + r_1 z^{-1} - r_2 z^{-2})(1 - r_2 z^{-2}) \\ &= 1 - r_2^3 z^{-6} \end{aligned} \tag{4.78}$$

It can be verified that this result may be obtained by just using the conventional look-ahead technique described in Section 4.2.

Using the same method described above to the other two cases, we can see that (4.77) is the condition for obtaining a stabilized algorithm with a degree of parallelism $L = 6$ for a second-order recursive filter when the condition for the unique solution (4.73) is not satisfied.

4.6. Discussion

In this chapter we first showed that conventional look-ahead computation for obtaining parallel algorithms for recursive systems can cause numerical instability in the direct-form implementation of recursive filters. The reason is that the introduced poles cannot be guaranteed to be inside the unit circle. Then we introduced a new method for deriving, in the Z domain, an algorithm for direct-form recursive filters. We showed that not only is the degree of parallelism of this new algorithm increased, but the stability is also improved. If the original serial computation is stable, that is, the eigenvalues z_i of the companion matrix in the original algorithm are all in the unit circle, the eigenvalues z_i^L of the companion matrix in the parallel system will also be in the unit circle and closer to the origin than their corresponding z_i's. This enhanced stability may mean that less bits per word are necessary in our parallel implementation.

The penalty for using the new method is a great increase in complexity. For example, an extra $N(L-1)$ multipliers are required if we want to increase the degree of parallelism by a factor of L for an N^{th} order recursive filter. However, to implement the original serial algorithm requires only $2N+1$ multipliers. To alleviate this problem, we introduced a decomposition technique. After decomposition, the extra multipliers introduced are decreased to $N(\sum_{k=1}^{K} l_k - K)$ where $L = \prod_{k=1}^{K} l_k$.

We also considered the method for obtaining, in the time domain, parallel algorithms with the same form as the one obtained in the Z domain. The derived parallel algorithm can be applied to time-varying recursive systems which are an important tool in modern digital signal processing. There are many different ways

to achieve the same goal. We then discussed the condition for uniqueness of the stabilized parallel algorithm. We showed that if this condition is not satisfied, the derived algorithms are stable only under certain conditions. Therefore, the problem of stability has to be carefully considered when one is deriving parallel algorithms for a time-varying recursive system.

The method described in this chapter can also be extended to obtain parallel algorithms for recursive filters in state-variable form. However, the decomposition technique is useful to reduce the increased complexity only if certain conditions are satisfied. A detailed description of this issue can be found in Appendix 4.6.

Appendix 4.1 :

In this appendix, we prove (4.5) and (4.6) by using induction.

From (4.2), (4.3) and (4.4), we see that (4.5) holds for $L = 2$. By induction, assume that the algorithm holds after $L-2$ iterations, that is,

$$y_n = \sum_{j=L-1}^{N+L-2} r_j^{(L-2)} y_{n-j} + \sum_{j=0}^{N+L-2} w_j^{(L-2)} x_{n-j} \qquad (4.A1.1)$$

where

$$r_j^{(L-2)} = \begin{cases} r_j^{(L-3)} + r_{L-2}^{(L-3)} r_{j-(L-2)}, & L-1 \le j < N+L-2; \\ r_{L-2}^{(L-3)} r_N, & j = N+L-2 \end{cases} \qquad (4.A1.2)$$

and

$$w_j^{(L-2)} = \begin{cases} w_j^{(L-3)}, & 0 \le j < L-2; \\ w_j^{(L-3)} + r_{L-2}^{(L-3)} w_{j-(L-2)}, & L-2 \le j < N+L-2; \\ r_{L-2}^{(L-3)} w_N, & j = N+L-2. \end{cases} \qquad (4.A1.3)$$

Using (4.1), we write $y_{n-(L-1)}$ explicitly as

$$y_{n-(L-1)} = \sum_{j=1}^{N} r_j y_{n-(L-1)-j} + \sum_{j=0}^{N} w_j x_{n-(L-1)-j} \qquad (4.A1.4)$$

Let $j' = j + L - 1$. Then

$$y_{n-(L-1)} = \sum_{j'=L}^{N+L-1} r_{j'-(L-1)} y_{n-j'} + \sum_{j'=L-1}^{N+L-1} w_{j'-(L-1)} x_{n-j'} \qquad (4.A1.5)$$

Substituting (4.A1.5) into (4.A1.1), we have

$$
\begin{aligned}
y_n &= r_{L-1}^{(L-2)}\Big(\sum_{j=L}^{N+L-1} r_{j-(L-1)}y_{n-j} + \sum_{j=L-1}^{N+L-1} w_{j-(L-1)}x_{n-j}\Big)+ \\
&+ \sum_{j=L}^{N+L-2} r_j^{(L-2)}y_{n-j} + \sum_{j=0}^{N+L-2} w_j^{(L-2)}x_{n-j} \\
&= \sum_{j=L}^{N+L-2} \big(r_j^{(L-2)} + r_{L-1}^{(L-2)}r_{j-(L-1)}\big)y_{n-j} + r_{L-1}^{(L-2)}r_n y_{n-(N+L-1)}+ \\
&+ \sum_{j=0}^{L-2} w_j^{(L-2)}x_{n-j} + \sum_{j=L-1}^{N+L-2} \big(w_j^{(L-2)} + r_{L-1}^{(L-2)}w_{j-(L-1)}\big)x_{n-j}+ \\
&+ r_{L-1}^{(L-2)}w_n x_{n-(N+L-1)} \\
&= \sum_{j=L}^{N+L-1} r_j^{(L-1)}y_{n-j} + \sum_{j=0}^{N+L-1} w_j^{(L-1)}x_{n-j}
\end{aligned}
\tag{4.A1.6}
$$

In the above equation,

$$
r_j^{(L-1)} = \begin{cases} r_j^{(L-2)} + r_{L-1}^{(L-2)}r_{j-(L-1)}, & L \le j < N+L-1; \\ r_{L-1}^{(L-2)}r_N, & j = N+L-1 \end{cases}
\tag{4.A1.7}
$$

and

$$
w_j^{(L-1)} = \begin{cases} w_j^{(L-2)}, & 0 \le j < L-1; \\ w_j^{(L-2)} + r_{L-1}^{(L-2)}w_{j-(L-1)}, & L-1 \le j < N+L-1; \\ r_{L-1}^{(L-2)}w_N, & j = N+L-1. \end{cases}
\tag{4.A1.8}
$$

Appendix 4.2 :

In the following, we prove

$$\sum_{j=0}^{N+L-1} w_j^{(L-1)} z^{-j} = (\sum_{j=0}^{N} w_j z^{-j}) D(z) \qquad (4.A2.1)$$

where

$$D(z) = 1 + \sum_{j=1}^{L-1} r_j^{(j-1)} z^{-j} \qquad (4.A2.2)$$

The proof of the denominator of $H(z)$ in (4.8) is similar.

First let $L = 2$. Since we have, from (4.6),

$$w_j^{(1)} = \begin{cases} w_0, & j = 0; \\ w_j + r_1 w_{j-1}, & 1 \le j < N+1; \\ r_1 w_N, & j = N+1 \end{cases}$$

thus

$$\begin{aligned} \sum_{j=0}^{N+1} w_j^{(1)} z^{-j} &= w_0 + \sum_{j=1}^{N} (w_j + r_1 w_{j-1}) z^{-j} + r_1 w_N z^{-(N+1)} \\ &= w_0 + \sum_{j=1}^{N} w_j z^{-j} + r_1 (\sum_{j=1}^{N} w_{j-1} z^{-j} + w_N z^{-(N+1)}) \qquad (4.A2.3) \\ &= \sum_{j=0}^{N} w_j z^{-j} + r_1 \sum_{j=1}^{N+1} w_{j-1} z^{-j} \end{aligned}$$

Let $j' = j - 1$. Then

$$\sum_{j=1}^{N+1} w_j z^{-j} = z^{-1} \sum_{j'=0}^{N} w_{j'} z^{-j'} \qquad (4.A2.4)$$

Substituting (4.A2.4) into (4.A2.3), we obtain

$$\begin{aligned} \sum_{j=0}^{N+1} w_j^{(1)} z^{-j} &= \sum_{j=0}^{N} w_j z^{-j} + r_1 z^{-1} \sum_{j=0}^{N} w_j z^{-j} \\ &= (\sum_{j=0}^{N} w_j z^{-j}) D(z) \end{aligned} \qquad (4.A2.5)$$

where $D(z) = 1 + r_1 z^{-1}$.

By induction, assume that (4.A2.1) holds for $L-2$, that is,

$$\sum_{j=0}^{N+L-2} w_j^{(L-2)} z^{-j} = (\sum_{j=0}^{N} w_j z^{-j})(1 + \sum_{j=1}^{L-2} r_j^{(j-1)} z^{-j}) \qquad (4.A2.6)$$

Since we can also have, from (4.6),

$$w_j^{(L-1)} = \begin{cases} w_j^{(L-2)}, & 0 \le j < L-1; \\ w_j^{(L-2)} + r_{L-1}^{(L-2)} w_{j-(L-1)}, & L-1 \le j < N+L-1; \\ r_{L-1}^{(L-2)} w_N, & j = N+L-1 \end{cases}$$

thus

$$\begin{aligned} \sum_{j=0}^{N+L-1} w_j^{(L-1)} z^{-j} &= \sum_{j=0}^{L-2} w_j^{(L-2)} z^{-j} + \sum_{j=L-1}^{N+L-2} (w_j^{(L-2)} + r_{L-1}^{(L-2)} w_{j-(L-1)}) z^{-j} + \\ &\quad + r_{L-1}^{(L-1)} w_N z^{-(N+L-1)} \\ &= \sum_{j=0}^{L-2} w_j^{(L-2)} z^{-j} + \sum_{L-1}^{N+L-2} w_j^{(L-2)} z^{-j} + \\ &\quad + r_{L-1}^{(L-2)} (\sum_{j=L-1}^{N+L-2} w_{j-(L-1)} z^{-j} + w_N z^{-(N+L-1)}) \\ &= \sum_{j=0}^{N+L-2} w_J^{(L-2)} z^{-j} + r_{L-1}^{(L-2)} \sum_{j=L-1}^{N+L-1} w_{j-(L-1)} z^{-j} \end{aligned} \qquad (4.A2.7)$$

But

$$\sum_{j=L-1}^{N+L-1} w_{j-(L-1)} z^{-j} = z^{-(L-1)} \sum_{j=0}^{N} w_j z^{-j} \qquad (4.A2.8)$$

Thus

$$\sum_{j=0}^{N+L-1} w_j^{(L-1)} z^{-j} = \sum_{j=0}^{N+L-2} w_j^{(L-2)} z^{-j} + r_{L-1}^{(L-2)} z^{-(L-1)} \sum_{j=0}^{N} w_j z^{-j} \qquad (4.A2.9)$$

Subtituting (4.A2.6) into (4.A2.9), we obtain (4.A2.1).

Appendix 4.3 :

In this appendix, we give an efficient method for computing $\mathbf{B}^k$. For simplicity, we show it in an example of $N = 4$.

First we mention the shift matrix

$$\mathbf{T} = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & & 0 \\ \vdots & \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ 0 & 0 & 0 & \cdots & 0 \end{pmatrix} \tag{4.A3.1}$$

The effect of premultiplication by $\mathbf{T}$ is to upshift a matrix by one row and replace its last row by zeros. For example,

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} = \begin{pmatrix} a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{4.A3.2}$$

If the elements on the last row in the shift matrix are not all equal to zero, it is easy to see that nothing in the resulting matrix in (4.A3.2) is changed except that the elements on the last row will not all be equal to zero.

Now consider $\mathbf{B}^2$, that is,

$$\mathbf{B}^2 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ r_4 & r_3 & r_2 & r_1 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ r_4 & r_3 & r_2 & r_1 \end{pmatrix} \tag{4.A3.3}$$

Using the property of $\mathbf{T}$, we have

$$\mathbf{B}^2 = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ r_4 & r_3 & r_2 & r_1 \\ r_5^{(1)} & r_4^{(1)} & r_3^{(1)} & r_2^{(1)} \end{pmatrix} \tag{4.A3.4}$$

where

$$r_j^{(1)} = \begin{cases} r_j + r_1 r_{j-1}, & 2 \le j < 4; \\ r_1 r_4, & j = 5. \end{cases}$$

By induction, suppose that, for $k = K-1$ where $k > N$, we have

$$\mathbf{B}^{K-1} = \begin{pmatrix} r_{K-1}^{(K-5)} & r_{K-2}^{(K-5)} & r_{K-3}^{(K-5)} & r_{K-4}^{(K-5)} \\ r_{K}^{(K-4)} & r_{K-1}^{(K-4)} & r_{K-2}^{(K-4)} & r_{K-3}^{(K-4)} \\ r_{K+1}^{(K-3)} & r_{K}^{(K-3)} & r_{K-1}^{(K-3)} & r_{K-2}^{(K-3)} \\ r_{K+2}^{(K-2)} & r_{K+1}^{(K-2)} & r_{K}^{(K-2)} & r_{K-1}^{(K-2)} \end{pmatrix} \tag{4.A3.5}$$

where $r_j^{(i)}$ can be computed by using the iterative algorithm in (4.6).

For $k = K$, then

$$\mathbf{B}^{K} = \mathbf{B}\mathbf{B}^{K-1} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ r_4 & r_3 & r_2 & r_1 \end{pmatrix} \begin{pmatrix} r_{K-1}^{(K-5)} & r_{K-2}^{(K-5)} & r_{K-3}^{(K-5)} & r_{K-4}^{(K-5)} \\ r_{K}^{(K-4)} & r_{K-1}^{(K-4)} & r_{K-2}^{(K-4)} & r_{K-3}^{(K-4)} \\ r_{K+1}^{(K-3)} & r_{K}^{(K-3)} & r_{K-1}^{(K-3)} & r_{K-2}^{(K-3)} \\ r_{K+2}^{(K-2)} & r_{K+1}^{(K-2)} & r_{K}^{(K-2)} & r_{K-1}^{(K-2)} \end{pmatrix} \tag{4.A3.6}$$

From the property of the shift matrix $\mathbf{T}$, we know that $\mathbf{B}^{\mathbf{K}}$ should have the following form

$$\mathbf{B}^{K} = \begin{pmatrix} r_{K}^{(K-4)} & r_{K-1}^{(K-4)} & r_{K-2}^{(K-4)} & r_{K-3}^{(K-4)} \\ r_{K+1}^{(K-3)} & r_{K}^{(K-3)} & r_{K-1}^{(K-3)} & r_{K-2}^{(K-3)} \\ r_{K+2}^{(K-2)} & r_{K+1}^{(K-2)} & r_{K}^{(K-2)} & r_{K-1}^{(K-2)} \\ x_{41} & x_{42} & x_{43} & x_{44} \end{pmatrix} \tag{4.A3.7}$$

To compute x_{4j} for $1 \le j \le 4$ from (4.A3.6) requires $N^2 = 16$ multiplications. However, we have

$$\mathbf{B}\mathbf{B}^{K-1} = \mathbf{B}^{K-1}\mathbf{B} \tag{4.A3.8}$$

Then we can easily prove that

$$\begin{pmatrix} x_{41} & x_{42} & x_{43} & x_{44} \end{pmatrix} = \begin{pmatrix} r_{K+3}^{(K-1)} & r_{K+2}^{(K-1)} & r_{K+1}^{(K-1)} & r_{K}^{(K-1)} \end{pmatrix} \tag{4.A3.9}$$

where $r_j^{(K-1)}$ can be computed from $r_j^{(K-2)}$ by using the iterative algorithm in (4.6), that is,

$$r_j^{(K-1)} = \begin{cases} r_j^{(K-2)} + r_{K-1}^{(K-2)} r_{j-(K-1)}, & K \le j < K+3; \\ r_{K-1}^{(K-2)} r_4, & j = K+3. \end{cases}$$

From the above equation, we see that it takes only $N = 4$ multiplications to compute x_{4j}.

In general, consider that $\mathbf{B}$ is an $N \times N$ matrix. Then

for $k < N$,

$$\mathbf{B}^k = \begin{array}{c} \\ N-k \\ N-k+1 \end{array} \begin{array}{c} \begin{array}{cccccc} & & & k+1 & k+2 & \end{array} \\ \begin{pmatrix} 0 & \cdots & 1 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 1 & & 0 \\ \vdots & \ddots & \vdots & & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & 1 \\ r_N & \cdots & r_{N-k} & r_{N-k-1} & \cdots & r_1 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ r_{N+k-1}^{(k-1)} & \cdots & r_{N-1}^{(k-1)} & r_{N-2}^{(k-1)} & \cdots & r_k^{(k-1)} \end{pmatrix} \end{array} \tag{4.A3.10}$$

and for $k \geq N$,

$$\mathbf{B}^k = \begin{pmatrix} r_k^{(k-N)} & r_{k-1}^{(k-N)} & \cdots & r_{k-N+1}^{(k-N)} \\ r_{k+1}^{(k-N+1)} & r_k^{(k-N+1)} & \cdots & r_{k-N+2}^{(k-N+1)} \\ \vdots & \vdots & \ddots & \vdots \\ r_{N+k-1}^{(k-1)} & r_{N+k-2}^{(k-1)} & \cdots & r_k^{(k-1)} \end{pmatrix} \tag{4.A3.11}$$

Using the iterative algorithm in (4.6), the number of multiplications for computing $\mathbf{B^k}$ is Nk.

Appendix 4.4 :

This appendix lists the detailed derivation steps using an example of $N = 2$ and $L = 6$ with decomposition.

The original equation can be written as

$$y_n = b_1^{(0)} y_{n-1} + b_2^{(0)} y_{n-2} + v_n^{(0)}$$

where $b_1^{(0)} = r_1$, $b_2^{(0)} = r_2$ and $v_n^{(0)} = v_n$. In each step of the following, the first equation is associated with the denotation on the second column in Fig. 4.2, the second one is the derived equation for the next step and the third is the modified equation for the linear part. We assume that no divisors below are equal to zero.

Step 1:

$$y_{n-1} = b_1^{(0)} y_{n-2} + b_2^{(0)} y_{n-3} + v_{n-1}^{(0)}$$

$$y_n = b_2^{(1)} y_{n-2} + b_3^{(1)} y_{n-3} + v_n^{(1)}$$

$$v_n^{(1)} = v_n^{(0)} + b_1^{(0)} v_{n-1}^{(0)}$$

Step 2:

$$y_{n-2} = b_1^{(0)} y_{n-3} + b_2^{(0)} y_{n-4} + v_{n-2}^{(0)}$$

$$y_n = b_3^{(2)} y_{n-3} + b_4^{(2)} y_{n-4} + v_n^{(2)}$$

$$v_n^{(2)} = v_n^{(1)} + b_2^{(1)} v_{n-2}^{(0)}$$

Step 3:

$$y_{n-4} = \frac{1}{b_1^{(0)}} y_{n-3} - \frac{b_2^{(0)}}{b_1^{(0)}} y_{n-5} - \frac{1}{b_1^{(0)}} v_{n-3}^{(0)}$$

$$y_n = b_3^{(3)} y_{n-3} + b_5^{(5)} y_{n-5} + v_n^{(3)}$$

$$v_n^{(3)} = v_n^{(2)} - \frac{b_1^{(0)}}{b_4^{(2)}} v_{n-3}^{(0)}$$

Step 4:

$$y_{n-5} = \frac{1}{b_2^{(1)}} y_{n-3} - \frac{b_3^{(1)}}{b_2^{(1)}} y_{n-6} - \frac{1}{b_2^{(1)}} v_{n-3}^{(1)}$$

$$y_n = b_3^{(4)} y_{n-3} + b_6^{(4)} y_{n-6} + v_n^{(4)}$$

$$v_n^{(4)} = v_n^{(3)} - \frac{b_5^{(3)}}{b_2^{(1)}} v_{n-3}^{(1)}$$

Step 5:

$$y_{n-3} = b_3^{(4)} y_{n-6} + b_6^{(4)} y_{n-9} + v_{n-3}^{(4)}$$

$$y_n = b_6^{(5)} y_{n-6} + b_9^{(5)} y_{n-9} + v_n^{(5)}$$

$$v_n^{(5)} = v_n^{(4)} + b_3^{(4)} v_{n-3}^{(4)}$$

Step 6:

$$y_{n-9} = \frac{1}{b_3^{(4)}} y_{n-6} - \frac{b_6^{(4)}}{b_3^{(4)}} y_{n-12} - \frac{1}{b_3^{(4)}} v_{n-6}^{(4)}$$

$$y_n = b_6^{(6)} y_{n-6} + b_{12}^{(6)} y_{n-12} + v_n^{(6)}$$

$$v_n^{(6)} = v_n^{(5)} - \frac{b_9^{(5)}}{b_3^{(4)}} v_{n-6}^{(4)}$$

From the above equations, it is easy to verify that

$$v_n^{(4)} = v_n^{(0)} + \sum_{j=1}^{4} d_j^{(1)} v_{n-j}^{(0)} \qquad (4.A4.1)$$

and

$$v_N^{(6)} = v_n^{(4)} + \sum_{j=1}^{2} d_j^{(2)} v_{n-j}^{(4)} \qquad (4.A4.2)$$

Let $\tilde{u}_n = v_n^{(4)}$ and $u_n = v_n^{(6)}$. We then obtain (4.58) and (4.59).

Appendix 4.5 :

In this appendix, we calculate the value of s_k for $1 \leq k \leq (L-1)N$.

We first define $b_i = 0$ for $i \neq jL$. Then

$$\sum_{j=0}^{N} b_{jL} z^{-jL} = \sum_{j=0}^{NL} b_j z^{-j} \tag{4.A5.1}$$

Thus (4.60) can be rewritten as

$$(1 - \sum_{j=0}^{N} r_j z^{-j})(\sum_{j=0}^{(L-1)N} a_j z^{-j}) = \sum_{j=0}^{NL} b_j z^{-j} \tag{4.A5.2}$$

Using the above equation, we can write a linear system of $(L-1)N+1$ equations as

$$\begin{pmatrix} 1 & & & & & \\ -r_1 & 1 & & & & \\ & \ddots & \ddots & & & \\ -r_N & & -r_1 & 1 & & \\ & \ddots & & \ddots & \ddots & \\ 0 & & -r_N & & -r_1 & 1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \\ \vdots \\ a_{(L-1)N} \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \\ \vdots \\ b_{(L-1)N} \end{pmatrix} \tag{4.A5.3}$$

or

$$\mathbf{Ra} = \mathbf{b} \tag{4.A5.4}$$

where $\mathbf{R}$ is an $((L-1)N+1) \times (((L-1)N+1)$ triangular Toeplitz matrix, $\mathbf{a}$ and $\mathbf{b}$ are $((L-1)N+1) \times 1$ vectors. It is easy to verify that

$$\det(\mathbf{R}) = 1 \tag{4.A5.5}$$

Therefore, $\mathbf{R}^{-1}$ exists and (4.A5.4) can be rewritten as

$$\mathbf{a} = \mathbf{R}^{-1}\mathbf{b} \tag{4.A5.6}$$

Substituting (4.A5.1) into (4.63), we have

$$\sum_{j=0}^{(L-1)N} a_j z^{-j} = (\sum_{j=0}^{NL} b_j z^{-j})(\sum_{k=0}^{\infty} s_k z^{-k}) \tag{4.A5.7}$$

We can also write, from (4.A5.7), a linear system of $(L-1)N+1$ equations as

$$\mathbf{a} = \mathbf{S}\mathbf{b} \tag{4.A5.8}$$

where

$$\mathbf{a} = \begin{pmatrix} a_0 & a_1 & \cdots & a_{(L-1)N} \end{pmatrix}^T,$$

$$\mathbf{b} = \begin{pmatrix} b_0 & b_1 & \cdots & b_{(L-1)N} \end{pmatrix}^T$$

and

$$\mathbf{S} = \begin{pmatrix} 1 & & & & \\ s_1 & 1 & & & \\ s_2 & s_1 & 1 & & \\ & \ddots & \ddots & \ddots & \\ s_{(L-1)N} & & s_2 & s_1 & 1 \end{pmatrix}$$

Comparing (4.A5.6) and (4.A5.8), we obtain

$$\mathbf{S} = \mathbf{R}^{-1} \tag{4.A5.9}$$

Therefore, to calculate s_k for $1 \leq k \leq (L-1)N$, we can use the following equation

$$\mathbf{S}\mathbf{R} = \mathbf{I} \tag{4.A5.10}$$

Since $\mathbf{S}$ is also a triangular Toeplitz matrix, all the information needed is contained in the first column (or the last row). Then we can also use the following simplified equation for computing s_k.

$$\mathbf{R} \begin{pmatrix} 1 \\ s_1 \\ \vdots \\ s_{(L-1)N} \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \tag{4.A5.12}$$

It is easy to verify that, for $1 \leq k \leq (L-1)N$,

$$s_k = r_k^{(k-1)} \tag{4.A5.13}$$

where $r_k^{(k-1)}$ can be computed by using the iteration algorithm in (4.6).

Appendix 4.6

This appendix extends the method derived in Chapter 4 to the state-variable form implementation of recursive filters.

The state-variable form of an N^{th} order recursive filter can be expressed in two equations, that is, the state update equation

$$\mathbf{x}(n+1) = \mathbf{A}\mathbf{x}(n) + \mathbf{b}u(n) \tag{4.A6.1}$$

and the output equation

$$y(n) = \mathbf{c}^T\mathbf{x}(n) + du(n) \tag{4.A6.2}$$

where $\mathbf{A}$ is an $N \times N$ matrix, $\mathbf{b}$ and $\mathbf{c}$ are $N \times 1$ vectors, d is a scalar, $\mathbf{x}(n)$ represents an $N \times 1$ state vector and $u(n)$ and $y(n)$ are input and output, respectively.

To obtain a parallel algorithm, we consider the state update equation (4.A6.1) since only this equation contains recursion. Transforming (4.A6.1) into the Z domain, we have

$$\mathbf{X}(z)z = \mathbf{A}\mathbf{X}(z) + \mathbf{b}U(z) \tag{4.A6.3}$$

or

$$\frac{\mathbf{X}(z)}{U(z)} = \frac{\mathbf{b}z^{-1}}{\mathbf{I} - \mathbf{A}z^{-1}} \tag{4.A6.4}$$

where $\mathbf{I}$ is an identity matrix and $\mathbf{X}(z)$ and $U(z)$ represent the Z transforms of $\mathbf{x}(n)$ and $u(n)$, respectively.

Multiplying both numerator and denominator of (4.A6.4) by a factor $\sum_{j=0}^{M-1} \mathbf{A}^j z^{-j}$, then

$$\begin{aligned} \frac{\mathbf{X}(z)}{U(z)} &= \frac{(\sum_{j=0}^{M-1} \mathbf{A}^j z^{-j})\mathbf{b}z^{-1}}{(\mathbf{I} - \mathbf{A}z^{-1})(\sum_{j=0}^{M-1} \mathbf{A}^j z^{-j})} \\ &= \frac{(\sum_{j=0}^{M-1} \mathbf{A}^j z^{-j})\mathbf{b}z^{-1}}{\mathbf{I} - \mathbf{A}^M z^{-M}} \end{aligned} \tag{4.A6.5}$$

The above equation can be divided into two parts, that is, a recursive part

$$\frac{\mathbf{X}(z)}{\mathbf{V}(z)} = \frac{1}{\mathbf{I} - \mathbf{A}^M z^{-M}} \tag{4.A6.6}$$

and a linear part

$$\begin{aligned} \frac{\mathbf{V}(z)}{U(z)} &= (\sum_{j=0}^{M-1} \mathbf{A}^j z^{-j})\mathbf{b}z^{-1} \\ &= \sum_{j=0}^{M-1} \mathbf{A}^j \mathbf{b} z^{-(j+1)} \end{aligned} \tag{4.A6.7}$$

where $\mathbf{V}(z)$ is a vector of intermediate variables.

To analyze the degree of parallelism, we transform (4.A6.6) into the time domain and then obtain

$$\mathbf{x}(n+M) = \mathbf{A}^M \mathbf{x}(n) + \mathbf{v}(n+M) \tag{4.A6.8}$$

It can be seen from the above equation that M vectors of state variables $\mathbf{x}(n+i)$ for $0 \leq i \leq M-1$ can be computed simultaneously.

Since $\mathbf{A}^j\mathbf{b}$ is an $N \times 1$ vector and can be precomputed, to implement (4.A6.7) requires NM multipliers. The complexity of the modified system is increased. Unlike the case in the direct form implementation, however, this number of multipliers may not be reduced by using decomposition if $\mathbf{A}$ is an $N \times N$ full matrix.

Suppose that M can be factored as $M = \prod_{k=1}^{K} m_k$ where m_k is a positive integer. Applying the decomposition technique, $\sum_{j=0}^{M-1} \mathbf{A}^j z^{-j}$ can be expressed as a product form. Then

$$\mathbf{b}z^{-1}(\sum_{j=}^{M-1} \mathbf{A}^j z^{-j}) = \mathbf{b}z^{-1} \prod_{k=1}^{K} \sum_{j=0}^{m_k-1} (\mathbf{A}z^{-1})^j \prod_{i=1}^{k-1} m_i \tag{4.A6.9}$$

When $\mathbf{A}$ is an $N \times N$ full matrix, it is easy to verify that to implement (4.A6.9) requires $N^2(\sum_{k=1}^{K} m_k - K) + N$ multipliers where N^2 is due to full-matrix and vector

multiplications involved in the computation. Thus the decomposition technique is not useful if $N(\sum_{k=1}^{K} m_k - K + 1)$ is greater than $M = \prod_{k=1}^{K} m_k$.

To solve the above described problem, we recommend that $\mathbf{A}$ is a block-diagonal matrix with block size q by q for q much smaller than N. A matrix with this form can easily be obtained from a parallel combination of q^{th} order recursive filters. When $\mathbf{A}$ is a block-diagonal matrix with block size q by q, only qN multipliers are required for implementing each matrix-vector multiplication involved in (4.A6.9). Thus the number of multipliers is reduced to $qN(\sum_{k=1}^{K} m_k - K)+N$. Note that $\mathbf{A}$ will not be a simple diagonal matrix in most cases. Otherwise complex numbers will appear in the computation, which increases the complexity of the system. Consider the best case with $q = 2$. The number of multipliers required for implementing (4.A6.9) is minimized to $2N(\sum_{k=1}^{K} m_k - K)+N$, which may be much smaller than $N \prod_{k=1}^{K} m_k$.

CHAPTER 5

PIPELINED AND/OR PARALLEL ARCHITECTURES FOR DIRECT-FORM RECURSIVE FILTERS

5.1. Introduction

This chapter describes some efficient architectures associated with the stabilized parallel algorithms derived in the previous chapter. The two-level pipeline, first introduced by H. T. Kung and his colleagues [19,21], is a good method not only for achieving high throughput computation, but also for having less area in VLSI implementation in comparison with other parallel approaches. In Section 5.2 we show that, by using the stabilized parallel algorithms, an efficient two-level pipelined structure can easily be obtained. Another approach to achieving high throughput computation is by using parallel processing. We derive two parallel structures in Section 5.3. The first structure has the advantage of regularity while the second one can achieve a linear complexity in parallel size for cascaded second-order recursive filters. Since the two-level pipelined structure has less area in VLSI implementation than parallel structures, the system should desirably be implemented by first using pipelining to the maximum possible extent, and then using parallel processing in combination with pipelining if further increase in throughput is required. Therefore, we finally describe pipelined and parallel architectures for direct-form recursive filters in Section 5.4.

5.2. Two-Level Pipelined Structure

In this section, we derive an efficient two-level pipelined structure with a throughput of M for N^{th} order recursive filters. As we know, the algorithm derived in the previous chapter can be divided into two parts, the recursive part and the linear part. We first introduce a structure for the recursive part and show that our stabilized algorithms can easily be used for second-level pipelining. Then we describe a structure for the linear part. Since this structure is unidirectional without feed-back, it can have as many second-level pipelined stages as desired. Therefore, the two structures can easily be matched with the same throughput.

Recursive part :

For convenience, we rewrite (4.27) as follows

$$y_n = \sum_{j=1}^{N} b_{jM} y_{n-jM} + u_n \tag{5.1}$$

It is easy to construct a structure for computing this algorithm, as shown in Fig. 5.1. The system has N identical basic processing elements (or cells), each of which performs a simple multiplication-and-accumulation operation. It inputs the data u_n at the leftmost cell and outputs the result y_n at the rightmost cell. The result is then fed back immediately into the system from the rightmost cell as another input for later results. Since y_n depends only on y_{n-jM} for $j = 1, 2, \cdots N$, the required NM delays are evenly distributed in the system. By using cut-set localization rules described in Chapter 2, a two-level pipeline with M stages can be obtained. Therefore the throughput is M times higher than that of the system for computing the unmodified algorithm.

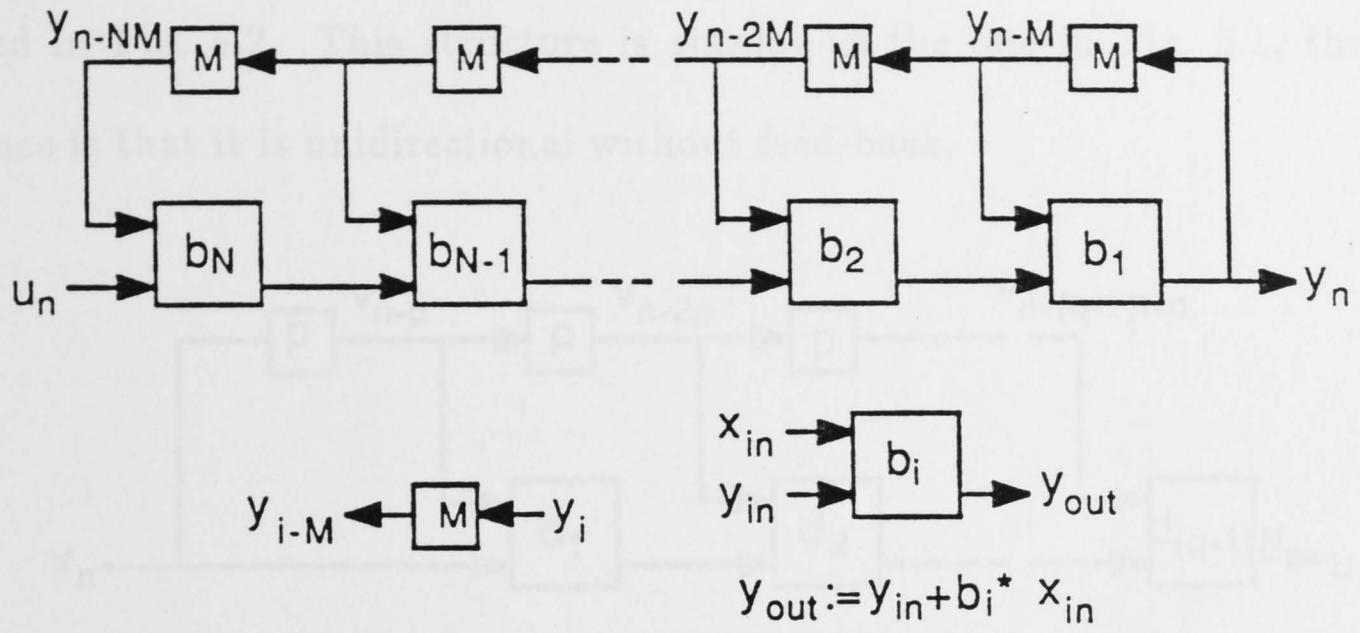


Fig. 5.1. The structure for the recursive part ($N = 2$ and $M = 4$)

Linear part :

From Lemmas 4.4 and 4.5, the impulse response function of the linear part $H_1(z)$ in (4.22) can be expressed as a product of $K+1$ polynomials in z^{-1} as

$$H_1(z) = (\sum_{j=0}^{N} w_j z^{-j}) \prod_{k=1}^{K} (1 + \sum_{j=1}^{(m_k-1)N} d_j^{(k)} z^{-j \prod_{i=1}^{k-1} m_i}) \tag{5.2}$$

where $M = \prod_{k=1}^{K} m_k$ for m_k is a positive integer and $d_j^{(k)}$ denotes the coefficient in the k^{th} stage in the product term. A $K+1$ stage cascaded structure may be applied for implementing this algorithm. Because of the similarity of these stages, we only consider one as follows

$$\frac{U(z)}{V(z)} = 1 + \sum_{j=1}^{(q-1)N} d_j z^{-jp} \tag{5.3}$$

where p and q are positive integers and $V(z)$ and $U(z)$ are the Z transforms of input and output, respectively.

Converting (5.3) into the time domain, we have

$$u_n = v_n + \sum_{j=1}^{(q-1)N} d_j v_{n-jp} \tag{5.4}$$

The above equation can then be computed by using a structure such as that depicted in Fig. 5.2. This structure is similar to the one in Fig. 5.1, the only difference is that it is unidirectional without feed-back.

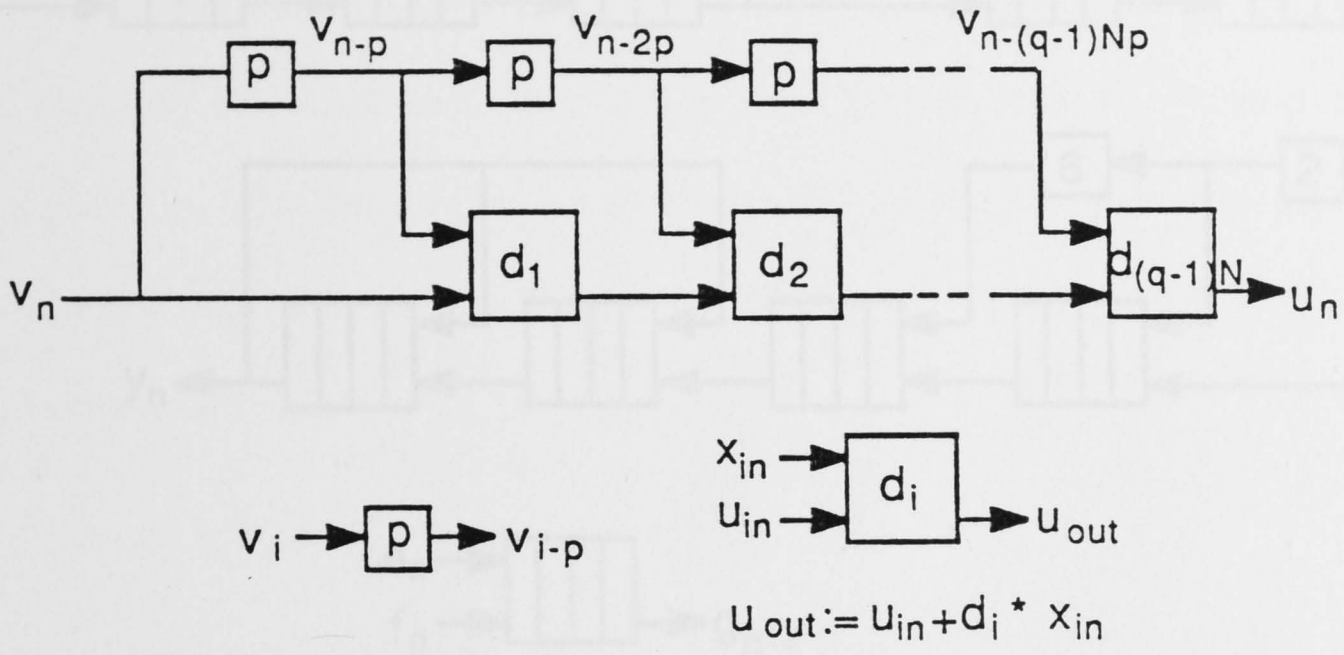


Fig. 5.2. The structure for computing (5.4)

Since the structure in Fig. 5.2 is unidirectional, as mentioned before it can have as many second-level pipelined stages as desired. Therefore, we can easily make Fig. 5.2 match Fig. 5.1 with the same throughput. An example of $N = 2$ and $M = 4$ is given in Fig. 5.3.

We see from the above discussion that N multipliers are required for computing the recursive part. For implementing $\sum_{j=0}^{N} w_j z^{-j}$, we need $N+1$ multipliers. Therefore, $N + 1 + N(\sum_{k=1}^{K} m_k - K)$ multipliers are required for computing the linear part. The total number of multipliers required for an N^{th} order recursive filter is then

$$N + N(\sum_{k=1}^{K} m_k - K) + N + 1 = N(\sum_{k=1}^{K} m_k - K + 2) + 1 \qquad (5.5)$$

When m_k is not a prime number and can be expressed as a product of some prime numbers, the small polynomial on the right-hand side of (5.2) can be further

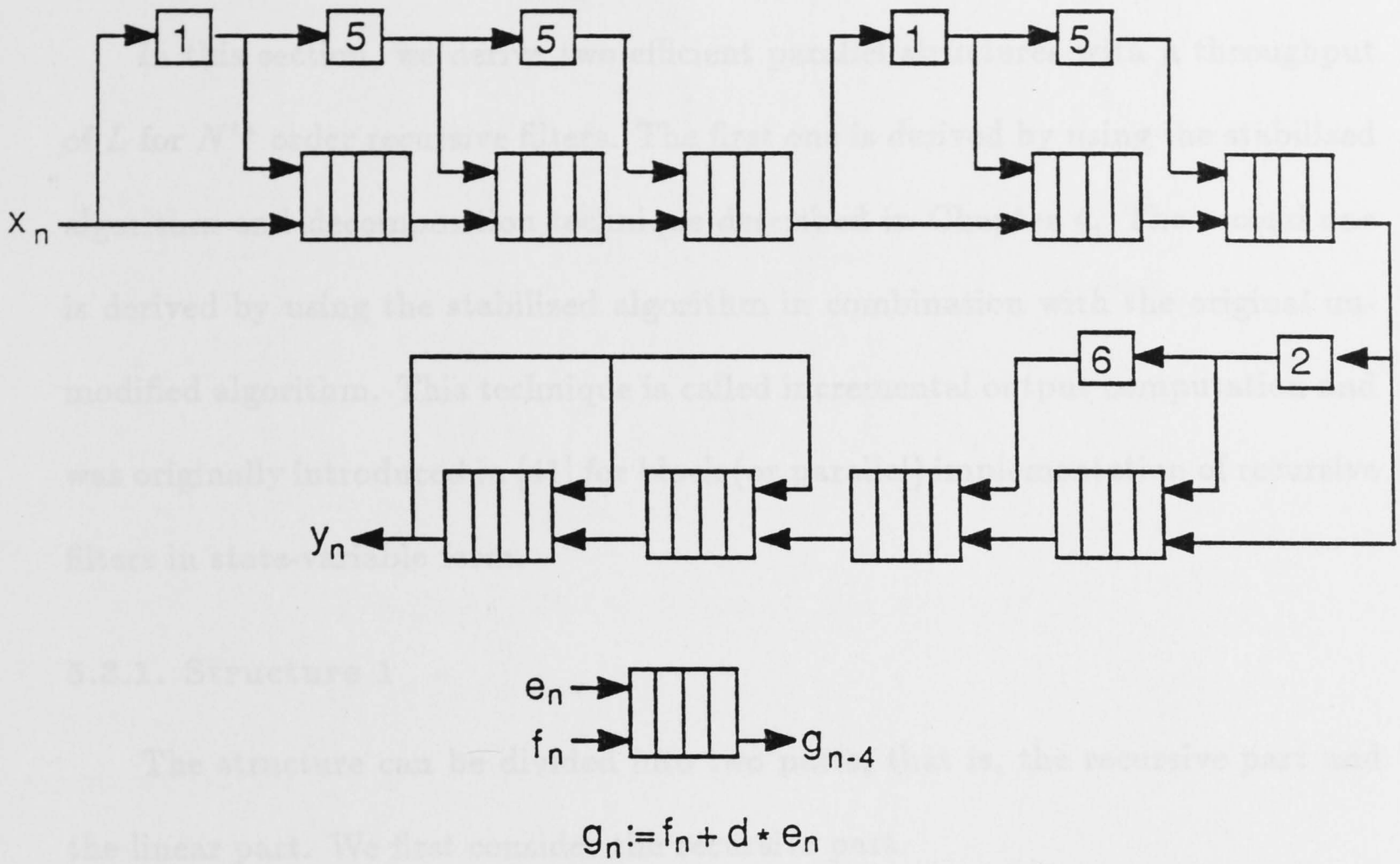


Fig. 5.3. The structure with four stages of second-level pipeline

for second-order recursive filters

decomposed. In the best case when $M = \prod_{k=1}^{K'} p_k$ where p_k is a prime number, therefore, the total number of multipliers can be reduced to $N(\sum_{k=1}^{K'} p_k - K' + 2) + 1$. If M is a power of two, for example, the total number of multipliers in (5.5) becomes

$$N + N\log_2 M + N + 1 = N(\log_2 M + 2) + 1 \tag{5.6}$$

5.3. Parallel Structures

In this section, we derive two efficient parallel structures with a throughput of L for N^{th} order recursive filters. The first one is derived by using the stabilized algorithm and decomposition technique described in Chapter 4. The second one is derived by using the stabilized algorithm in combination with the original unmodified algorithm. This technique is called incremental output computation and was originally introduced in [41] for block (or parallel) implementation of recursive filters in state-variable form.

5.3.1. Structure 1

The structure can be divided into two parts, that is, the recursive part and the linear part. We first consider the recursive part.

Recursive part :

The algorithm for the recursive part is expressed as

$$y_n = \sum_{j=1}^{N} b_j y_{n-jL} + u_n \tag{5.7}$$

Arranging the output y_n into L groups, we have

$$\text{for } 0 \leq l \leq L-1$$

$$y_{Ln+l} = \sum_{j=1}^{N} b_j y_{L(n-j)+l} + u_{Ln+l} \tag{5.8}$$

From the above equation it can be seen that to compute y_{Ln+l} we need $y_{L(n-j)+l}$ for $j = 1$ to N. However, they are just consecutive outputs in the same group. Therefore, the structure for computing (5.7) or (5.8) may consist of L identical $1-D$ sub-structures, which are independent of each other. Fig. 5.4 gives an example with $N = 2$ and $L = 4$.

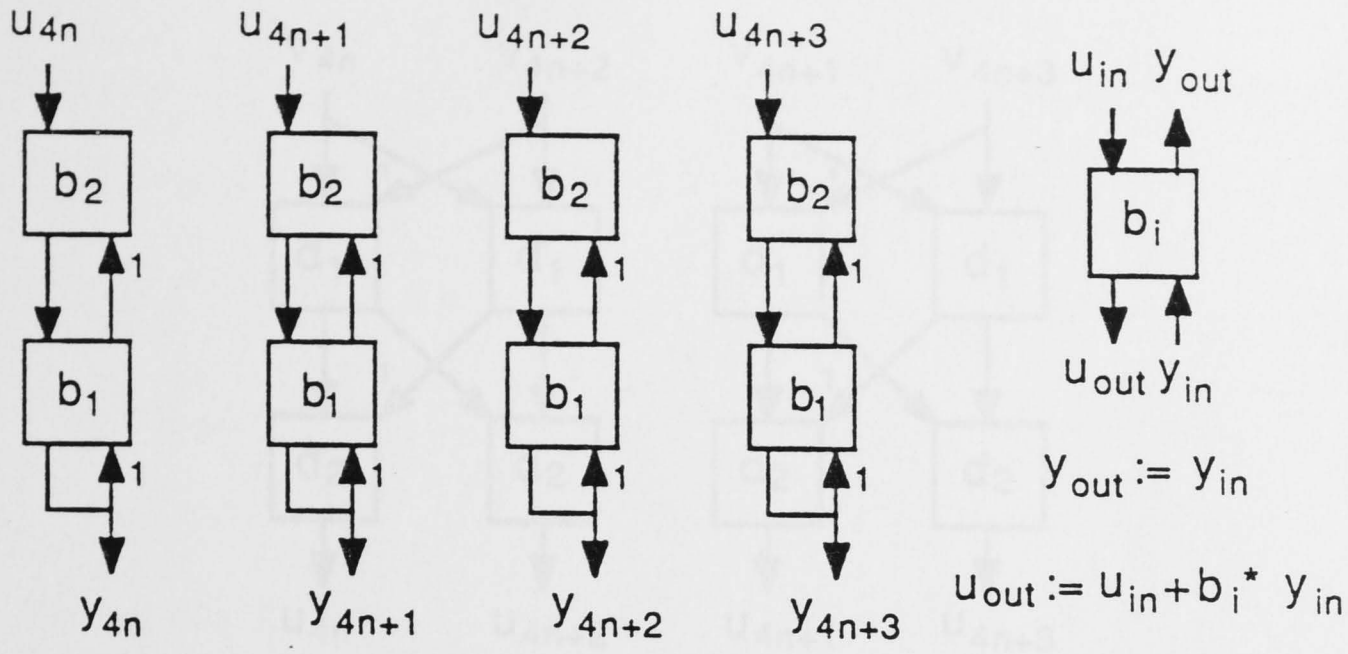


Fig. 5.4. The structure for the recursive part

($N = 2$ and $L = 4$)

Linear part :

By using the decomposition techique, the linear part can be expressed as

$$H_1(z) = (\sum_{j=0}^{N} w_j z^{-j}) \prod_{k=1}^{K} (1 + \sum_{j=1}^{(l_k-1)N} d_j^{(k)} z^{-j \prod_{i=1}^{k-1} l_i}) \tag{5.9}$$

where $L = \prod_{k=1}^{K} l_k$ with l_k a positive integer (or a prime number). A $K+1$ stage cascaded structure may be applied for computing this equation. However, to derive a parallel structure for each stage is not as simple as that in the two-level pipelined structure. Further effort is required.

Consider one stage in (5.9) as follows

$$\frac{U(z)}{V(z)} = 1 + \sum_{j=1}^{(q-1)N} d_j z^{-jp} \tag{5.10}$$

or in the time domain

$$u_n = v_n + \sum_{j=1}^{(q-1)N} d_j v_{n-jp} \tag{5.11}$$

where p and q are positive integers.

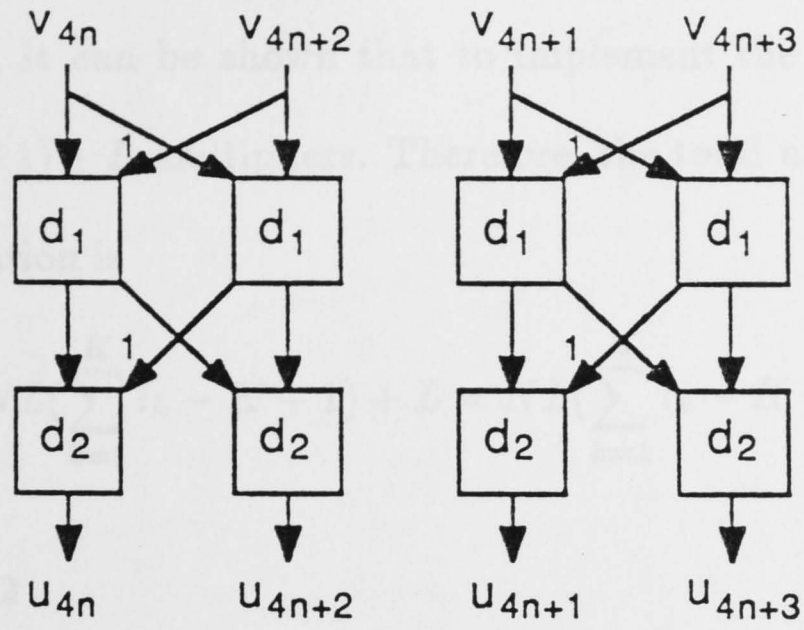


Fig. 5.5. The structure for computing (5.12)

$(N = 2, L = 4 \text{ and } p = q = 2)$

We first arrange the output u_n into p groups as

$$\text{for } 0 \le n_2 \le p \text{ and } n_1 = 0, 1, 2 \cdots,$$

$$\begin{aligned} u_{pn_1+n_2} &= v_{pn_1+n_2} + \sum_{j=1}^{(q-1)N} d_j v_{pn_1+n_2-jp} \\ &= v_{pn_1+n_2} + \sum_{j=1}^{(q-1)N} d_j v_{p(n_1-j)+n_2} \end{aligned} \tag{5.12}$$

From (5.12) we see that to compute $u_{pn_1+n_2}$ we need those inputs only in group $pn_1 + n_2$. Thus p groups of the output can be computed independently. We can also see that each sub-equation in (5.12) is just a linear convolution problem so that they can be computed by using the systolic ring structure described in Chapter 3. Therefore, the system for computing (5.11) or (5.12) may consist of p identical $(q-1)N \times L/p$ systolic ring structures. It is easy to determine that the number of multipliers required for computing (5.12) is

$$p(q-1)NL/p = NL(q-1) \tag{5.13}$$

An example with $N = 2$, $L = 4$ and $p = q = 2$ is depicted in Fig. 5.5.

We have shown that NL multipliers are needed for computing the recursive part. Using (5.13), it can be shown that to implement the linear part requires $NL(\sum_{k=1}^{K} l_k - K + 1) + L$ multipliers. Therefore, the total number of multipliers for this implementation is

$$NL + NL(\sum_{k=1}^{K} l_k - K + 1) + L = NL(\sum_{k=1}^{K} l_k - K + 2) + L \qquad (5.14)$$

5.3.2. Structure 2

The second structure is derived from the incremental output computation. Consider the unmodified algorithm of an N^{th} order recursive filter

$$y_n = \sum_{j=1}^{N} r_j y_{n-j} + v_n \qquad (5.15)$$

where $v_n = \sum_{j=0}^{N} w_j x_{n-j}$.

We arrange the output into L groups and assume that (1) $N < L$ and (2) the first N groups of the output are known. Then a regular and computable structure can easily be obtained. Fig. 5.6 gives an example with $N = 4$ and $L = 8$.

It is known from the previous subsection that each group of the output can be computed independently by using the modified algorithm. Therefore our modified algorithm is applied for computing the first N groups of the output, which is expressed as follows

$$\text{for } 0 \le l \le N-1$$

$$\begin{cases} y_{Ln+l} = \sum_{j=1}^{N} b_j y_{L(n-j)+l} + u_{Ln+l} \\ u_{Ln+l} = v_{Ln+l} + \sum_{j=1}^{(L-1)N} d_j v_{Ln+l-j} \end{cases} \qquad (5.16)$$

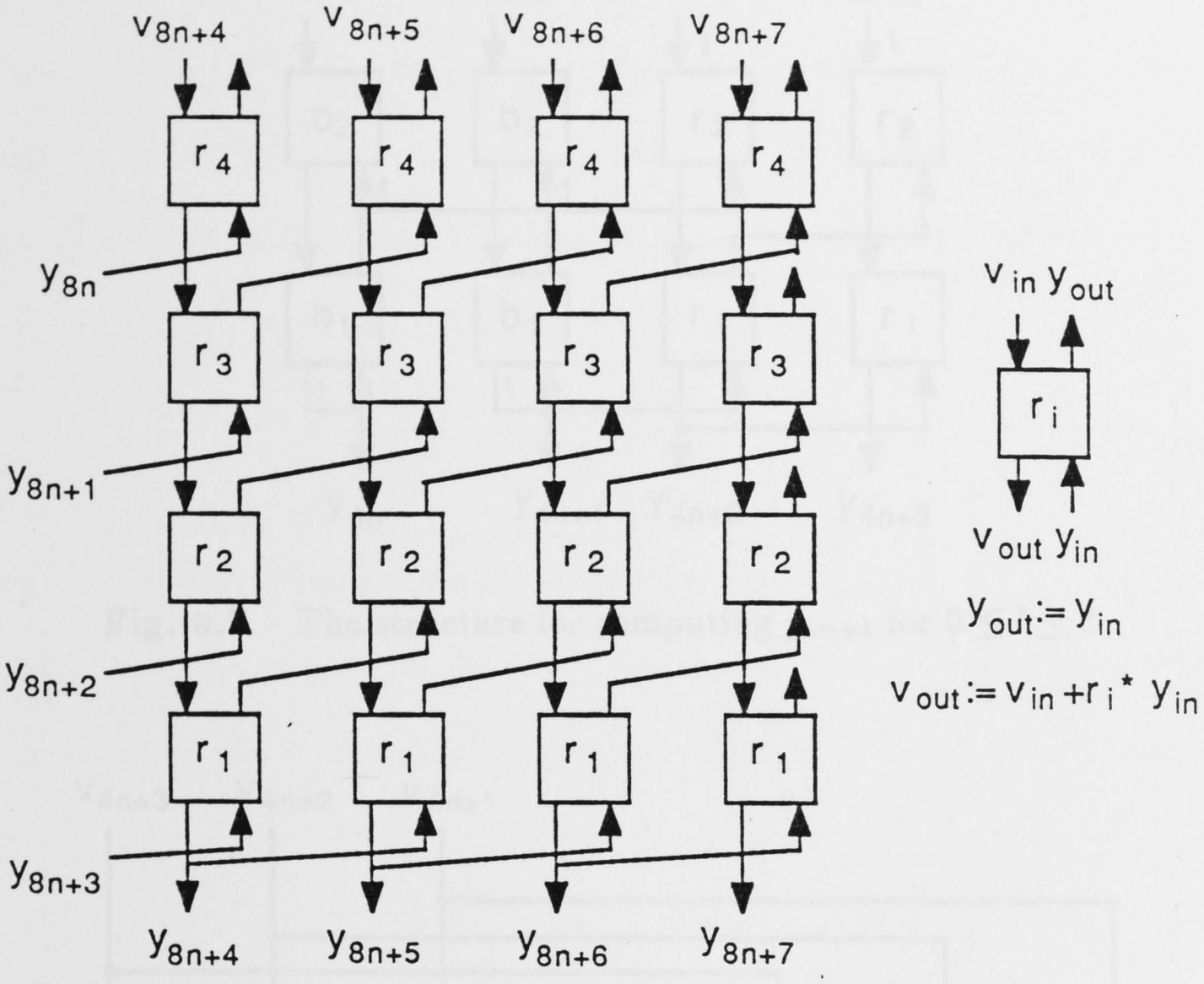


Fig. 5.6. The structure for computing y_{8n+l} for $4 \leq l \leq 7$

where $v_n = \sum_{j=0}^{N} w_j x_{n-j}$.

The structures for computing y_{Ln+l} for $0 \leq l \leq L-1$ and u_{Ln+l} for $0 \leq l \leq N-1$ are depicted in Figs. 5.7 and 5.8, respectively.

For computing v_n with a throughput rate of L we need $(N+1)L$ multipliers. There are NL multipliers for computing L groups of the output y_{Ln+l} for $l = 0$ to $L-1$. However, for the first N groups of the output we need to compute the intermediate result u_{Ln+l} for $l = 0$ to $N-1$, which requires $(L-1)N^2$ multipliers. Therefore, the total number of multipliers for this implementation is

$$(N+1)L + NL + (L-1)N^2 = (2N+1)L + (L-1)N^2 \qquad (5.17)$$

There are several disadvantages in this direct implementation of N^{th} order

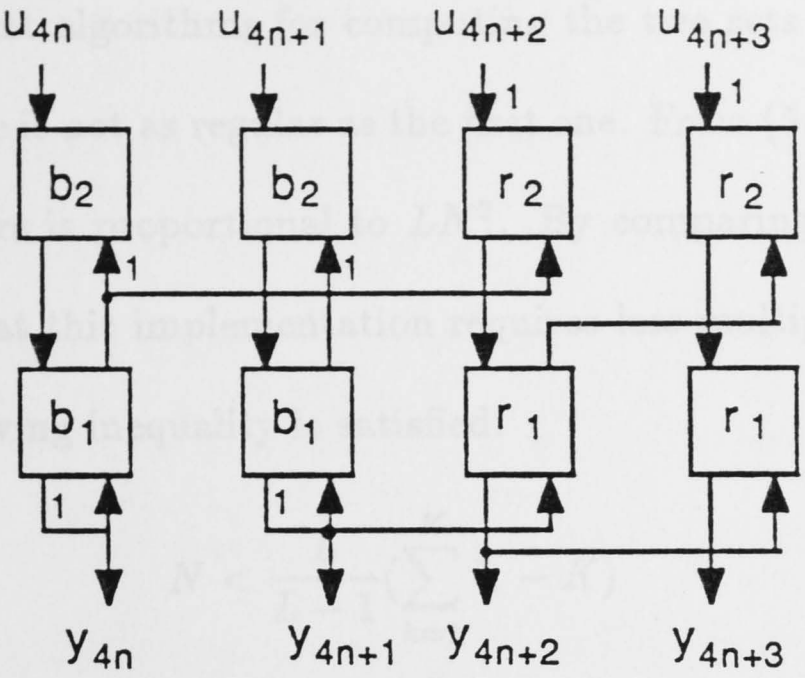


Fig. 5.7. The structure for computing y_{4n+l} for $0 \leq l \leq 3$

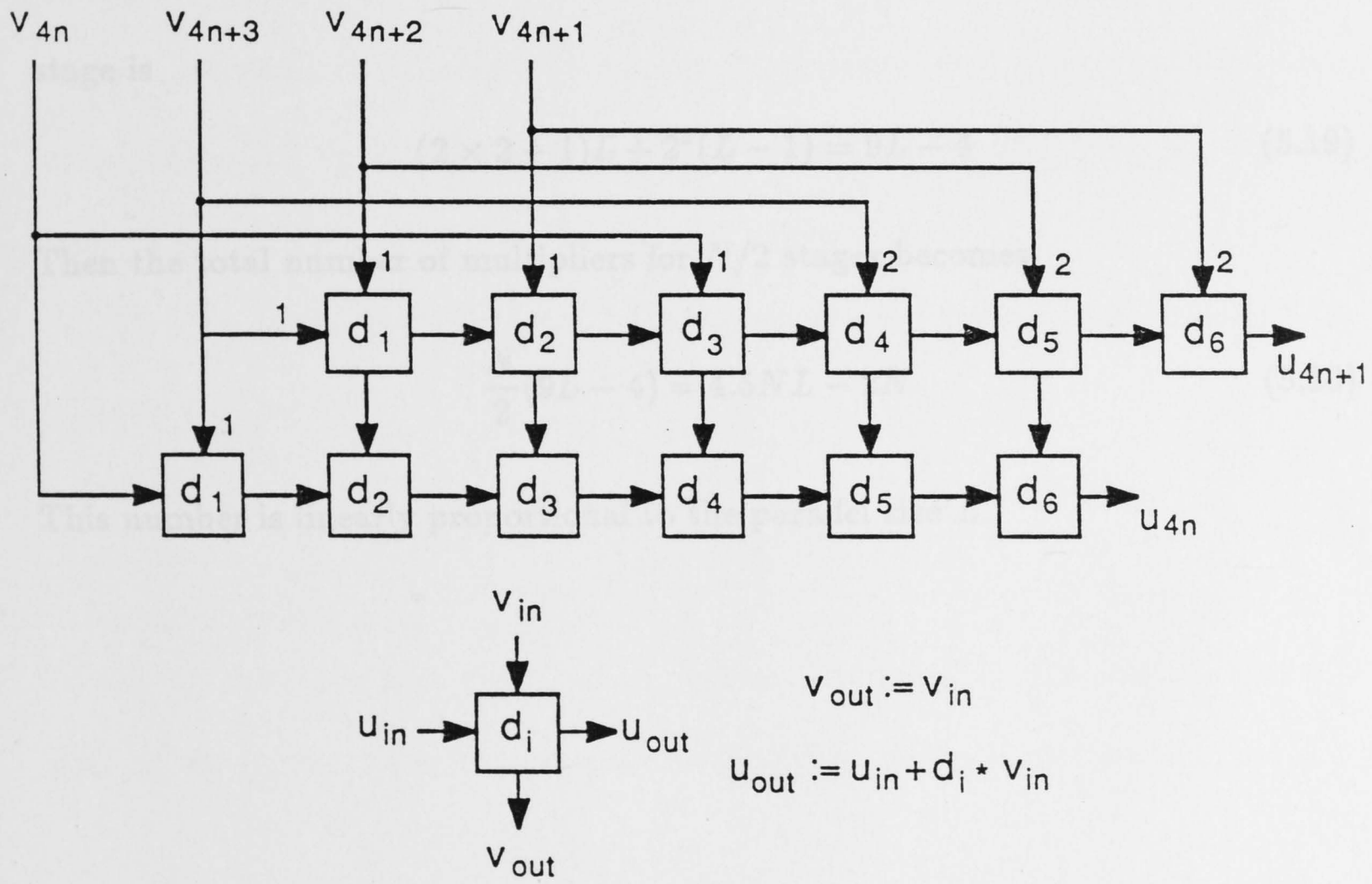


Fig. 5.8. The structure for computing u_{4n+l} for $0 \leq l \leq 1$

recursive filters. Under the assumption we made previously, the incremental output computation technique can only be applied in the case when $N < L$. Since we

must use two different algorithms for computing the two sets of L groups of the output, the structure is not as regular as the first one. From (5.17) we see that the number of multipliers is proportional to LN^2. By comparing (5.14) and (5.17), it can be verified that this implementation requires less multipliers than the first one only if the following inequality is satisfied:

$$N < \frac{L}{L-1}(\sum_{k=1}^{K} l_k - K) \tag{5.18}$$

where $L = \prod_{k=1}^{K} l_k$. For example, if $L = 8$ then $N < 4$.

This implementation can, however, be applied very efficiently for cascaded second-order recursive filters. The number of multipliers for one second-order stage is

$$(2 \times 2 + 1)L + 2^2(L-1) = 9L - 4 \tag{5.19}$$

Then the total number of multipliers for $N/2$ stages becomes

$$\frac{N}{2}(9L - 4) = 4.5NL - 2N \tag{5.20}$$

This number is linearly proportional to the parallel size L.

5.4. Pipelined and Parallel Structures

In the previous two subsections, we have constructed one two-level pipelined structure and two parallel structures for N^{th} order recursive filters. Now we combine these two techniques together to construct efficient pipelined and parallel structures. To achieve the throughput of LM, we shall construct two systems by using a parallel size of L and a two-level pipeline with M stages. The first one can efficiently be applied in the case when $L \leq N$ and the second one is used when the parallel size L is greater than the filter order N.

5.4.1. Case 1: $L \leq N$

As described in Subsection 5.3.2, the incremental output computation technique cannot be applied when $L \leq N$. To construct this structure, we then use the method in Subsection 5.3.1 for parallel processing.

To achieve the throughput of LM, the algorithm (derived in Section 4.3) can be expressed as

$$H(z) = \frac{(\sum_{j=0}^{N} w_j z^{-j}) \det(\sum_{j=0}^{LM-1} \mathbf{B}^j z^{-j})}{\det(\mathbf{I} - \mathbf{B}^{LM} z^{-LM})} \tag{5.21}$$

Similar to the structures described in Section 5.2 and 5.3.1, this system also consists of two parts, the recursive part and the linear part. These are as follows:

Recursive part :

It is known that the algorithm in the time domain for the recursive part can be expressed as

$$y_n = \sum_{j=1}^{N} b_{jLM} y_{n-jLM} + u_n \tag{5.22}$$

Arranging the output y_n into L groups, we have

$$\text{for } 0 \leq l \leq L-1,$$

$$\begin{aligned} y_{Ln+l} &= \sum_{j=1}^{N} b_{jLM} y_{Ln+l-jLM} + u_{Ln+l} \\ &= \sum_{j=1}^{N} b_{jLM} y_{L(n-jM)+l} + u_{Ln+l} \end{aligned} \tag{5.23}$$

It can be seen that each group of outputs can be computed independently since y_{Ln+l} depends only upon $y_{L(n-jM)+l}$ for $j = 1$ to N. We can also see that each sub-equation in (5.23) is essentially identical to the equation in (5.1). Therefore, the structure for the recursive part may consist of L identical 1-D sub-structures, each of which is a two-level pipelined structure with M stages as depicted in Fig. 5.1.

Linear part :

The algorithm for the linear part is expressed as

$$H_1(z) = (\sum_{j=0}^{N} w_j z^{-j}) \det(\sum_{j=0}^{LM-1} \mathrm{B}^j z^{-j}) \tag{5.24}$$

From Lemma 4.3, we can rewrite (5.24) as

$$H_1(z) = (\sum_{j=0}^{N} w_j z^{-j}) \det(\sum_{j=0}^{L-1} \mathbf{B}^j z^{-j}) \det(\sum_{j=0}^{M-1} \mathbf{B}^{jL} z^{-jL}) \tag{5.25}$$

In order to apply the decomposition technique, we assume that L and M can be factored, that is, $L = \prod_{k=1}^{K_1} l_k$ and $M = \prod_{k=1}^{K_2} m_k$ for l_k and m_k positive integers (not necessarily prime numbers).

Since $L = \prod_{k=1}^{K_1} l_k$, $\det(\sum_{j=0}^{L-1} \mathbf{B}^j z^{-j})$ can decomposed as a product of K_1 small polynomials in z^{-1}. Then the method for implementing this factor is exactly the same as that described in Section 5.3.1. The term $\sum_{j=0}^{N} w_j z^{-j}$ can be implemented by using an $(N+1) \times L$ systolic ring structure. Thus for implementing the linear part, we need only to consider the factor $\det(\sum_{j=0}^{M-1} \mathbf{B}^{jL} z^{-jL})$.

Let $\mathbf{B}^L = \mathbf{C}$ and $z^L = \lambda$. Then

$$\det(\sum_{j=0}^{M-1} \mathbf{B}^{jL} z^{-jL}) = \det(\sum_{j=0}^{M-1} \mathbf{C}^j \lambda^{-j}) \tag{5.26}$$

Since $M = \prod_{k=1}^{K_2} m_k$, by using the decomposition technique we have

$$\det(\sum_{j=0}^{M-1} \mathbf{C}^j \lambda^{-j}) = \prod_{k=1}^{K_2} (1 + \sum_{j=1}^{(m_k-1)N} c_j^{(k)} \lambda^{-j \prod_{i=1}^{k-1} m_i}) \tag{5.27}$$

Replacing λ by z^L, then

$$\det(\sum_{j=0}^{M-1} \mathbf{B}^j z^{-jL}) = \prod_{k=1}^{K_2} (1 + \sum_{j=1}^{(m_k-1)N} c_j^{(k)} z^{-jL \prod_{i=1}^{k-1} m_i}) \tag{5.28}$$

Consider stage k in the above equation, that is,

$$\frac{U(z)}{V(z)} = 1 + \sum_{j=1}^{(m_k-1)N} c_j^{(k)} z^{-jLm(k)} \tag{5.29}$$

where $m(k) = \prod_{i=1}^{k-1} m_i$. Transforming (5.29) into the time domain, we have

$$u_n = v_n + \sum_{j=1}^{(m_k-1)N} c_j^{(k)} v_{n-jLm(k)} \tag{5.30}$$

Arranging u_n into L groups, we obtain

$$\begin{aligned} &\text{for } 0 \le l \le L-1, \\ u_{Ln+l} &= v_{Ln+l} + \sum_{j=1}^{(m_k-1)N} c_j^{(k)} v_{Ln+l-jLm(k)} \\ &= v_{Ln+l} + \sum_{j=1}^{(m_k-1)N} c_j^{(k)} v_{L(n-jm(k))+l} \end{aligned} \tag{5.31}$$

It is easy to see that the L sub-equations in (5.31) are essentially the same as that in (5.4). Therefore, they can be implemented independently in the 1-D structure depicted in Fig. 5.2.

Let $LM = \prod_{k=1}^{K} l'_k$ where $K = K_1 + K_2$. It is easy to determine that the number of multipliers required for computing the linear part is $NL(\sum_{k=1}^{K} l'_k - K + 1) + L$. To implement the recursive part requires NL multipliers. Therefore, the total number of multipliers for this implementation is

$$NL(\sum_{k=1}^{K} l'_k - K + 1) + L + NL = NL(\sum_{k=1}^{K} l'_k - K + 2) + L \tag{5.32}$$

Using parallel implementation alone to achieve the same throughput requires $NLM(\sum_{k=1}^{K} l'_k - K + 2) + LM$ multipliers. Thus the number of multipliers is reduced by a factor of M by using the pipelined and parallel implementation described above.

5.4.2. Case 2: $L > N$

When $L > N$, the incremental output computation technique can be applied to achieve a further reduction of complexity.

As in the construction of the parallel structure in Section 5.3.2, the output y_n is arranged into L groups, that is, y_{Ln+l} for $0 \leq l \leq L-1$. For computing the first N groups of the output, we use our modified algorithm in (5.22) or (5.23). Then the associated structure may consist of N identical 1-D sub-structures and each sub-structure is just a two-level pipelined structure with M stages depicted in Fig. 5.1. By using the original unmodified algorithm in (5.15), the next $L-N$ groups of output can be implemented in the structure in Fig. 5.6. Combining these two structures together, we obtain a structure for computing the L groups of the output y_{Ln+l} for $0 \leq l \leq L-1$. An example with $N = 2$ and $L = 4$ is depicted in Fig. 5.9. However, this structure is not a two-level pipelined structure with M stages. To solve this problem, we introduce a set of cuts to the system, as shown in Fig. 5.9. It can be seen that all the lines in the cuts are pointing in one direction. Therefore, the second-level pipeline with M stages in this structure is easily obtained by adding M delays to every line in the cuts.

For the first N groups of the output, we need to compute

$$\begin{aligned} \frac{U(z)}{V(z)} &= \det\Big(\sum_{j=0}^{LM-1} \mathbf{B}^j z^{-j}\Big) \\ &= \det\Big(\sum_{j=0}^{L-1} \mathbf{B}^j z^{-j}\Big)\det\Big(\sum_{j=0}^{M-1} \mathbf{B}^{jL} z^{-jL}\Big) \end{aligned} \tag{5.33}$$

The first term $\det(\sum_{j=0}^{L-1} \mathbf{B}^j z^{-j})$ can be implemented in the structure in Fig. 5.8. For implementing the second term, as described in the previous subsection a

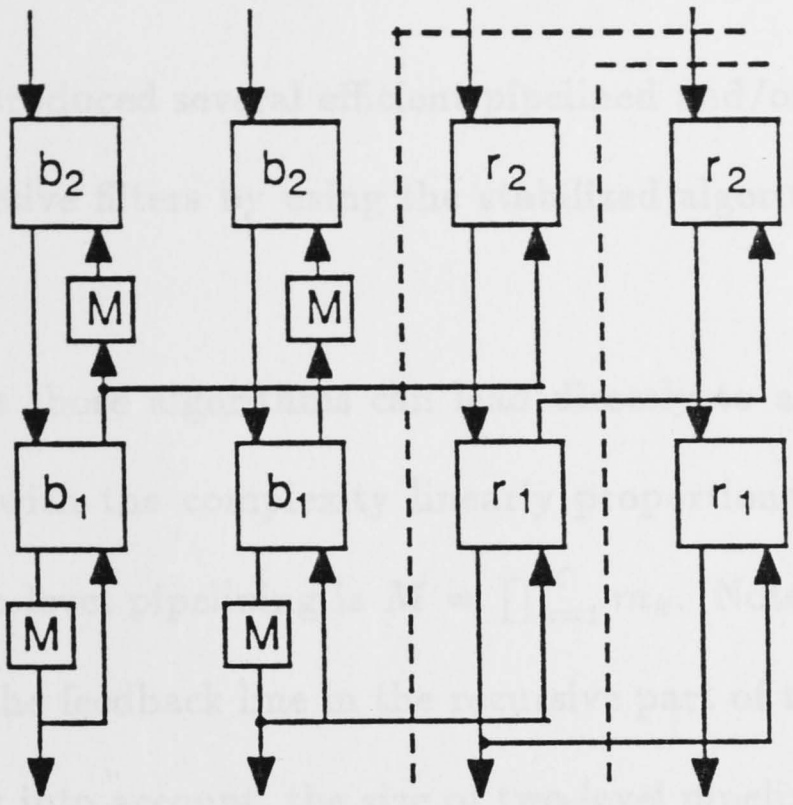


Fig. 5.9. The structure with M stages of second-level pipeline for computing y_{4n+l} for $0 \leq l \leq 3$

K_2 stage cascaded structure can be applied (when $M = \prod_{k=1}^{K_2} m_k$) and each stage consists of N independent 1-D sub-structures depicted in Fig. 5.2.

It is easy to determine that to implement (5.33) we need $N^2(L-1) + N^2(\sum_{k=1}^{K_2} m_k - K_2)$ multipliers. To implement $\sum_{j=0}^{N} w_j z^{-j}$ requires $(N+1)L$ multipliers, and NL multipliers are required for computing y_{Ln+l} for $0 \leq l \leq L-1$. Therefore, the total number of multipliers in this implementation is $N^2(\sum_{k=1}^{K_2} m_k + L - K_2 - 1) + L(2N+1)$. For cascaded second-order recursive filters, this number is reduced to $4.5NL + 2N(\sum_{k=1}^{K_2} m_k - K_2 - 1)$.

5.5. Discussion

This chapter introduced several efficient pipelined and/or parallel structures for direct-form recursive filters by using the stabilized algorithms derived in the previous chapter.

We showed that those algorithms can lead directly to an efficient two-level pipelined structure with the complexity linearly proportional to $\sum_{k=1}^{K} m_k - K$ when the size of two-level pipelining is $M = \prod_{k=1}^{K} m_k$. Note that for simplicity we did not localize the feedback line in the recursive part of the structure. When this matter is taking into account, the size of two-level pipelining will be reduced to $M - 1$. In Chapter 2 we mentioned that applying cut-set localization rules to localize a recursive system derived from the original unmodified algorithm will decrease the throughput because time-scaling cannot be avoided. Our modified algorithm can be applied to maintain the original throughput. To achieve this goal, $3N + 1$ multipliers are required. (This is easily verified by setting $M = 2$ in (5.6).) However, the system derived from the original unmodified algorithm only requires $2N + 1$ multipliers. Thus the penalty is an increase in complexity.

By using the method of parallel processing, we constructed two parallel architectures. Although the first parallel implementation requires more multipliers than the second one, it has a regular form. Thus it is more easily implemented in VLSI. Moreover, the first architecture only uses our modified algorithm. It is known that this algorithm is more stable than the original unmodified one. Therefore, in some applications the number of bits per word used in this implementation may be smaller than that used in the second one.

We also constructed two efficient pipelined and parallel structures for direct-

form recursive filters. With this combination of the methods for parallel processing and pipelining, the constructed architectures can achieve the same throughput with many fewer multipliers than pure parallel structures. This results in more-efficient VLSI implementation of recursive filters.

CHAPTER 6

CONCLUSIONS

Special-purpose systolic arrays are a most promising VLSI architecture for modern signal processing and form a useful basis for high throughput applications. Demands for ever-higher speeds in signal processing are seemingly insatiable. This thesis demonstrates that substantial increases in throughput can be achieved by developing parallel computing structures which take advantage of the high degree of parallelism inherent in typical signal processing algorithms. These structures can at the same time benefit from the power of VLSI and fit within its contraints.

Since the throughput of 1-D systolic arrays is limited by their word serial nature, in many high-throughput applications (two-level) pipelined and/or (2-D) parallel architectures have to be considered seriously. This thesis has investigated a number of pipelined and/or parallel systolic architectures for high-speed signal processing, particularly for high-throughput digital filters. The main conclusions which can be drawn from this study are summarized below.

The architecture transformation technique based on cut-set localization rules can convert all computable SFG networks into systolic arrays. The key problem in using this technique is to minimize the time-scaling factor. Using the commutative rule described in Chapter 2, together with the previously known rules, one can avoid time-scaling and obtain better results from a class of feedforward SFG computing networks.

A very important characteristic of non-recursive filters is that they can be designed to have exactly linear phase. 1-D systolic implementations of linear-phase

non-recursive filters can easily be obtained. Although the task of constructing 2-D systolic architectures is more difficult, we have demonstrated that this task can be simplified by synthesizing the 2-D structure from 1-D computational modules. Using twice the area in VLSI implementation, our 2-D systolic architecture can achieve twice the throughput of 1-D systolic arrays for a given problem. This results in no changes in the complexity measure AP.

Linear convolution is a very important computational problem in modern signal processing. The simple and regular pattern of the convolution equation makes it well suited for VLSI implementation. Suppose that N_1 is the number of coefficients and L is a positive integer smaller than the number of inputs. Two $N_1 \times L$ systolic structures have been introduced. These structures are more efficient than those described in the literature. They are based on nearest neighbour interconnections and can achieve L times the throughput of 1-D systolic arrays for the same problem. Moreover, the complexity measure AP is independent of L and the most efficient 1-D systolic array for solving linear convolution problems is just a special case of a 2-D systolic ring structure with $L = 1$. By varying L we can thus trade off area versus time for a given problem. Our 2-D systolic ring structures can also be applied to solve a number of other problems such as DFT, circular convolution and 2-D linear convolution because these problems can easily be transformed into 1-D linear convolution problems.

Although look-ahead computation is a good method to derive parallel algorithms for state-variable-form recursive filters, this conventional technique may cause numerical instability in direct-form recursive filters due to the effect of finite wordlength. Using the new method of Z domain derivation given in Chapter 4,

not only can parallel algorithms for direct-form recursive filters with guaranteed stability be derived, but the additional complexity required for this purpose can be minimized through a decomposition technique.

A time domain derivation of parallel algorithms for direct-form recursive filters has also been introduced. The derived algorithms, which have the same form as those derived in the Z domain, are of a particular interest for time-varying recursive systems. The condition for unique solution of the stabilized parallel algorithms has been derived as one tool for determining the stability of the derived algorithm.

Using the stabilized parallel algorithms for N^{th} order recursive filters, very efficient pipelined and/or parallel architectures can be constructed. An efficient 1-D systolic structure with two-level pipelining is directly obtained from those algorithms. Two different parallel systolic structures have also been derived based on those algorithms. The first one has the advantage of regularity while the second one can achieve a linear complexity in parallel size for cascaded second-order recursive filters. Using 2-D parallel processing in combination with two-level pipelining, pipelined and parallel architectures can also be constructed. With the same degree of complexity, these architectures can achieve much greater throughput than pure parallel structures.

REFERENCES

[1] Bandyopadhyay, S., Jullien, G. and Bayoumi, M., Systolic arrays over finite rings with applications to digital signal processing, in *Systolic Arrays*, Moore, W., McCabe, A. and Urquhart, R., (eds.) Adam Hilger, Bristol and Boston, 1986, pp. 123-132.

[2] Barnes, C.W. and Shinnaka, S., Block shift invariance and block implementation of discrete-time filters, *IEEE Trans. Circuits and Systems*, Vol. CAS-27, Aug. 1980, pp. 667-672.

[3] Barnes, C.W. and Shinnaka, S., Finite word effects in block-state realizations of fixed-point digital filters, *IEEE Trans. Circuits and Systems*, CAS-27, May 1980, pp. 345-349.

[4] Brent, R.P. and Luk, F.T., A systolic array for the linear-time solution of Toeplitz systems of equations, *Journal of VLSI and Computer Systems*, Vol. 1, No. 1, 1983, pp. 1-22.

[5] Brent, R.P. and Luk, F.T., The solution of singular-value and symmetric eigenvalue problems on multiprocessor arrays, *SIAM Journal of Sci. Stat. Comput.*, Vol. 6, No. 1, 1985, pp. 69-84.

[6] Burrus, C.S., Block implementation of digital filters, *IEEE Trans. Circuit Theory*, Vol. CT-18, Nov. 1971, pp. 697-701.

[7] Burrus, C.S., Block realization of digital filters, *IEEE Trans. Audio Electroacoust.*, Vol. AU-20, Oct. 1972, pp. 230-235.

[8] Cappello, P.R. and Steiglitz, K., Unifying VLSI array design with linear transformations of space-time, *Advances in Computing Research*, Vol. 2,

1984, pp. 23–65.

[9] Chern, M.Y. and Murata, T., Efficient matrix multiplications on a concurrent data-loading array processor, *Proc. 1983 Int. Conf. on Parallel Processing*, pp. 90–94.

[10] Drake, B.L., Luk, F.T., Speiser, J.M. and Symanski, J.J., SLAPP: A systolic linear algebra parallel processor, *IEEE Comput. Mag.*, Vol. 21, July 1987, pp.45-49.

[11] Ersoy, O., Semisystolic array implementation of circular, skew circular, and linear convolutions, *IEEE Trans. Computers*, Vol. C-34, No. 2, Feb. 1985, pp. 190-196.

[12] Evans, R.A., Wood, D., Wood, K., McCanny, J.V., McWhirter, J.G. and McCabe, A.P.H., A CMOS implementation of a systolic multi-bit convolver chip, in *VLSI '83*, Anceau, F. and Aas, E.J., (eds.) North-Holland, Aug. 1983, pp. 227-235.

[13] Fu, K.S., Hwang, K. and Wah, B.W., VLSI architectures for pattern analysis and image database management, in *VLSI and Modern Signal Processing*, Kung, S.Y., Whitehouse, H.J. and Kailath, T., (eds.) Prentice-Hall, 1985, pp. 434-450.

[14] Gentleman, W.H. and Kung, H.T., Matrix triangularization by systolic arrays, *Proc. SPIE*, Vol. 298, 1981, pp. 19-26.

[15] Gnanasekaran, R. and Mitra, S.K., A note on block implementation of IIR digital filters, *Proc. IEEE*, Vol. 65, July 1977, pp. 1063-1064.

[16] Gold, B. and Jordan, K.L., A note on digital filter synthesis, *Proc. IEEE*, Vol. 56, Oct. 1968, pp. 1717–1718.

[17] Kung, H.T., Special-purpose devices for signal and image processing: an opportunity in VLSI, Tech. Report, Dept. of Comput. Science, Carnegie-Mellon Univ., Pittsburgh, Pa., 1980.

[18] Kung, H.T., Why systolic architectures? *IEEE Comput. Mag.*, Vol. 15, No. 1, Jan. 1982, pp. 32-63.

[19] Kung, H.T. and Lam, M.S., Wafer-scale integration and two-level pipelined implementations of systolic arrays, *Journal of Parallel and Distributed Computing*, Vol. 1, 1984, pp. 32–63.

[20] Kung, H.T. and Leiserson, C.E., Systolic arrays (for VLSI), *Sparse Matrix Proc. 1978*, Academic Press, Orlando, Fla., 1979, pp. 256-282.

[21] Kung, H.T., Ruane, L.M. and Yen, D.W.L., A two-level pipelined systolic array for convolutions, in *VLSI Systems and Computations*, Kung, H.T., Sproull, R.F. and Steele. Jr., G.L., (eds.) Computer Science Press, Oct. 1981, pp. 273-278.

[22] Kung, H.T. and Song, S.W., A systolic $2D$ convolution chip, Tech. Report, Dept. of Comput. Science, Carnegie-Mellon Univ., Pittsburgh, Pa., 1981.

[23] Kung, S.Y., From transversal filter to VLSI wavefront array, *Proc. Int. Conf. on VLSI 1983*, IFIP, North-Holland, 1983, pp.247-261.

[24] Kung, S.Y., On supercomputing with systolic/wavefront array processors, *Proc. IEEE*, Vol. 72, No. 7, July 1984, pp. 867-884.

[25] Kung, S.Y., VLSI array processors, in *Systolic Arrays*, Moore, W., McCabe, A. and Urquhart, R., (eds.) Adam Hilger, Bristol and Boston, 1986, pp. 7-24.

[26] Kung, S.Y., Lo, S.C. and Annevelink, J., Temporal localization and systolization of signal flow graph (SFG) computing networks, *Proc. SPIE*, Real Time Signal Processing VII, SPIE, 1984, pp. 58–66.

[27] Leiserson, C.E., Systolic and semisystolic design, *Proc. IEEE Int. Conf. on Computer Design: VLSI in Computers*, NY, Nov. 1983, pp. 627–632.

[28] Leiserson, C.E. and Saxe, J.B., Optimizing synchronous systems, *Twenty-Second Annual Symposium on Foundations of Computer Science IEEE*, Oct. 1981, pp. 23–26.

[29] Li, G.J. and Wah, B.W., The design of optimal systolic arrays, *IEEE Trans. on Comput.*, Vol. C–34, No. 1, Jan. 1985, pp. 66–77.

[30] Lu, H.H., Lee, E.A. and Messerschmitt, D., Fast recursive filtering with multiple slow processing elements, *IEEE Trans. Circuits and Systems*, Vol. CAS–32, No. 11, Nov. 1985, pp. 1119–1129.

[31] Mead, C. and Conway, L., Introduction to VLSI Systems, Addison–Wesley, Reading, Mass., 1980.

[32] Meyer, R.A. and Burrus, C.S., A unified analysis of multirate and periodically time-varying digital filters, *IEEE Trans. Circuits and Systems*, Vol. CAS-22, Mar. 1975, pp. 162-168.

[33] Meyer, R.A. and Burrus, C.S., Design and implementation of multirate digital filters, *IEEE Trans. Acoust., Speech, Signal Process.*, Vol. ASSP-24, Feb. 1976, pp. 55-58.

[34] Mitra, S.K. and Gnanasekaran, R., Block implementation of recursive digital filters – new structures and properties, *IEEE Trans. Circuits and Systems*,

Vol. CAS-25, April 1978, pp. 200-270.

[35] Nussbaumer, H.J., Fast Fourier transform and convolution algorithms, Springer-Verlag, New York, 1982.

[36] Nikias, C.L., Fast block data processing via new IIR digital filter structure, *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 32, No. 4, Aug. 1984.

[37] Nudd, G.R. and Nash, J.G., Application of concurrent VLSI systems to two-dimensional signal processing, in *VLSI and Modern Signal Processing*, Kung, S.Y., Whitehouse, H.J. and Kailath, T., (eds.) Prentice-Hall, 1985, pp. 307-325.

[38] Oppenheim, A.V. and Schafer, R.W., Digital Signal Processing, Prentice–Hall, Inc., Englewood Cliffs, N.J., 1975.

[39] Parhi, K.K. and Messerschmitt, D.G., A bit-parallel bit level recursive filter architecture, *Proc. of the IEEE Int. Conf. Computer Design*, NY, 1986.

[40] Parhi, K.K. and Messerschmitt, D.G., Block digital filtering via incremental block-state structure, *Proc. of the IEEE International Symposium on Circuits and Systems*, Philadelphia, May 1987.

[41] Parhi, K.K. and Messerschmitt, D.G., Concurrent cellular VLSI adaptive filter architectures, *IEEE Trans. Circuits and Systems*, vol. 10, Oct. 1987, pp. 1141-1151.

[42] Parhi, K. K. and Messerschmitt, D. G., Pipelined VLSI recursive filter architectures using scattered look-ahead and decomposition, *Proc. IEEE Int. Conf. Acoustics, Speech, Signal Process.*, NY, Apr. 1988.

[43] Robert, Y. and Tchuente, M., An efficient systolic array for the $1D$ recursive convolution problem, *Journal of VLSI and Computer Systems*, Vol. 1, No.

4, pp. 398–407.

[44] Swartzlander, E., Systolic FFT processors, in *Systolic Arrays*, Moore, W., McCabe, A. and Urquhart, R., (eds.) Adam Hilger, Bristol and Boston, 1986, pp. 133-140.

[45] Travassos, R.H., Real-time implementation of systolic Kalman filters, *Proc. SPIE*, Vol. 431, Aug. 1983.

[46] Ullman, J.D., Computational Aspects of VLSI, Computer Science Press, Rockville, Maryland 1984.

[47] Voelcker, H.B. and Hartquist, E.E., Digital filtering via block recursion, *IEEE Trans. Audio Electroacoust.*, Vol. AU-18, Oct. 1972, pp. 169-176.

[48] Wambergue, C.A. and Roberts, R.A., Block processing structures for fixed point digital filters, *Proc. of the IEEE Int. Conf. ASSP*, Paris, May 1982, pp. 498-501.

[49] Weiser, U. and Davis, A., A wavefront notation tool for VLSI array design, in Kung, H.T., Sproull, B. and Steele, G., (eds) *VLSI Systems and Computations*, Computer Science Press, Rockville MD, 1981.

[50] Whitehouse, H.J., Speiser, J.M. and Bromley, K., Signal processing applications of concurrent array processor technology, in *VLSI and Modern Signal Processing*, Kung, S.Y., Whitehouse, H.J. and Kailath, T., (eds.) Prentice-Hall, 1985, pp. 25-41.

[51] Wu, C. W. and Cappello, R., Application-specific CAD of high throughput IIR filters, *Proc. Thirty-Second IEEE Computer Society International Conference*, Feb. 1987, pp. 302-305.

[52] Yen, D.W.L. and Kulkarni, A.V., Systolic processing and an implementation

for signal and image processing, *IEEE Trans. Computers*, Vol. C-31, No. 10, Oct. 1982, pp. 1000-1009.

[53] Zeman, J. and Lindgren, A.G., Fast digital filters with low roundoff noise, *IEEE Trans. Circuits and Systems*, Vol. CAS-28, July 1981, pp. 716-723.

[54] Zhou, B. B. and Brent, R. P., An efficient architecture for solving the recursive convolution equation with high throughput, *Proc. 1st IASTED International Symposium on Signal Processing and Its Applications*, Aug. 1987, pp. 771-775.

[55] Zhou, B. B. and Brent, R. P., A high throughput systolic implementation of the second order recursive filter, *Proc. IEEE Int. Conf. Acoustics, Speech, Signal Process.*, Apr. 1988.

[56] Zhou, B.B. and Brent, R.P., A stabilized parallel implementation of direct-form recursive filters, Tech. Report TR-CS-88-07, Comput. Sci. Lab., Australian National Uni., 1988, submitted to *Journal of Parallel and Distributed Computing.*

[57] Zhou, B.B. and Brent, R.P., A two-level pipelined implementation of direct-form recursive filters, Tech. Report TR-CS-88-06, Comput. Sci. Lab., Australian National Uni., 1988, submitted to *IEEE Trans. Acoust., Speech, Signal Processing.*