AN ABSTRACT OF THE THESIS OF

Aihua Li for the degree of Master of Science in Electrical and Computer Engineering presented on March 9, 1990.

Title: Synthesis of Multi-Rate Arrays from Directional Uniform Recurrence Equations

Abstract approved: Redacted for privacy
Sayfe Kiaei

Advances in VLSI array processing have led to many new parallel structures for real-time Digital Signal Processing (DSP) applications. Among all the architectures, systolic arrays have played an important role because systolic arrays have regular, local interconnections with modular structure. In ordinary systolic arrays, however, all processing operations and data transmissions use the same time clock, which degenerates the speed performance when different processing operations take different time to execute. Moreover, the application scope of systolic arrays is restricted to Uniform Recurrence Equations (URE), which is not applicable to all DSP algorithms.

This thesis introduces a new type of array processor architecture, Multi-rate Arrays (MRA). MRAs enhance the computation speed by assigning different clocks to different processing operations. They also enlarge the application scope of systolic arrays because MRAs can be synthesized from Directional Uniform Recurrence Equations (DURE), which is a more general form than URE.

The thesis first introduces the idea of MRA, demonstrates its enhancement on speed performance over systolic arrays, and gives a criterion in choosing different array structures. It then relates MRAs with DURE by analyzing the characteristics of DURE and formulates a systematic procedure for the synthesis of MRA from DURE.At last, it demonstrates the synthesis of MRA by two examples in DSP applications---decimation filter and Toeplitz matrix factorization.

SYNTHESIS OF MULTI-RATE ARRAYS FROM DIRECTIONAL UNIFORM RECURRENCE EQUATIONS

by

Aihua Li

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Completed March 9, 1990

Commencement June 1990

APPROVED:

Redacted for privacy

Assistant Professor of Electrical and Computer Engineering
in Charge of Major

Redacted for privacy

Head of department of Electrical and Computer Engineering

Redacted for privacy

Dean of Graduate School

Date thesis is presented March 9, 1990

Typed by Aihua Li for Aihua Li

ACKNOWLEDGEMENT

I express my sincere appreciation to Prof. James H. Herzog, who introduced me to this school, gave strong support on my application and is continuously offering help to me.

I also want to thank my advisor, Dr. Sayfe Kiaei, for his valuable suggestions, criticism and patience during the research work. I have also felt help from him on my personal life, besides his academic advice.

My thanks are also extended to Prof. D.J. Allstot, Dr. P. Lenders, from whom I have learned a lot of new technology. I also want to thank them for willingness to serve on my committee.

Table of Contents

List of Figures

SYNTHESIS OF MULTI-RATE ARRAYS FROM DIRECTIONAL UNIFORM RECURRENCE EQUATIONS

CHAPTER 1
INTRODUCTION

The rapid advances in VLSI (Very Large Scale Integration) technology have made a major breakthrough in parallel computer architectures. Only with the innovations of these architectures may Digital Signal Processing (DSP) applications become imaginable since these applications require very high processing speed for real-time response. This chapter gives a brief survey on the related work in this field, the objective and an outline of this thesis.

1.1 Related Work

The parallel architecture for DSP applications has some distinct characteristics. Because of the real time processing speed of DSP applications, the architecture are generally application-oriented and have less complex hardware structure. Its Processing Elements (PEs) must be as modular and as regular as possible in order to meet the requirement of low cost and fast turn-around design for the large varieties of the applications. For an VLSI array processor design for DSP application, an application is first specified and formulated into a certain form of algorithm. Such an algorithm is then analyzed and mapped onto array structures according to some systematic mapping procedure. The obtained array structure is further realized using hardware and implemented onto the final chip. The general design procedure is depicted in Fig 1.

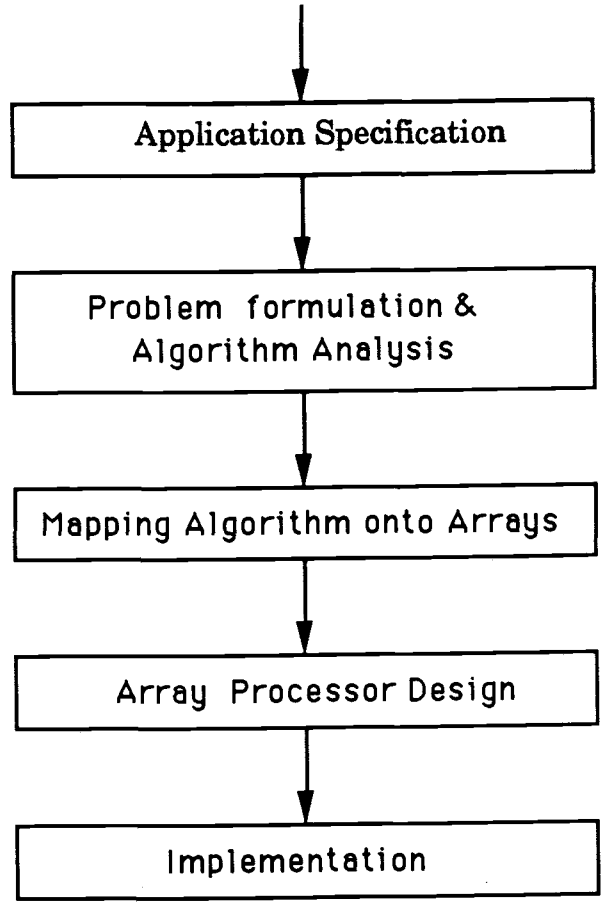


Fig 1. Array Processor Design Procedure

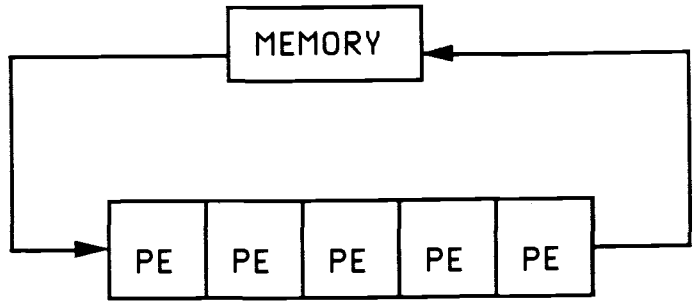


Fig 2. Structure of Systolic Arrays

The focus of this thesis is on the step of mapping algorithm onto arrays, which involves two aspects, a good algorithm form and a good array architecture. On the architecture aspect, among many new array processor architectures for signal processing, systolic arrays have played a significant role since they are modular, regular and have local inter-processor connections. The processing is generally pipelined and in synchronized multiprocessing fashion. According to H.T. Kung, "*A systolic system is a network of processors that rhythmically compute and pass data through the system.*"[1] Such property enables the designer to reduce the circuit complexity and eliminate the need of system management in each PE. Fig 2 depicts a typical systolic array structure.

However, in ordinary systolic arrays, all processing operations and data transmissions use the same time clock, which degenerates the speed performance when different processing operations takes different time to execute. For example, multiplication generally takes far longer time to execute than addition operation, but they are all finished in one global time unit, which makes the addition circuit stays in idle state most of the time. On the algorithm aspect, the requirement of low design cost and fast turn-around has inspired many efforts on giving an easy-to construct specification, unified initial algorithm and a systematic method to map algorithms onto arrays.

Earlier work on the synthesis method was based on using the Uniform Recurrence Equations (URE) as the initial algorithmic specification for the application problems and as the starting point for systematic methodology of mapping algorithm onto arrays. The advantage to use URE as the algorithm description is that URE can be easily mapped onto arrays with modular structure and regular and local interconnections. However, not all DSP application can be specified by URE. Examples are Toeplitz matrix decomposition, decimation filters. Even if a problem can be specified by URE, it is not good as the initial specifications [6], because using URE as the initial

specification would involve too much intelligence, thus a systematic procedure cannot be achieved. Examples are LU decomposition, longest common sub-string etc [6]. Later work done by Rajopadhy, Yaacoby [6,'40] was to use Affine Recurrence Equations (ARE) as the initial specifications. In ARE, data dependencies are of the form:

$$p \text{ ----> } \delta(p); \text{ where } \delta(p) = Dp+d$$

The ideas behind their work were similar: use ARE as initial specifications, try to convert it to an URE as the intermediate form, and once an intermediate URE is obtained, it can be mapped onto systolic array or other arrays accordingly. However, systematic synthesis from ARE are still far from being complete. Rajopadhy's work requires that the linear part D of the affine dependency be singular. Yaacoby's work requires that D and all combinations of these D be roots of I, i.e., D^L=DD...D=I, which is a very restrictive requirement.

1.2 Objectives

The objective in this thesis is to introduce a new type of array processor architecture, Multi-Rate Arrays (MRA), which overcomes the drawbacks and enlarges the application scope of systolic arrays. First, MRA enhance the processing speed by assigning different clocks to different processing operations. Second, on the algorithm aspect, it has been found that a more general class of ARE where D-I is singular has the property that they have uniform dependency in some directions and multi-rated dependency in other directions. We call this class of ARE as Directional Uniform Recurrence Equation (DURE).

It has also been found that DURE can be allocated to processor space in such a way that regular inter-processor connections can be maintained. Furthermore, the scheduling function for each variable is can be in multi rate fashion and thus can be mapped onto MRAs. The importance of these results is that a more general class of AREs can be

mapped onto MRAs suitable for VLSI implementation, just like all URE can be mapped onto systolic arrays. Note the major difference between MRA and systolic arrays is that MRA requires multiple clocks, which does not add any difficulty because low rate clocks can be easily obtained from high rate clocks.

1.3 Overview of the dissertation

This thesis is organized as follows. First, chapter 2 introduces the concept of MRAs and their siblings, Bounded Broadcast Arrays (BBAs). It also compares the speed performance and gives a criterion in choosing different architecture. Chapter 3 gives the characterizations and formalizes the synthesis method for DURE. Chapter 4 applies the so formalized procedure to the two well known problem, decimation filter and Toeplitz matrix decomposition, as two illustration examples.

Chapter 2

MULTI-RATE/BOUNDED BROADCAST ARRAY STRUCTURES

This chapter introduces the concepts of Multi-Rate Arrays (MRA) and Bounded Broadcast Arrays (BBA) and illustrates their speed performance enhancement over conventional array structures. It also provides a criterion in choosing among different array structure for a given algorithm.

The whole chapter is organized as follows. Section 2.1 introduces the concepts of MRA and BBA structure. Section 2.2 demonstrates the MRA/BBA design for convolution and compare the speed performance with conventional array structure by example of convolution. Section 2.3 discuss the criterion in choosing different array structures.

2.1 Concepts of MRA and BBA

Among all VLSI array structures, systolic arrays have played an important role because they have regular, local interconnections with modular structure. In conventional systolic arrays, however, all processing operations and data transmissions use the same time clock, which degenerates the speed performance.

For example, multiplication generally takes far longer time to execute than addition operation, but if they are all allocated the same amount of time, the addition circuit will stay idle most of the time.

Such a problem also exists between the main processing operation and data transmissions. Data transmissions usually are much faster than the main operations and always exists in array processors.

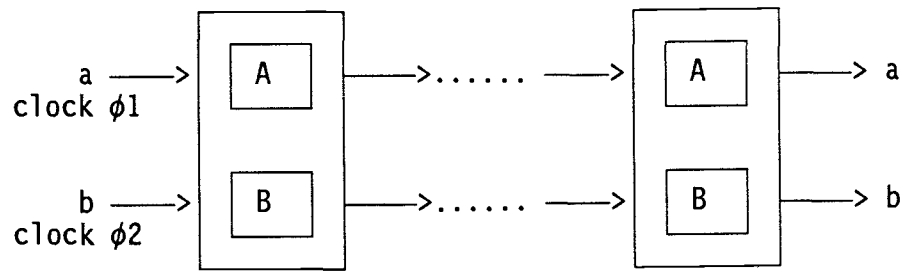
The latter problem can be avoided by permitting broadcast, which transmits data to all Processor Elements (PEs) simultaneously. However, in VLSI design, broadcast is not preferable because it takes too much

chip area. Moreover, it does not solve the former problem. We call arrays permitting broadcast as broadcast arrays. Such a dilemma can be solved by two approaches.

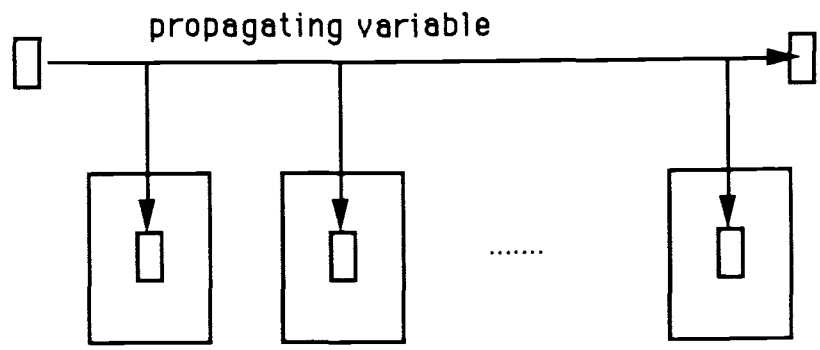
(1) Assigning different clocks to different processing operations. (Data transmission can be treated as special operation).

(2) Broadcast data to PEs only within limited distance.

The former approach leads to the Multi-Rate Arrays (MRA) as shown in Fig 3(a). The latter leads to Bounded Broadcast Arrays (BBA) as shown in Fig 3(b).



(a) MRA structure.



(b) BBA structure.

Fig 3. MRA and BBA structure.

It should be pointed out that BBA can be applied only to the data transmission, while MRA is not restricted so. In this sense, MRA is superior to BBA. In the following discussion about the design of MRA and BBA, the concentration is put on MRA. The results can be similarly applied to BBA with a few modifications.

It is known that in broadcast array, it is necessary and sufficient for the scheduling vector s and projection vector 1 to satisfy,

$$(1)\ s^T e \geq 0, \text{ for any e; (causality condition)} \tag{2.1}$$

$$(2)\ s^T \mu \neq 0 \quad \text{(conflict-free condition)} \tag{2.2}$$

where e denotes dependency arcs in the Data Dependency Graph (DG). In systolic arrays, since no broadcast is allowed, the above two constraints are changed to

$$(1)\ s^T e \geq 1, \text{ for any e; (causality condition)} \tag{2.3}$$

$$(2)\ s^T \mu \neq 0 \text{ (conflict-free condition)} \tag{2.4}$$

The vector s for both broadcast arrays and systolic arrays must have integer components.

In MRA, if we still use processing time of the slowest operation as the basic time unit, the fast processing operation takes only a fractional part of the basic unit. The components of the vectors s are not necessarily be integers. Instead, they can be fractions. Thus, the constraints are changed to

$$(1)\ s^T e \geq 1, \text{ if e originates from basic operation;} \tag{2.5}$$

$$(2)\ s^T e \geq \frac{1}{K}, \text{ if e originates from operations K times faster.} \tag{2.6}$$

$$(3)\ s^T \mu \neq 0 \tag{2.7}$$

To understand the constraints $s^T e \neq \frac{1}{K}$, consider e corresponding to data propagation. Equality $s^T e = \frac{1}{K}$ means that it takes $\frac{1}{K}$ of the basic time unit to transmit the data between two points in DG, which is permissible now.

Similar changes can be found for BBA. They are as follows,

(1) $s^T e \geq 1$, if e originates from basic operation (2.8)

(2) $s^T e \geq \frac{1}{K}$, if e corresponds to propagation. (2.9)

(3) $s^T \mu \neq 0$ (2.10)

It should be pointed out, however, the difference between MRA and BBA still exists. In BBA, all operation must start at an integer instance in the time space, while in MRA, operations can start at any fractional instance in time space. Specifically, if p is a point in DG, its computation starts at

$$T = s^T p \tag{2.11}$$

while in BBA, operation starts at

$$T = \text{ceil}[s^T p] \tag{2.12}$$

where ceil denotes the ceiling function.

2.2 Comparison between different array structure.

To compare the performance between MRA/BBA and the conventional array structures, convolution is used as an application example for illustration.

Generally, a VLSI array design for an application involves the following procedures.

Stage 1. Problem formulation.
Stage 2. DG design.
Stage 3. Array architecture design. This stage is to obtain
(a) Processor assignment.
(b) scheduling function.
(c) PE cell structure.

Generally, an optimal objective is given, and this stage is further devided into substeps as follows.

Step 3.1. obtain the optimality function.
Step 3.2. list the constraints.
Step 3.3. obtain solution to the optimality problems.

[Example 1] Convolution is defined as follows.

Given: weight sequence $W=\{w_1,w_2,.....w_k\}$,
input sequence $X=\{x_1,x_2,.....,x_{n+k-1}\}$,
Compute: output sequence $Y=\{y_1,y_2,.....,y_n\}$ defined by

$$y_i = w_1x_i + w_2x_{i+1} +..... + w_kx_{i+k+1} \qquad (2.13)$$

Generally, W is a finite sequence while X might be an infinite sequence.

2.2.1 Problem formulation and DG for convolution

An obvious recursive form for array variable y based on conventional programming technique is

$$y(i) = y(i) + w(j)x(i+j) \tag{2.14}$$

In array processor design, it is desired to convert such a form into a SAC form as,

$$y(i,j)=y(i,j-1)+w(j)x(i+j) \tag{2.15}$$

The result for y(i) is then stored in y(i,n). Further localizing the broadcasting variable w and x, we get the following recurrence form,

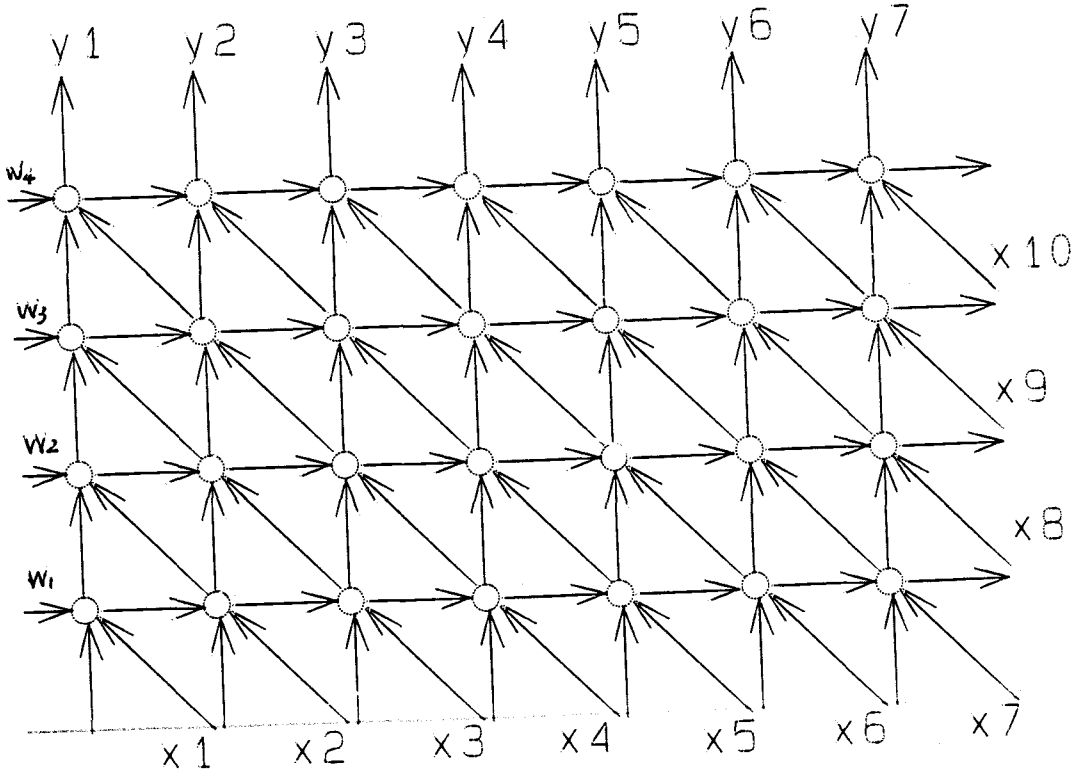
$$\begin{aligned} &y(i,j)=y(i,j-1)+w(i,j)x(i,j);\\ &w(i,j)=w(i-1,j);\\ &x(i,j)=x(i+1,j-1);\\ &y(i,0)=0;\quad w(0,j)=w_j,\quad x(i,0)=x_i; \end{aligned} \tag{2.16}$$

Its correponding DG is shown in Fig 4.

To compare performance between various array structure, we set speed as the optimal objective. For the DG in Fig 4, assuming scheduling vector $s=[s_1,s_2]$, it can be seen that the computation time is

$$\begin{aligned} T &= [s_1,s_2]\begin{bmatrix} n \\ k \end{bmatrix} - [s_1,s_2]\begin{bmatrix} 1 \\ 1 \end{bmatrix} + 1 \\ &= s_1(n-1)+s_2(k-1)+1 \end{aligned} \tag{2.17}$$

Fig 4. DG for convolution.



2.2.2 Broadcast Array and Systolic Array Solution

BROADCAST ARRAY

Step 3.1. Obtain optimality function. From (2.17),

$$T=s_1(n-1)+s_2(k-1)+1 \qquad (2.18)$$

Step 3.2 list the constraints. Notice e_x, e_w corresponds to data propagation and e_y corresponds to computed variable y. We have the following,

$$s^T e_x=[s_1,s_2]\begin{bmatrix}1\\1\end{bmatrix}=s_1+s_2\geq 0$$

$$s^T e_w=[s_1,s_2]\begin{bmatrix}1\\0\end{bmatrix}=s_1\geq 0$$

$$s^T e_y=[s_1,s_2]\begin{bmatrix}0\\1\end{bmatrix}=s_2 1$$

$$s^T \mu \neq 0$$

s_1,s_2 integer, $[s_1,s_2]\neq[0,0]$

But notice, since the optimality expression does not contain any term related to l, the choice of l does not affect total computation T. Thus, one can solve s first without considering l. Such a l can be obtained afterwards.

Step 3.3 Solve the optimality problem. Combining optimality function (2.18) and constraints (2.19), the broadcast array design is now equivalent to the integer programming problem as

Minimize

$$T=s_1(n-1)+s_2(k-1)+1$$

Under the constraint

$$s^T e_x=s_1+s_2 \geq 0$$
$$s^T e_w=s_1 \geq 0 \quad (2.20)$$
$$s^T e_y=s_2 \geq 1$$
$$s_1,s_2 \text{ integer}, [s_1,s_2] \neq [0,0]$$

Generally, it is assumed the input sequence X is for longer than the weight sequence W, i.e, n>>k. An obvious solution to the above problem is

$$s=[s_1,s_2]^T=[0,1]^T$$

If we choose $\mu=[0,1]^T$, $s^T\mu=1>0$, the resulted array is shown in Fig 5. The minimum $T_{min}=s_2(k-1)+1=k$. i.e, k steps are needed.

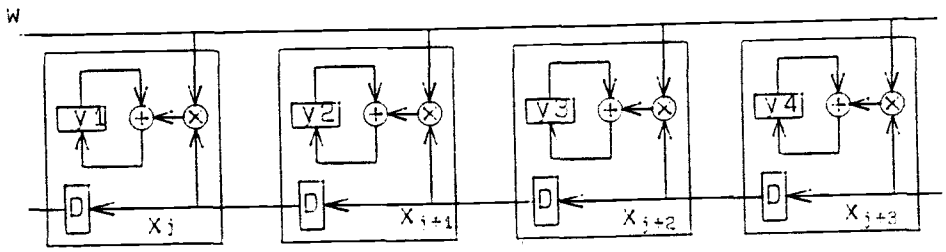


Fig 5. Broadcast array for convolution.

SYSTOLIC ARRAY

The systolic array design for convolution can be similarly converted to the following optimal problem.

Minimize

$$T=s_1(n-1)+s_2(k-1)+1$$

Under the constraint

$$\begin{aligned} s^T e_x &= s_1+s_2 \geq 1 \\ s^T e_w &= s_1 \geq 1 \\ s^T e_y &= s_2 \geq 1 \\ s_1, s_2 &\text{ integer, } [s_1,s_2] \neq [0,0] \end{aligned} \tag{2.21}$$

The solution can be obtained as

$$s=[s_1,s_2]^T=[1,2]^T$$

$$T_{min}=n-1+2(k-1)+1=n+2k-2$$

Choosing $\mu=[1,0]^T$, an array structure as shown in Fig 6. results.

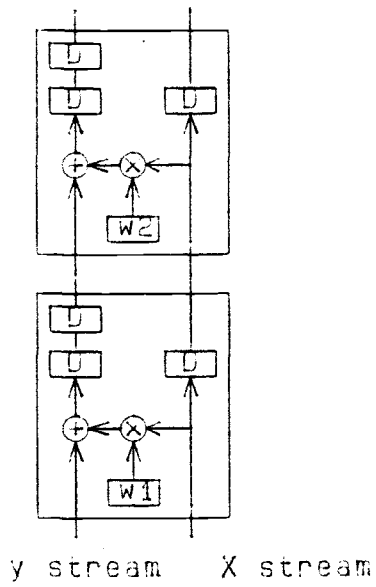


Fig 6. Systolic Array for convolution.

2.2.3 MRA/BBA solutions for convolution.

step 3.1 Optimality function. Still from (2.17),

$$T=s_1 1(n-1)+s_2(k-1)+1 \tag{2.22}$$

step 3.2 List constraints. Remember that the constraints for propagation arcs e_x and e_w is modified to $s^T e \geq \frac{1}{K}$, they are as follows,

$$\begin{aligned} &s^T e_x=-s_1+s_2 \geq \frac{1}{K}, \\ &s^T e_w=s_1 \geq \frac{1}{K}, \\ &s^T e_y=s_2 \geq 1 \\ &s^T \mu > 0 \\ &s_1,s_2 \text{ integer multiples of } \frac{1}{K}, [s_1,s_2] \neq [0,0] \end{aligned} \tag{2.23}$$

step 3.3 Solve the optimality problem. In the same way, the solution to the optimality function (2.22) under the constraints (2.23) can be found to be

$$s=[s_1,s_2]^T=[\frac{1}{K},1]^T \tag{2.24}$$

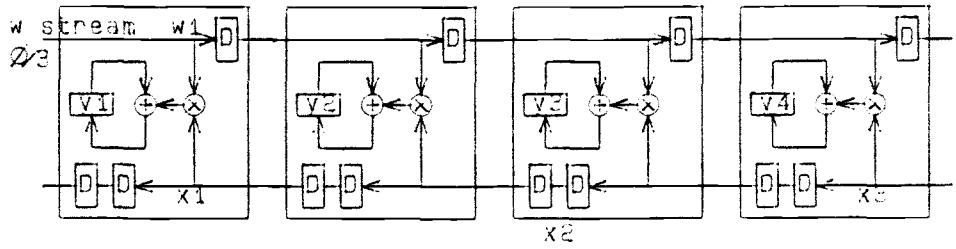
For MRA,

$$T_{min}=\frac{n-1}{K}+k; \tag{2.25}$$

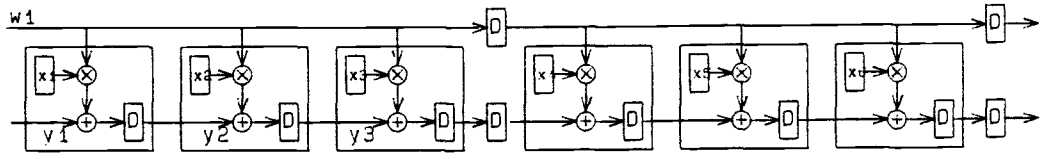
For BBA,

$$\begin{aligned} T_{min} &= \text{ceil}[s^T q] - \text{ceil}[s^T p] + 1 \\ &= \text{ceil}[\frac{n}{K}+k] - \text{ceil}[\frac{1}{K},+1] + 1 \\ &= k-1 + \text{ceil}[\frac{n}{K}] \end{aligned} \tag{2.26}$$

Choosing μ=[0,1], the resulted MRA and BBA are illustrated in Fig 7(a) and Fig 7(b) correspondingly.



(a) MRA for convolution.



(b) BBA for convolution.

Fig 7. MRA/BBA for convolution.

2.2.4 Performance Comparison.

The speed performance of the various array structure is given below. It is seen that MRA/BBA generally achieves a speed K times as

fast as the systolic array, where K is the speed ratio between the fastest processing operation and the slowest operation. Broadcast array is even faster, but they occupy too much chip area in implementation, which is not desirable.

It is also seen that when K=1 MRA/BBA reduces to systolic array and when K approaches to infinity, MRA has the same structure as broadcast arrays. In this sense, systolic arrays and broadcast arrays are nothing but subsets of MRA/BBA family.

Table 1. Speed Performance Comparison

	Computation Time;	Computation Time when n>>k
Systolic:	T=n+2k-2	n
MRA:	$T = \frac{n-1}{K} + k$	$\frac{n}{K}$
BBA:	$T=\text{ceil}[\frac{n}{K}]+k$	$\frac{n}{K}$
Broadcast:	T=k	0

2.3 Criterion in choosing the array structure.

It is not always more efficient to use broadcast or MRA/BBA than systolic array. Whether to use MRA/BBA or general systolic array depends on the design objective. Sometimes, using MRA/BBA or broadcast array does not necessarily give better performance. In the following, a criterion is given to determine whether using MRA/BBA or broadcast would give better performance.

Suppose a DG is given and the set of dependency arcs is

$$e_1, e_2, \ldots\ldots, e_m$$

From previous discussion, we know the set of constraints s must satisfy is

$$s^T e_1 \geq b_1$$
$$s^T e_2 \geq b_2$$
$$\ldots\ldots$$
$$s^T e_m \geq b_m$$

or

$$\begin{bmatrix} e_1^T \\ e_2^T \\ . \\ . \\ e_m^T \end{bmatrix} .s \geq \begin{bmatrix} b_1 \\ b_2 \\ . \\ . \\ b_m \end{bmatrix} \qquad (2.28)$$

where b_i=1 or $\frac{1}{K}$, i=1, ..., m, depending on whether e_i corresponds to propagating or computed variables. Equation (2.28) defines a convex cone in the n-space, as shown in Fig 8. In other words, the feasible values the components of s can assume are within the convex cone. Suppose the objective is to minimize

$$g=f(s)=s^T b$$

From the knowledge of integer programming, we know an objective function defined over a convex figure assumes its optimal value at one of the vertices. The optimal value can be obtained by moving the hyperplane $s^T b=C$ until it reaches to a vertex. Observing the neighboring hyperplanes defined by $s^T e_i=b_i$ and their corresponding b_i would tell whether the use of broadcast would improve the objective. We still use the convolution problem to make this idea clear.

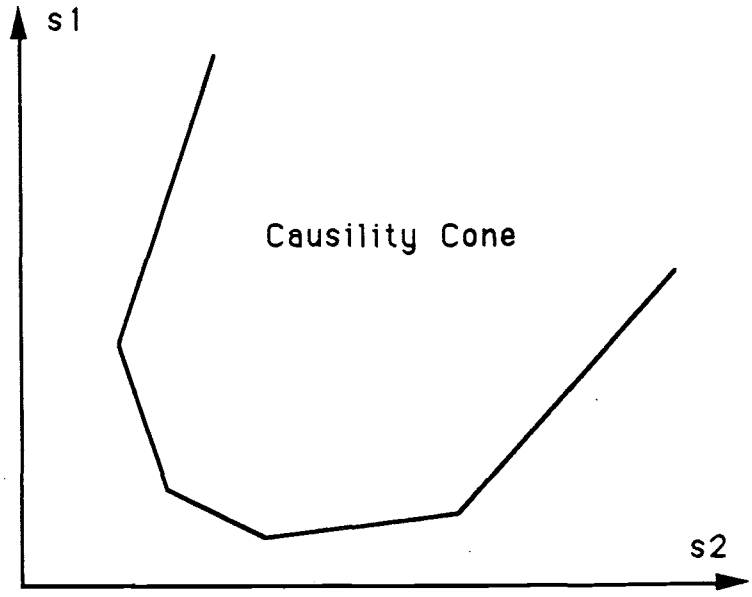


Fig 8. Causality Convex Cones.

For the DG of convolution in Fig 4, the constraints are

$$\begin{aligned} s^T e_x &= -s_1 + s_2 \geq \frac{1}{K}, \\ s^T e_w &= s_1 \geq \frac{1}{K}, \\ s^T e_y &= s_2 \geq 1 \end{aligned} \tag{2.29}$$

The convex cone defined by these constraints is shown in Fig 9. The objective function is

$$T_1 = (n-1)s_1 + (K-1)s_2.$$

Let $(n-1)s_1+(K-1)s_2=C$ and move the line until it reaches the vertex, which results in the state as shown. Obviously, if K increase, the objective functions will decrease and will result better performance. Thus, using MRA/BBA would improve the computation time performance.

On the contrary, if the objective function is to minimize pipeline period defined as

$$\alpha=s^T\mu$$

and m is chosen $\mu=[1,0]^T$, then

$$\alpha=s^T\mu=s_1$$

Letting $\alpha=s_1=C$ and move it to pass $[\frac{1}{K},1]^T$ we got

$$\alpha=s_1=1$$

Obviously, the change of K has no any effect on the objective function α Thus, using broadcast does not improve the performance of pipelining rate.

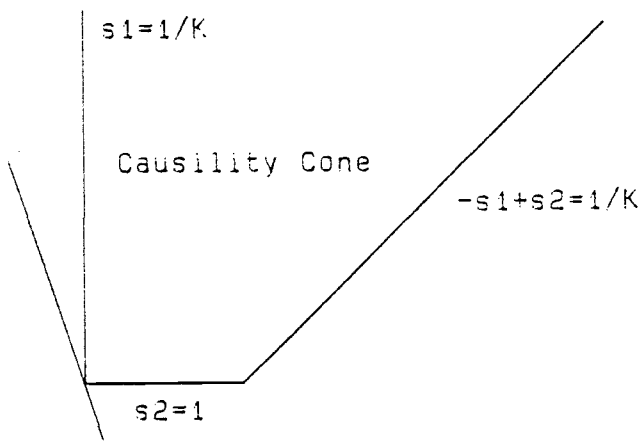


Fig 9. Causality cone for convolution.

Chapter 3

SYNTHESIS OF DURE ON MRA

In a systematic synthesis, a formal procedure must be give for automatically mapping a certain form of algorithm onto certain type of array structure. This chapter is devoted to the systematic synthesis of a class of ARE in which D-I is singular. It will be shown that this class of ARE has uniform dependencies along some directions, thus it is termed as Directional Uniform Recurrence Equations (DURE). It will be shown that the processor allocation function and scheduling functions for DURE are well defined and can be well mapped onto MRA.

This chapter is organized as follows. Section 3.1 deals with the classification of recurrence equations and gives the definition of DURE. Section 3.2 gives the characterization of DURE. Section 3.3 formalizes the synthesis procedure.

3.1 Classification of Recurrence Equations

We first introduce the concept of system of recurrence equations in the following definition.

[Definition 1] System of Recurrence Equations (SRE) is a set of equations of the form

$$y_i(p)=f[y_1(\delta_{1i}(p)),y_2(\delta_{2i}(p)),...,y_k(\delta_{ki}(p))] \qquad (3.1)$$

where:

1. p is the index point in n-dimensional Euclidean space.
2. y_i is an array whose index set belongs to the lattice L^w.
3. f is a function. [The function form is not relevant.]
4. δ_{ij} is a dependence map function
5. C_i is the computation domain of y_i.

Notations about the definition of SRE.

1. When we to want refer to each of the individual equations we call it as Recurrence Equations (RE); When we refer to the whole set of equations, we call them as a system of REs or SREs.

2. In equation (3.1) of the definition, variable y_i at point p depends on variable y_j at δ_{ij}, denoted as:

$$y_i(p) \dashrightarrow y_j(\delta_{ji}(p)). \tag{3.2}$$

3. Variable y_i can have more than one dependency relationship with y_j. In such case, superscripts will be used.

4. Computation domains C_i, C_j,... need not be the same.

[Definition 2] An Affine Recurrence Equation (ARE) is a RE in which all the dependency maps are affine, i.e.,

$$\delta(p) = D_{ij}p+d_{ij}, \quad D_{ij} \in Z^{nxn}, d_{ij} \in z^n. \tag{3.3}$$

Here, z^n is an integer lattice point in R^n.

If in a SRE, all individual REs are ARE, we call this SRE as System of ARE (SARE).

[Definition 3] In an ARE, if all dependency maps are reduced to uniform dependency maps, i.e.,

$$\delta_{ij}(p) = p+d_{ij}, \quad D_{ij} \in Z^{nxn},, d_{ij} \in z^n. \tag{3.4}$$

then it is called Uniform Recurrence Equations (URE). If all REs in a SRE are URE, it is called system of URE (SURE).

[Example 1] Consider the following RE for decimation filter.

$$y(i,j) = y(i,j-1)+h(i,j)x(2i-j,j); \tag{3.5}$$

In this case $p=[i,j]^T$ is the index points in z^2 and the dependencies are

$$\delta_{yy}(p) = Ip + d = p + [0, -1]^T;$$
$$\delta_{hy}(p) = p;$$

$$\delta_{xy}(p) = Dp+d = \begin{bmatrix} 2 & -1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix}$$

This is an ARE, but not URE, since the dependency δ_{xy} is not uniform, Fig 10 illustrates the DG for this dependency.

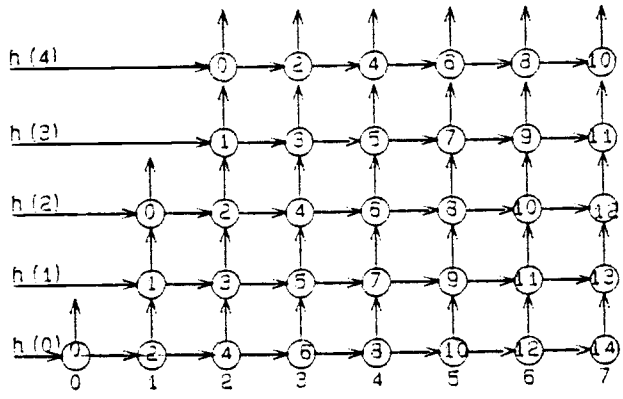


Fig 10. DG for decimation filter.

[Example 3] The following is a RE used in Toeplitz matrix factorization.

y(i,j)=y(i-1,j-1)+z(i,j)y(i-1,i-j).

Observe only the non-uniform dependency,

$$\delta_{yy}(p)=\begin{bmatrix}1 & 0\\ 1 & -1\end{bmatrix}\begin{bmatrix}i\\ j\end{bmatrix}+\begin{bmatrix}-1\\ 0\end{bmatrix}$$

This is also an ARE, not an URE. Fig 11 gives its DG.

An advantage of URE is that they can be mapped onto processor space with regular interconnections. By careful examinations of the DGs for example 1 and example 2, we found that although the dependencies are not uniform, they do show uniformity along some directions. In the DG for example 1, the dependency is uniform along the lines i-j=C. In the DG for example 2, the uniformity is along the lines of the form i-2j=C.

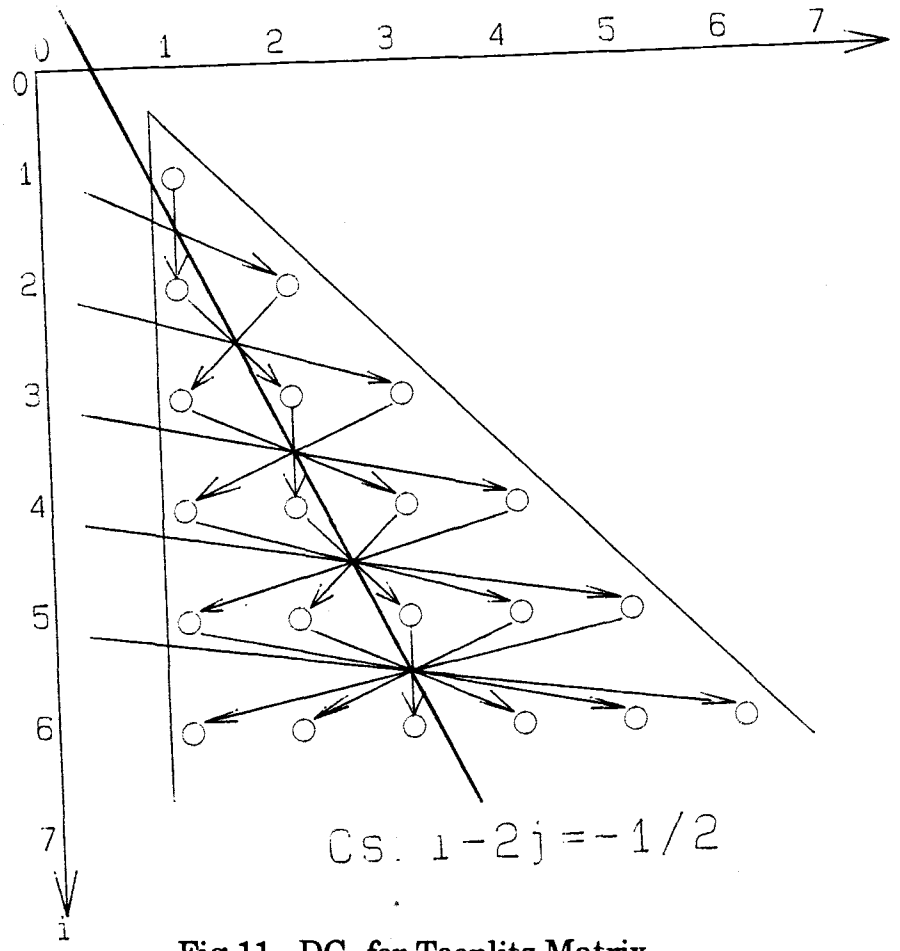


Fig 11. DG for Toeplitz Matrix.

Such observations brings the motivation for the study of a new type of REs. A common characteristics of the REs in example 2 and example 3 is that the D matrix of the affine dependency are such that D-I is singular. Moreover, in example 1,

$$D\text{-}I = \begin{bmatrix} 1 \\ 0 \end{bmatrix} [1\ 0];$$

Similarly, in example 2, D-I is also singular and

$$D\text{-}I = \begin{bmatrix} 0 \\ 1 \end{bmatrix} [1,-2].$$

Indeed, as will be shown later, if D-I is singular and can be written as $\mu\alpha^T$, where $\alpha^T\mu$ is not zero, then the AREs will present uniformity in some directions. This brings the following definition for this class of ARE.

[Definition 4] In ARE, if all dependencies Dp+d are such that D-I is singular and can be written as $\mu\alpha^T$ where $\alpha^T\mu \neq 0$, then it is called a Directional Uniform Recurrence Equation (DURE).

Generally if matrix D-I is singular and has rank 1, then one of its row vector is not zero, and all other vectors are combinations of this vectors, i.e.,

$$D\text{-}I=[k_1\alpha,k_2\alpha,...,\alpha,...]^T=[k_1,k_2,...k_n]^T[\alpha] =\mu\alpha^T; \qquad (3.6)$$

In the case rank p[D-I]>1, D-I can be written as the sum of series of the form $\mu\alpha^T$ and can be dealt similarly. In this thesis, however, only the case with p[D-I]=1 is considered.

3.2 Characterizations of DURE.

In this section, it'll be shown that DUREs have some nice properties. Such properties allows one not only be able to project their DGs in an appropriate direction such that the resulted array has regular interconnections, but also be able to find the linear scheduling functions easily.

First we demonstrate the use of MRA in DURE by an simple example before we give any formal results.

[Example 3] Consider an ARE with the following dependency

$$x(p) \dashrightarrow y[\delta_{yx}(p)],\ p=[i,\ j]^T \tag{3.7}$$

$$\delta_{yx}(p) = \begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Matrix D satisfy

$$D\text{-}I = \begin{bmatrix} 2 & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} [2\ 0];$$

The dependency arcs from y to x in a DG is shown in Fig 12 where x(i,j) -> y(3i,j).

In this example,the dependency is not uniform in the j direction but is uniform in the i direction. This example is a DURE and is directional along the direction $e=[0\ 1]^T$. If the DG is projected along the i axis $\mu=[1,0]^T$, and the timing function is specified as t(i,j)=i, the spatial locality among PEs is preserved but the temporal locality is not preserved (e.g., x(i,j)-> y(3i,j) at time 3t).One solution is to delay the computation of x(i,j) until t=3i.This corresponds to transform the node x(i,j) to node [3i,j] which results in enlarging the computation domain and increasing the latency by a factor of 3.

For the above DURE, a MRA scheme could be used with spatial and temporal locality. Still choosing the projection as $\mu=[1,0]^T$ the following MRA timing function is obtained.

Let $t_x(i,j) = i$ be the scheduling function for x and ty the scheduling function for y satisfying:

$$t_y(3i,j) \leq t_x(i,j) = i.$$

Choose

$$t_y(3i,j) = i \text{ or } t_y(i,j) = \frac{i}{3}.$$

Clearly, the computation of y is 3 times faster than x resulting in a MRA as shown in Fig 12. Notice since the computations of x and y are at different speeds the inputs to the two units are entered in multi-rate also.

[Lemma 1] In the DG of DURE, there exists a set of parallel hyperplanes such that the dependencies on these hyperplanes are uniform.

[Proof] For dependency p-->Dp+d, the dependency vector T at any point p can be written as

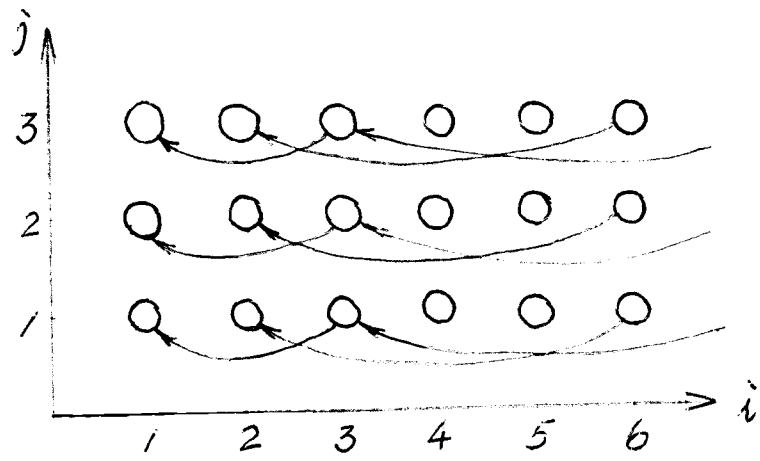
$$T = [Dp+d]-p = [D-I]p+d;$$

Since it is DURE, D-I can be expressed as $\mu\alpha^T$, thus

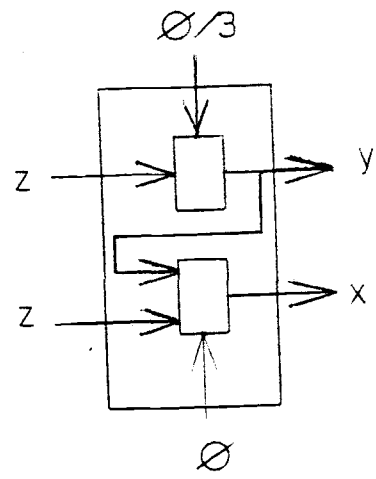
$$T = \mu\alpha^T p+d;$$

From algebra, we know $\alpha^T p=C$ defines a hyperplane in n-space and on this hyperplane

$$T = \mu\alpha^T p+d = \mu C+d = \text{constant vector;} \qquad (3.8)$$



(a) DG for the simple case of DURE.



(b) MRA structure for the simple case of DURE.

Fig 12. Application of MRA to DURE.

Since we assume p is an arbitrary point, $\alpha^T p=C$ is just one hyperplane from the set of parallel hyperplanes $\alpha^T p=C_1, \alpha^T p=C_2$, The same result applies to all these hyperplanes. Thus the lemma is proved. [Q.E.D]

For ease of discussion, in the following we call these parallel hyperplanes as Uniform Hyper-Planes (UHPs).

We then immediately have the following theorem about the processor allocation function.

[Theorem 2] For any DURE, there exists a linear processor allocation with an UHP as the processor space and as the projection vector.

[Proof] Refer to Fig 13. Consider the dependency vector T at point p which is on the UHP $\alpha^T p=C$. Denote Ts as the projection of T along μ onto the processor space. The theorem is proved if we can show that T_s is invariant in the entire domain, that is, T_s is independent of the index point p we pick.

First from lemma 1, we can express T as

$$T = \mu C+d \tag{3.9}$$

Since Ts is the projection of T along μ, T can be expressed as

$$T = T_s + x\mu \tag{3.10}$$

or

$$x\mu = T - T_s; \tag{3.11}$$

where x is a scalar to be determined.

Since the processor space is parallel to the UHPs, it must be true

$$\alpha^T T_s = 0; \tag{3.12}$$

Multiply both sides of (3.11) by α^T, we have

$$\alpha^T(x\mu)= x\,(\alpha^T\mu) = \alpha^T T - \alpha^T T_s$$
$$= \alpha^T T$$
$$=\alpha^T(\mu C+d) \qquad (3.13)$$

Since in the definition of DURE, it is assumed $\alpha^T\mu\neq 0$, we can divide both sides of (3.13) by the scalar $\alpha^T\mu$. Thus,

$$x=\frac{\alpha^T(\mu C+ d)}{\alpha^T\mu} \qquad (3.14)$$

Substitute back to (3.10) and rearrange we get

$$T_s = T - x\mu$$
$$= (\mu C+d) -(C+\frac{\alpha^T d}{\alpha^T\mu})\,\mu$$

$$= d -\frac{\alpha^T d}{\alpha^T\mu}\,\mu \qquad (3.15)$$

Observe the expression for Ts in (3.15), p or C does not appear. That is, Ts is independent of p. Thus the theorem is proved.

[Q.E.D]

Next we want to show the multi-rated dependency along the norm directions of UHPs. Such a property enables us to map DUREs onto MRAs. Before we can show such a result, we first have to find an UHP as the time reference. This is given in the following lemma.

To give the concrete view of these ideas, the formal proof for all the following lemma and theorems are placed in appendix A.

[Lemma 3] For DURE, there exists an UHP in its DG such that T is parallel to this UHP.

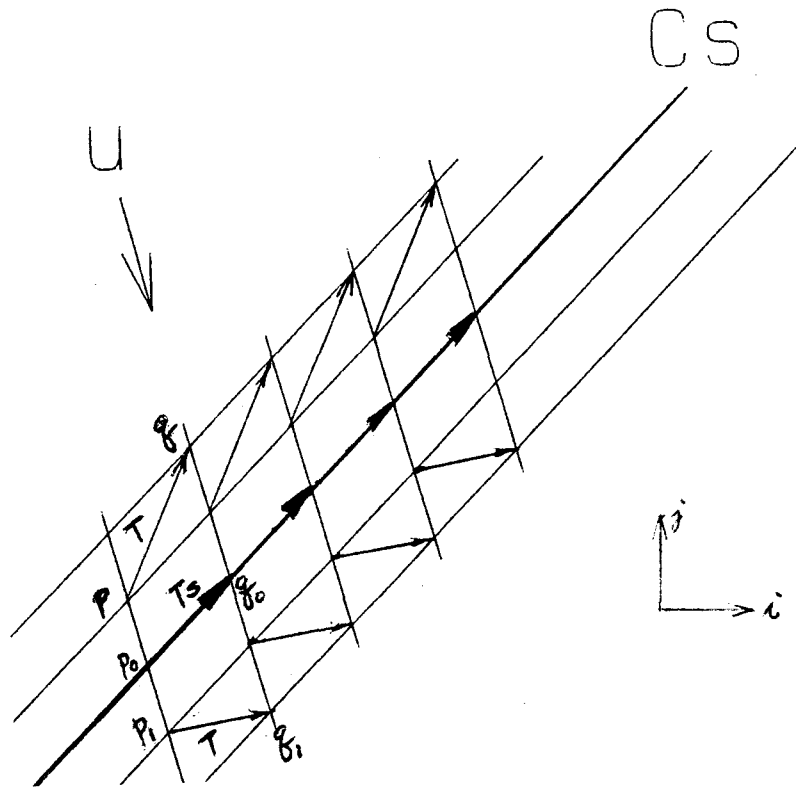


Fig 13. Characteristics of DURE DG.

Since we will frequently refer to this UHP, we assume it is expressed as

$$\alpha^T p = C_s \tag{3.16}$$

and simply call it UHP C_s. In the proof procedure in Appendix A, it is found the formula for C_s is as

$$C_s = -\frac{\alpha^T d}{\alpha^T \mu}$$

[Theorem 4] For any dependency vector form node p on UHP $\alpha^T p=C_s+C_i$, their end points lie on an UHP defined as

$$\alpha^T p=C_s+rC_i\ ; \tag{3.17}$$

where $r=1+\alpha^T\mu$ is an integer constant.

The proof can be found in appendix A. Here we only interpret the significance of such a property. We first assume r>1.

For clearance, suppose the dependency is

x(p)-->y(Dp+d)

and we call the UHP $\alpha^T p=C_s+C_i$ as C_i and call UHP $\alpha^T p=C_s+rC_i$ as rC_i. Then (3.17) tells us that any x at C_i depends on some y at rC_i. The causality condition of scheduling function requires the computation of y at rC_i must be done before computation of x at C_i, i.e.,

$$t_y(rC_i) \le t_x(C_i) \tag{3.18}$$

Since we want to get linear timing functions, t_x, t_y are assumed linear, thus

$$rt_y(C_i) \le t_x(C_i) \tag{3.19}$$

or

$$t_y(C_i) \le \frac{1}{r} t_x(C_i) \tag{3.20}$$

Think of the UHPs as computation wave-fronts, (3.20) says, to travel from base C_s to UHP C_i, it must take the wavefronts of y in no more than $\frac{1}{r}$ of the time it takes for the wavefronts of x. In other words, computation of y must be at least r times as fast as that of x. Thus an MRA similar to example 3 could be adopted. Here, UHP serves as the timing hyperplanes, r as the computation speed ratio of a pair of variables having such DURE dependency relations.

For cases where r<-1, a suitable timing function can not be obtained from theorem 4 alone. Fortunately, all DURE shows symmetry in their DGs. Such symmetry property enables one to fold the computation domain along the symmetrical plane to convert the DURE into one whose r>1.

Formally, we have the following theorem regarding to the symmetry of DURE's DG.

[Theorem 5] The DG for DURE is symmetrical around C_s along the direction of μ in the sense that the structures on both sides of C_s are mirror images of each other.

3.3 Synthesis of DURE

Having done the characterizing DURE, we are now able to formalize the synthesis of DURE. The processor assignment is actually already specified by theorem 2, which implies the vector μ can be always served as the projection vector, while the UHP is the processor space. It is the scheduling function that needs to be classified according to the different r.

Fig 3.7 illustrates the DG structures for different case of r The corresponding scheduling functions can be correspondingly constructed as follows.

1) r>1. As mentioned in section 3.2, a MRA scheme can be adopted. UHPs are served as the timing hyperplane, and Cs is the time reference.

2) r=1. No corresponding DURE exists, since $r=1+\alpha^T\mu=1$ implies $\alpha^T\mu=0$, which contradicts the assumption $\alpha^T\mu\neq 0$ in the definition of DURE.

3) r=0. The DG structure is as shown in Fig 14(b). The end points of any dependency vectors stay on Cs. It can shown that D in such case is itself singular, and a pipelining technique similar to [6][32] can be adopted to convert the DURE to URE.

4) r=-1. The DG structure is shown in Fig 14(c). By theorem 5,such a DURE can be converted to URE by folding the computation domain along Cs. The scheduling function thus can be obtained as in ordinary URE. An application example of such a DURE is Toeplitz matrix factorization and will be illustrated in chapter 4.

5) r<-1. The DG structure is shown in Fig 14(d). The general strategy is first to fold along Cs to convert it to the DURE of type 1 and then apply corresponding manipulations.

The systematic synthesis for DURE is thus formulated as follows. The algorithm is assumed to have been already specified.

Step 1. Find the affine dependency $\delta(p)=Dp+d$, and factor D-I into form

$$D-I=\mu\alpha^T \tag{3.21}$$

Step 2. Draw the UHP Cs which is specified as

$$\alpha^T p=C_s=\frac{\alpha^T d}{\alpha^T \mu} \tag{3.23}$$

Step 3. Compute r according to

$$r=1+\alpha^T\mu \tag{3.22}$$

(a) r>1. Adopt MRA scheme, r is the clock (or computation speed) ratio between variables. UHPs are the timing hyperplane, Cs is the time reference. Continue to step 4.

(b) r=0. Adopt pipelining method as in [6][32].

(c) r=-1. Convert DURE to URE by folding along C_s Adopt any convenient URE design method.

(d)r<-1. Convert it into a DURE of type (a) by folding along C_s. Adjust r value. Continue to step 4.

Step 4. Project DG onto C_s along the direction of μ.

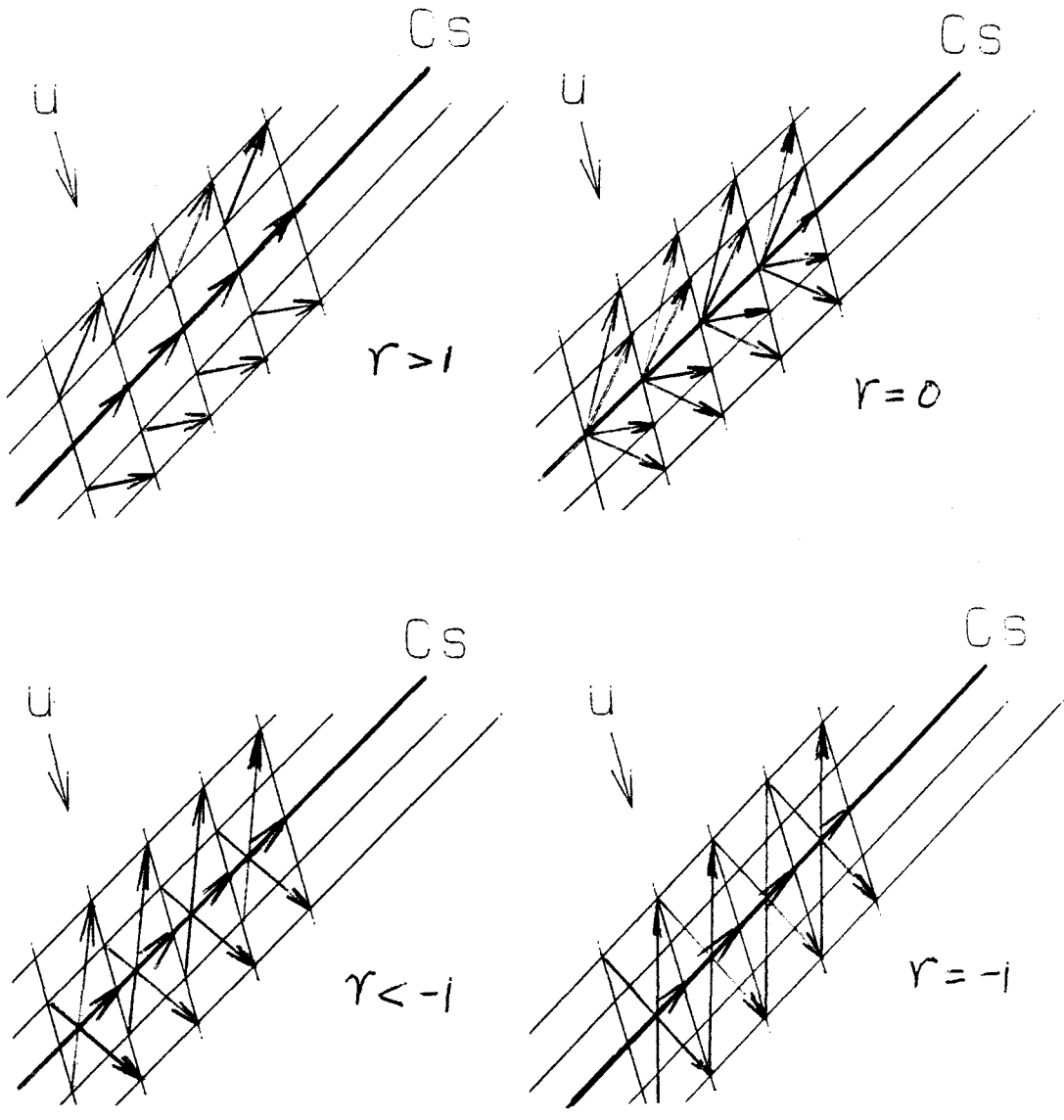


Fig 14. Different DG structure of DURE.

CHAPTER 4

APPLICATION EXAMPLES OF DURE

In chapter 3, we have given a method for the synthesis of DURE. In this chapter, we'll illustrate the synthesis of DURE by two examples in signal processing applications. Section 4.1 gives the MRA solution to the digital decimation filter which is often used in telecommunication for code conversion. Section 4.2 gives the MRA solution for Toeplitz matrix decomposition.

4.1 Decimation Filter

In telecommunication, it is often desired to convert one digital code form to another. When a over-sampled signal needs to be converted to a form with lower sampling frequency, a decimation devices is needed. Fig 15 is the block diagram for the decimation filter, where the box h(n) serves as an anti-aliasing filter. The relationship between x(n) and y(m) is

$$y(i)=\sum_{j=-\infty}^{\infty} h(j)x(Mi-j)=\sum_{j=-N}^{N} h(j)x(Mi-j) \qquad (4.1)$$

SYSTTOLIC ARRAY SOLUTION

Assume h(j) $\neq 0$ only in [0,N]. Converting it to a SAC form we get:

$$y(i,j)=h(i,j)x(Mi-j)+y(i,j-1) \qquad (4.2)$$
$$y(i,0)=0$$
$$h(i,j)=h(i-1,j)$$

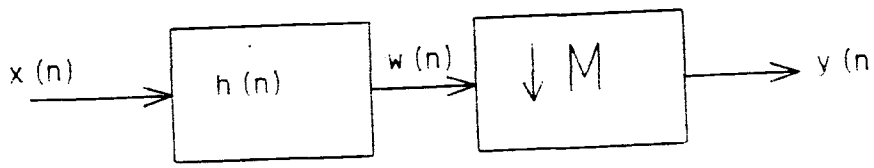
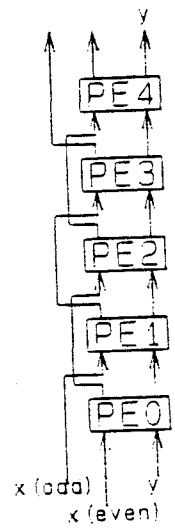


Fig 15. Decimation Filter.



Note: h resides in PEs

Fig 16. Systolic Array for decimation filter.

The desired output is stored in y(i,N). The initial DG is given in Fig 10.

For the systolic array design, treat the above as URE and pipeline x along the lines Mi-j = C. The resulting array is resulted as shown by Fig 16 (M=2). Such a systolic array needs M path for x. Obviously, such a solution is not practical, since, as M increases the number of buses and I/O paths for x increases.

MRA SOLUTION

For multi-rate implementation examine x(i) at i=i1=constant and obtain the ARE. The dependency graph for $\delta_{xy}(p)$ is illustrated in Figure 17(a) with the following ARE:

$$\begin{aligned} y(i,j) &= y(i,j-1) + x(2i-j,j)h(i,j) \\ x(i,j) &= x(i,j-1) \\ h(i,j) &= h(i-1,j) \end{aligned} \tag{4.13}$$

Following the procedure outlined in chapter 3, we have,

Step 1. The affine dependency is

$$\delta_{xy}(p) = \begin{bmatrix} 2 & -1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} = Dp$$

$$D - I = \begin{bmatrix} 1 \\ 0 \end{bmatrix} [1 \ -1] = \mu\alpha^T$$

Step 2. Compute r as

$$r = 1 + \alpha^T\mu = 2 > 1$$

According to chapter 3, a MRA can be adopted. But before going further, we first apply the following transformation,

$$M=\begin{bmatrix}1 & -1\\ 0 & 1\end{bmatrix}; \qquad M^{-1}=\begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}$$

The ARE after the transformation could be obtained as:

$$\begin{aligned} y(i,j) &= y(i+1,j-1) + x(2i,j)h(i,j) \\ x(i,j) &= x(i+1,j-1) \\ h(i,j) &= h(i-1,j) \end{aligned} \tag{4.4}$$

The directions of the dependency arcs does not allow us to use the timing function $t_y(i,j) = i$. But notice, since the dependency δ_{yy} is an addition operation and δ_{xx} is a propagation operation, the direction of the dependency arcs for both variables can be reversed using associative property.

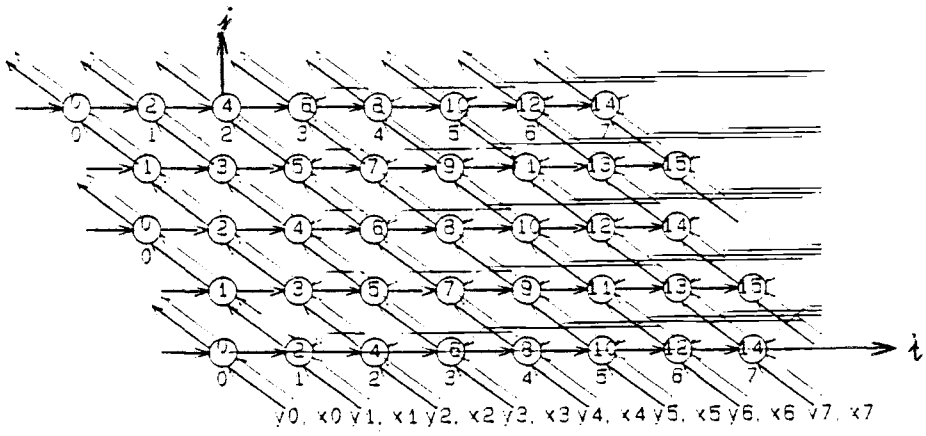
Step 3. Obtain scheduling function. In the DG in Fig 17(a), the UHPs can be observed directly by inspection. They are UHPs i=C. The C_s plane is just the j axis. The timing function is thus obtained as,

$$t_y(i,j) = i,$$
$$t_x(2i,j) = i, \text{ or } t_x(i,j) = \frac{i}{2}.$$

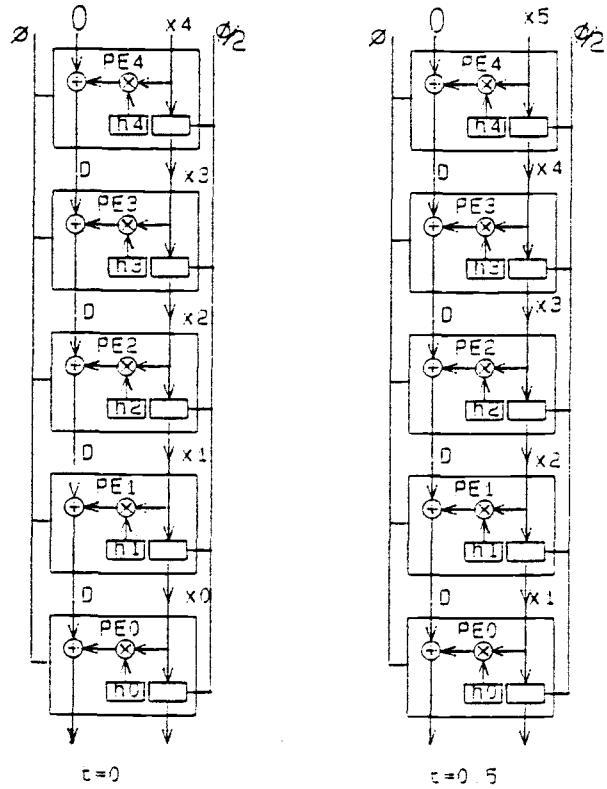
Here operation of x is twice as fast as y (M=2).

Step 4. It is still obtained by inspection that μ is [1,0]. Thus, the DG should be projected along μ.

The resulting array structure and its snapshots are shown in Figure 17(b).



(a) Modified DG for decimation filter.



(b) MRA structure.

Fig 17. MRA for decimation filter.

4.2 Solving Toeplitz System.

Problem Specification

[Toeplitz Matrix Factorization] For a Toeplitz matrix R with r(i,j) = t(|i-j|), or

$$R = \begin{bmatrix} t_0 & t_1 & . & . & t_{n-1} \\ t_1 & t_0 & . & . & t_{n-2} \\ . & . & . & . & . \\ . & . & . & t_0 & t_1 \\ t_{n-1} & t_{n-2} & . & t_1 & t_0 \end{bmatrix} \tag{4.5}$$

find a lower-triangular matrix Y with one on the diagonal and an upper-triangular matrix X such that

$$YR=X \tag{4.6}$$

where

$$Y = \begin{bmatrix} 1 & & & & \\ y_{21} & 1 & & & \\ y_{31} & y_{32} & 1 & & \\ . & . & . & . & \\ y_{n1} & y_{n2} & . & y_{n,n-1} & 1 \end{bmatrix}, \quad X = \begin{bmatrix} x_{11} & x_{12} & . & . & x_{1n} \\ & x_{22} & . & . & x_{2n} \\ & & . & . & . \\ & & & . & . \\ & & & & x_{nn} \end{bmatrix}$$

Algorithm formulation

Details of the derivation of the SRE for the above problem are avoided here. The final SRE is found to be

$$
\begin{aligned}
&z(i) = - w(i-1,i)/x(i-1,j-1) \\
&y(i,j) = z(i)y(i-1,i-j)+y(i-1,j-1) \quad j \le i \\
&x(i,j) = z(i)w(i-1,j) + x(i-1,j-1) \quad j \ge i \\
&w(i,j) = z(i)x(i-1,j-1) + w(i-1,j) \quad j > i
\end{aligned}
\tag{4.7}
$$

$$
\begin{aligned}
&x(1,j) = t(j-1) \quad j \ge 1 \\
&w(1,j) = t(j-1) \quad j > 1 \\
&y(1,1) = 1 \\
&y(i,0) = 0 \quad \text{for all i.}
\end{aligned}
$$

Here array variable z and w are temporary variables used to hold immediate results. The computation domains for x,y and w are shown in Fig 18.

DURE DESIGN

SRE (4.18) is a DURE since the dependency $\delta_{yy}(p)$ is

$$
\delta_{yy}(p)=\begin{bmatrix}1 & 0\\ 1 & -1\end{bmatrix}\begin{bmatrix}i\\ j\end{bmatrix}+\begin{bmatrix}-1\\ 0\end{bmatrix} \tag{4.8}
$$

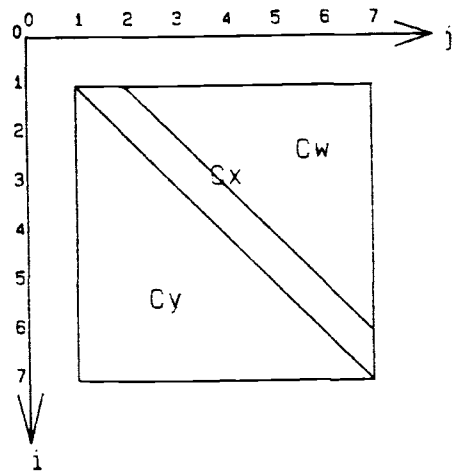


Fig 18 Computation Domains of Toeplitz Matrix.

But carefully examine (4.7), we can find that the whole SRE can be split into two separate parts -- the computation of array variable x and w in the upper part and the computation of array variable y in the lower part. The computation of x and w does not require any result from that of y, while computation of y depends on x and w via the variable z. Thus we got two sub SREs as follows.

SRE 1:

$$\begin{aligned} &z(i)=-w(i-1,i)/x(i-1,i-1) \\ &x(i,j)=x(i-1,j-1)+z(i)w(i-1,j) \\ &w(i,j)=w(i-1,j)+z(i)x(i-1,j) \end{aligned} \tag{4.9}$$

$x(1,j)=t(j-1)$, $w(1,j)=t(j-1)$,
$x \in C_x$, $w \in C_w$

SRE 2:

$$y(i,j)=z(i)y(i-1,j-1)+y(i-1,j-1) \tag{4.10}$$

with
$y(1,1)=1$; $y(i,0)=0$;
$y \in C_y$, $z(i)$ is an input array variable.

Consequently, we could design the array processor architecture separately, one for SRE 1 and one for SRE 2.

UPPER PART DESIGN

Notice SRE 1 is an URE and can be easily designed by using conventional approach. Here, we only give some special consideration for this particular problem.

From Fig 19(a), it is noticed that although the x values required by z are on the boundary, those w values required by z are not. Moreover, Their correponding dependency arcs from w to z on the boundary are different from those arcs originating from w in the interior part of the computation domain. These two facts means that we need to give a different design for the node on the boundary, which is not desirable.

Such a problem can be avoided by shifting C_w along j axis by one position, i.e., to apply the following transformation,

$$M_w(p)=p+[0,-1], \qquad M_w^{-1}(p)=p+[0, 1]$$

SRE 1 is thus changed to

$$\begin{aligned} z(i)&=-w(i-1,j-1)/x(i-1,j-1) \\ x(i,j)&=z(i)w(i-1,j-1)+x(i-1,j-1) \\ w(i,j)&=z(i)x(i-1,j)+w(i-1,j) \end{aligned} \tag{4.11}$$

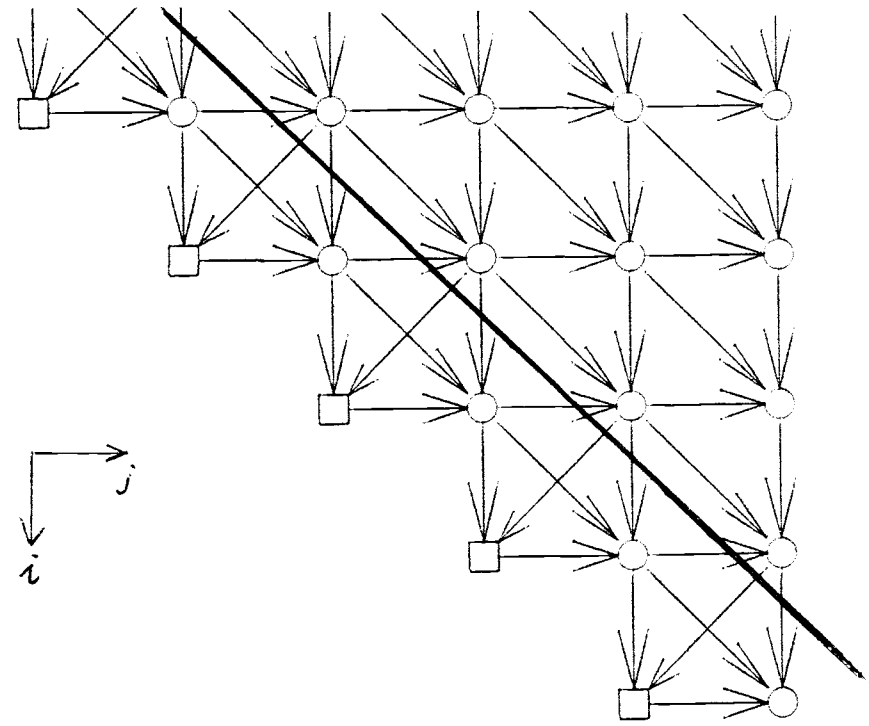
Further localizing the propagation of z variable, we have a DG as shown in Fig 19(b). It is seen that the w values required by z are on the boundary and the dependency arcs are the same throughout the entire domain. A final array structure in Fig 19(c) is obtained by choosing $s=[1,0]^T$ or $s=[1,1]^T$.

LOWER PART DESIGN

To deal with SRE 2, which is a DURE, follow the procedure outlined in chapter 3. The DG is shown in Fig 11.

Step 1. Notice the affine dependency is

$$\delta_{yy}(p)=\begin{bmatrix}1 & 0\\ 1 & -1\end{bmatrix}\begin{bmatrix}i\\ j\end{bmatrix}+\begin{bmatrix}-1\\ 0\end{bmatrix}$$



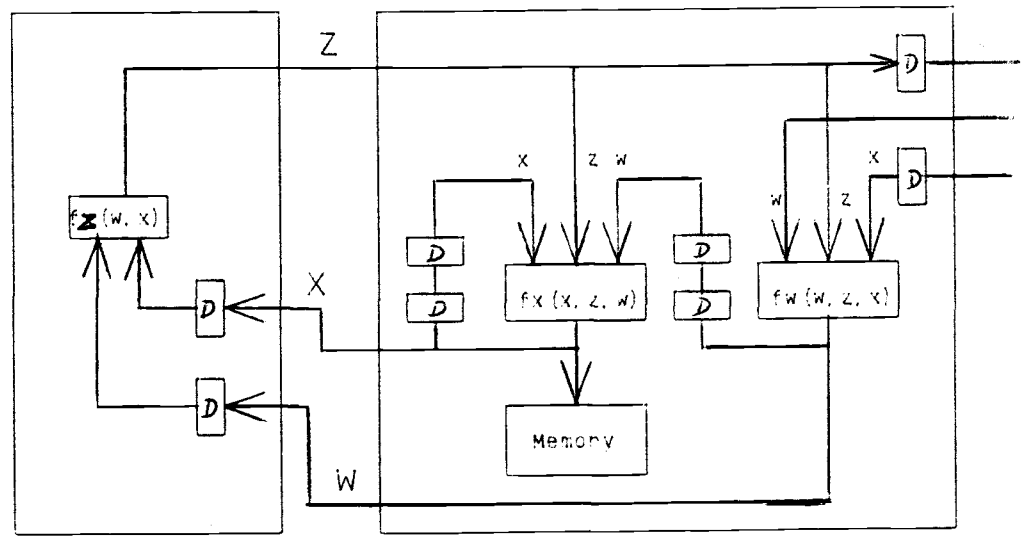


Fig 19. DG and Array Structures for Upper Part of Toeplitz Matrix.

$$D\text{-}I=\begin{bmatrix}0\\1\end{bmatrix}[1,\ -2]=\mu\alpha^T \tag{4.12}$$

Step 2. Find UHP Cs.

$$C_s=-\frac{\alpha^T d}{\alpha^T\mu}=-\frac{1}{2} \tag{4.13}$$

$$C_s:\alpha^T p=-\frac{1}{2}\ ,\text{or } i-2j=-\frac{1}{2}$$

Step 3. Compute r.

$$r=1+\alpha^T\mu=-1. \tag{4.14}$$

which suggests to convert DURE into a URE. We first transform SRE 2 to a DURE with D in diagonal form by the following transformation.

$$M_y=\begin{bmatrix}1 & 0\\ -1 & 2\end{bmatrix} \qquad M_y^{-1}=\begin{bmatrix}1 & 0\\ 0.5 & 0.5\end{bmatrix}$$

Applying the above diagonal transformation, SRE 2 is transformed to

$$\begin{aligned}&y(i,j)=y(i-1,j-1)+z(i)y(i-1,-j+1)\\ &y(1,1)=0;\quad y\in C_y\end{aligned} \tag{4.15}$$

Now

$$D=\begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}, \qquad D\text{-}I=\begin{bmatrix}0 & 0\\ 0 & -2\end{bmatrix}=\begin{bmatrix}0\\1\end{bmatrix}[0,\ -2]=\mu\alpha^T$$

$$C_s=-\frac{\alpha^T d}{\alpha^T \mu}=-1,$$

$$Cs: \alpha^T p=-1, \text{ or } j=\frac{1}{2}; \tag{4.16}$$

The new DG is shown in Fig 20.

Now we name y on the left side of C_s as v and fold the computation domain along C_s. Equation (4.15) is changed to

$$\begin{aligned} &y(i,j)=v(i-1,j-1)+z(i)v(i-1,-j+1),\ j=1\\ &y(i,j)=y(i-1,-j-1)+z(i)v(i-1,-j+1),\ j>1\\ &v(i,j)=v(i-1,j-1)+z(i)y(i-1,-j+1) \end{aligned} \tag{4.17}$$

The folding transformation M_v is,

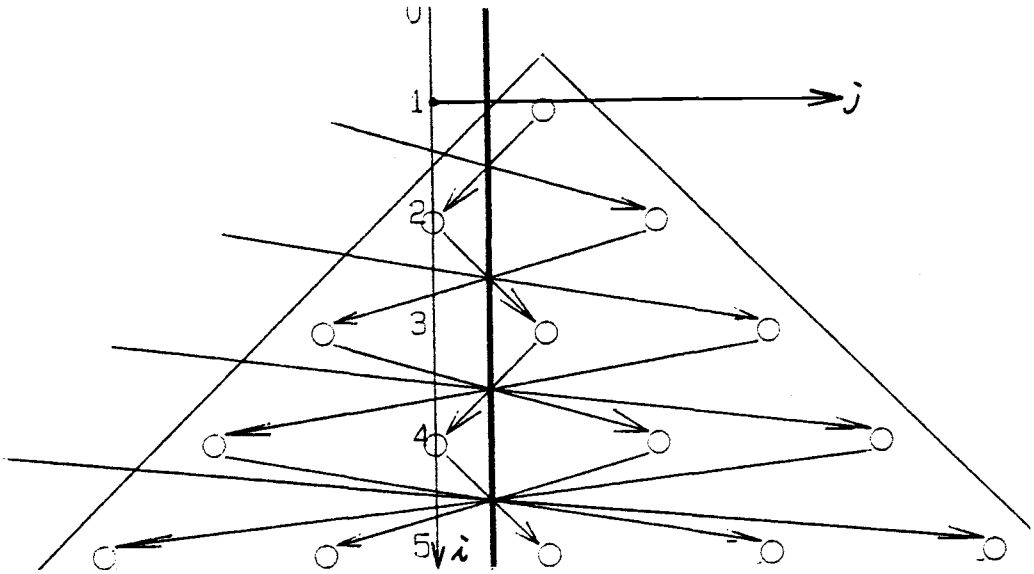
$$M_v: (i,j)\text{--->}(i,\ 1-j)$$

SRE 2 after folding is

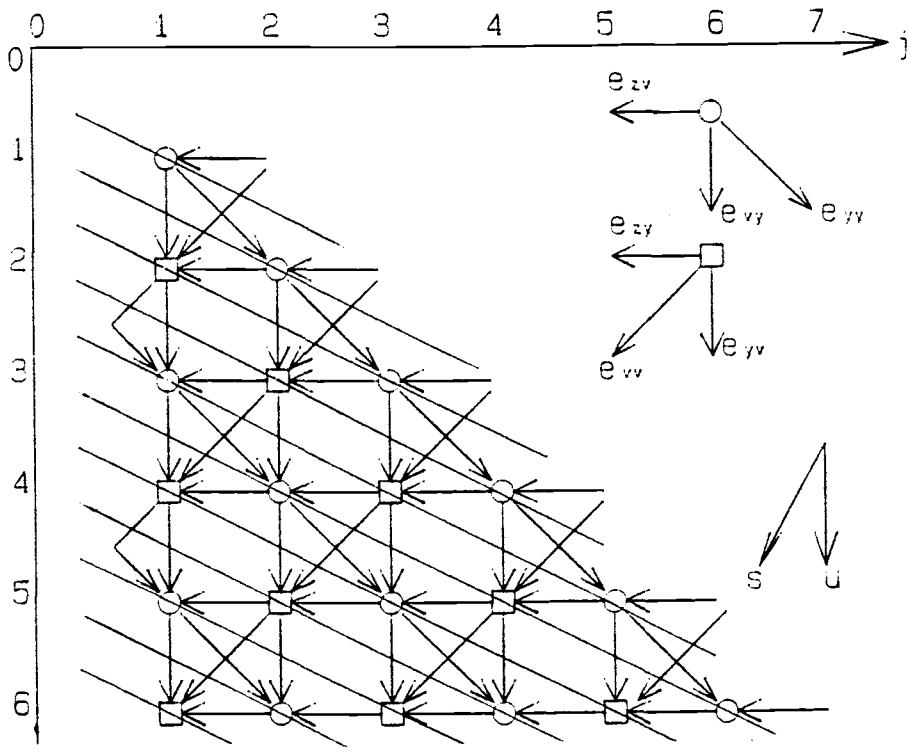
$$\begin{aligned} &y(i,j)=v(i-1,-j+2)+z(i)v(i-1,j)\\ &\quad =v(i-1,j)+z(i)v(i-1,j) \quad j=1,\\ &y(i,j)=y(i-1,j)+z(i)v(i-1,j) \quad j>1;\\ &v(i,j)=v(i-1,j+1)+z(i)y(i-1,j) \end{aligned} \tag{4.18}$$

The above equations are a SURE and its DG is shown in Fig 20. Two designs are obtained from such a SURE.

Design (a). Design (a) is obtained directly from the DG. The dependency structure can be viewed as uniform except on the boundary j=1. From (4.18), we can see that on the boundary j=1, the second term for y(i,j) is v(i-1, j) while at other nodes it is y(i-1,j-1). This can be made virtually uniform with other nodes by connecting the arc e_v and e_y on the boundary as shown.



(a) DG before folding.



(b) DG after folding.

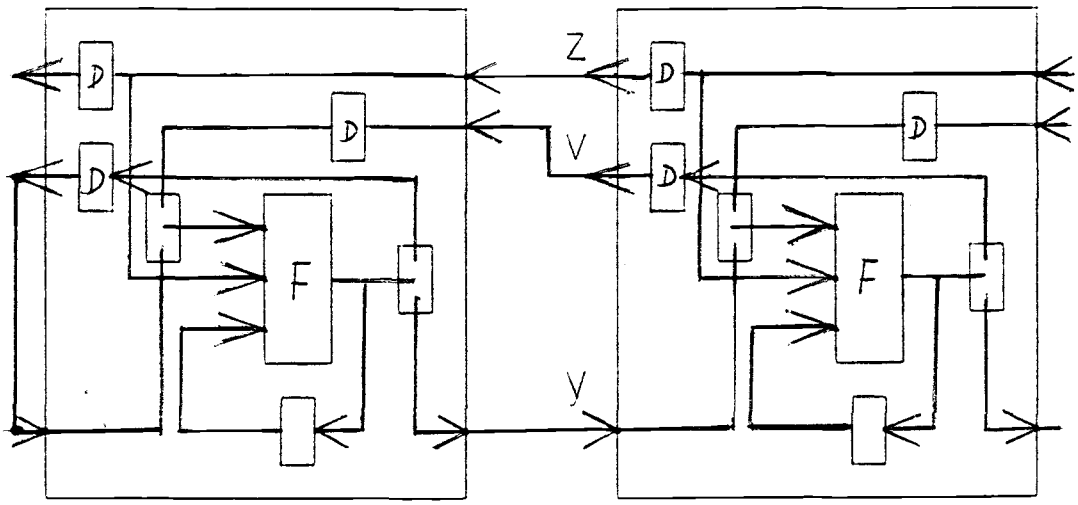
Fig 20. DG for Lower Part of Toeplitz Matrix.

Notice the node function for y and v are the same, it is the inter-PE connections varies with time. This can be accomplished by using switches. The PE structure when choosing $s=[1,0]^T$, $\mu=[2,-1]^T$ is shown in Fig 21(a).

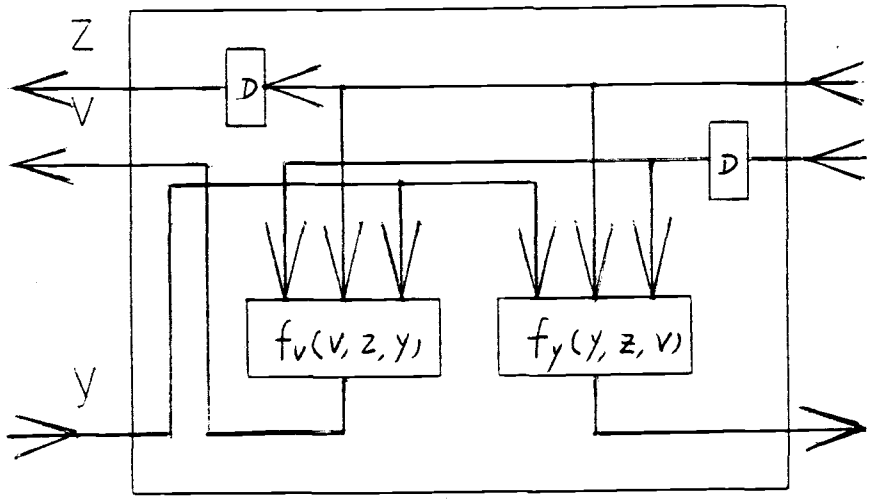
Design (b). The second design get rid of the switches by further transformations on the DG. First shift v by 1 in the j direction and then rotate the (i,j) axis set 45 degree in clockwise direction. The final SRE is transformed as follows.

$$y(i,j)=v(i-1,j)+z(i+j)v(i-1,j); \quad j=i+1$$
$$y(i,j)=y(i,j-1)+z(i+j)v(i-1,j); \quad j\geq i+1$$
$$v(i,j)=v(i-1,j)+z(i+j)y(i,j-1);$$

Choosing $\mu=[1,1]^T$, $s=[2,1]^T$, we get the array structure as shown in Fig 21(b).



Design (a) of Lower Part.



Design (b) of Lower Part.

Fig 21. Final Array Structure for the Lower Part.

Bibliography

[1]. H.T. Kung, "Why systolic arrays," Computer, 15(1):37-45, Jan. 1982.

[2]. R.M. Karp, R.E. Miller, and S. Winograd, "The organization of computations for uniform recurrence equations," J. of the Assoc. of Comput. Machinery, pp. 563-590, 1967.

[3]. P. Quinton, "Automatic synthesis of systolic arrays from uniform recurrence equations," Proc., 11th Ann. Symp. on Computer Architecture, pp. 208-214, 1984.

[4]. D.I. Moldovan, "On the analysis and synthesis of VLSI algorithms," IEEE Trans. Comput., C-31:1121-1126, November 1982.

[5]. S. Rojopadhye, "Synthesizing systolic arrays with control signals from recurrence equations," Distributed Computing, pp. 88-105, May 1989.

[6]. S. Rajopadhye, "Synthesis, verification and optimization of systolic arrays", Ph.D thesis, University of Utah, Dec. 1986.

[7]. S.K. Rao, "Regular Iterative Algorithms ...," Ph.D. thesis, Stanford University, Standford, CA, October 1985.

[8]. Y. Yaacoby and P.R. Cappello, "Bounded Broadcast in Systolic Arrays," Technical Report, Dept, of Computer Science, UCSB, Santa Barbara, April 1988.

[9]. R.E. Crochiere and L.R. Rabiner, "Multirate Digital Signal Processing", Prentice Hall, Inc., Englewood, 1983.

[10]. P. Raychowdhury, S.K. Rao, et al., "On the localization of algorithms for VLSI Processor Arrays," in VLSI Signal Processing IV, IEEE Press, 1988.

[11]. S. Kiaei and L. Aihua, "Analysis of Multirate/Bounded Broadcast," Technical Report, Elec. & Comp. Engr., Oregon State Univ., 1989.

[12]. C. Mongenet, and G. Perrin, "Synthesis of Systolic Arrays for Inductive Problems," in Parallel Architectures and Languages Europe's, June 1987.

[13]. J.M. Delosme and I.C.F. Ipsen, "Systolic Array Synthesis: Computability and Time Cones," Cosnard et al. editors, Parallel Algorithms and Architectures Conference, pp. 295-312, 1986.

[14]. L. Aihua and S. Kiaei, "VLSI Design of Multi-Rate Arrays for DSP Algorithms," 1990 Int. Conf. on Acoustics, Speech and Signal Processing, New Mexico.

[15]. S. Kiaei and L. Aihua, "Synthesis of MRAs for Toeplitz Matrix Solution," Technical Report, Elec. & Comp. Engr., Oregon State Univ. 1990; also to be submitted to Journal of VLSI Signal Processing.

[16]. S. Kiaei and J.K. Durgam, "VLSI Design of Dynamically Reconfigurable Array Processors - DRAP," ICASSP, Glasgow, Scotland, 1989.

[17]. Y. Yaacoby and P. Capello, "Scheduling a System of Non-singular AREs onto a Processor Array," J. of VLSI Signal Processing, Vol. 1, No. 2, October 1989, Kluwer Academic Publishers.

[18]. R.E. Gomory, "On the relation between integer and non-integer solutions to linear programs," Proc. N.A.S, Vol, 53, pp. 260-265, 1965.

[19]. R.E. Gomory, "Faces of an integer polyhedron," Proc, N.A.S, Vol, 57, pp. 16-18, 1967.

[20]. E. Kamen, "Introduction to signal and Systems," MacMillan Publishing Company, New York, 1987.

[21]. H.A. Taha, "Integer Programming, Theory, Application and Computations," Academic Press, 1975.

[22]. K. Nomizu, "Fundamentals of Linear Algebra," McGran-Hill, Book Company, 1966.

[23]. W. Moore, et, "Systolic Arrays," Adan Hilger, 1987.

[24]. P. Denyer & D. Renshow, "VLSI signal processing, a bit-serial approach", Addisn-Wesley Publishing Company, 1985.

[25]. S.I. Gass, "Linear programming," McGraw-Hill Book Company, 1975.

[26]. E.E, Smartzlander, J, "Systolic Signal Processing Systems," Marcel Dekker, Inc, 1987.

[27]. G.J. Myers, "Advances in Computer Architecture," John Wiley & Sons, Inc, 1982.

[28]. M.J. Rochkind, "Advanced Unix Programming," Prentice-hull, Inc, Eaglewood, 1985.

[29]. P. Quinton, "The systematic design of systolic arrays," IRISA, International Publication 193, April, 1983.

[30]. P.R. Cappello, et al, "Unifying VLSI array designs with geometric transformations, Proc. Int. Conf. on Parallel processing, Columbus, OH 448-457. Aug. 1983.

[31]. C.E. Leiserson, et al, "Optimizing synchronous circuitry by retiming," 3rd Caltech. Conf. on VLSI, in R. Bryont ed., Computer Science Press, Rockville, MD, pp. 87-116, 1983.

[32]. J.A.B. Fortes, & D.I. Moldevan, "Data broadcasting in linearly scheduled array processors," Proc. Third Caltech Conf. on VLSI, Computer Science Press, Rockville, MD, 1983.

[33]. R.M. Karp & R.E. Miller, "Properties of a model for parallel computations; determinacy, termination, queuing," SIAM, J. Appl. Math., Vol 14, No.6, pp. 1390-1411, Nov. 1966.

[34]. K. Hwang, "Computer architecture and parallel processing," McGraw-Hill, Inc, 1984.

[35]. S.Y. Kung, "VLSI array processors", Prentice-Hall, Inc, 1988.

[36]. I.V. Rannakrishman, et al, "Mapping homogeneous Graphs on Linear Arrays", IEEE Trans. on computers, Vol. c-35, March, 1986.

[37]. D.I.Moldovan & J.A.B. Fortes, "Partitioning and mapping algorithms into Fixed Size Systolic Arrays", IEEE Trans, on computers, Vol. c-35, No.1, Jan. 1986.

[38]. C. Choffrut & Culik II, "Folding of the plane and the design of systolic arrays", Information letters, Vol. 17, No. 3, Oct. 1983.

[39]. W.L, Miranker & A. Winkles, "Spacetime Representations of Computational Structures", Computing 32, pp. 93-114, 1984.

[40]. Y. Yaacoby and P.P. Cappello, "Converting Affine Recurrence Equations to Quasi-Uniform Recurrence Equations," Technical Report, Dept. of Computer Science, UCSB, Santa Barbara, CA, February 10, 1988.

APPENDIX

APPENDIX

PROOFS OF SOME THEOREMS

This appendix gives the formal proofs to those lemma and theorems in chapter 3. Figures are refereed to chapter 3.

[Lemma 4] For DURE, there exists an UHP in its DG such that T is parallel to this UHP.

[Proof] Assume such an UHP exists and its corresponding equation is

$$\alpha^T p = C_s$$

If we can get a determined value for C_s , then such a plane does exist and the lemma is proved.

From (3.8) in Lemma 1. T's on this UHP is

$$T = \mu\alpha^T p + d = \mu C_s + d$$

From the assumption, T is parallel to UHPs,

$$\alpha^T T = 0$$

or

$$(\alpha^T \mu) C_s + \alpha^T d = 0$$

$$(\alpha^T \mu) C_s = -\alpha^T d \qquad \text{(A.1)}$$

From the definition of DURE, scalar $\alpha^T\mu$ is nonzero.

$$C_s = -\frac{\alpha^T d}{\alpha^T \mu} \qquad \text{(A.2)}$$

[Q.E.D]

[Theorem 4] For any dependency vector form node p on UHP $\alpha^T p=C_s+C_i$, their end points lie on an UHP defined as

$$\alpha^T p=C_s+rC_i \; ; \tag{A.3}$$

where $r=1+\alpha^T\mu$ is an integer constant.

[Proof] The end points of dependency vectors from C_i is

$$\begin{aligned}\delta(p) &= Dp+d \\ &= (I+\mu\alpha^T)p+d \\ &= p+\mu\alpha^T p+d\end{aligned}$$

Multiply both sides by α^T, we have

$$\begin{aligned}\alpha^T\delta(p) &= \alpha^T p+\alpha^T\mu\alpha^T p+\alpha^T d \\ &= (C_s+C_i)+\alpha^T\mu(C_s+C_i)+\alpha^T d \\ &= C_s+(1+\alpha^T\mu)C_i+(\alpha^T\mu C_s+\alpha^T d)\end{aligned} \tag{A.4}$$

From (A.2),

$$C_s = -\frac{\alpha^T d}{\alpha^T\mu}$$

(A.4) becomes

$$\begin{aligned}\alpha^T\delta(p) &= C_s+(1+\alpha^T\mu)C_i \\ &= C_s+rC_i\end{aligned}$$

that is, $\delta(p)$ is located at the UHP defined by

$$\alpha^T p = C_s+rC_i$$

where

$$r = 1+\alpha^T\mu$$

[Q.E.D]

[Theorem 5]. The DG for DURE is symmetrical around C_s along the direction of μ such that the structures on both sides of C_s are mirror images of each other.

[Proof]. Refer to Fig 13. First we can obtain the mirror transformation as follows.

Suppose p is any point on $-C_i$. Let p_1 be its image, p_0 be their projection on C_s. From Fig 13, we have

$$p-p_0=p_0-p_1=x\mu \tag{A.5}$$

where x to be determined. Also from assumption,

$$\alpha^T p=C_s-C_i \tag{A.6}$$

$$\alpha^T p_0=C_s \tag{A.7}$$

or

$$\alpha^T(p-p_0)=-C_i$$

From (A.5),

$$\alpha^T(p-p_0)=\alpha^T(x\mu)=x(\alpha^T\mu) \tag{A.8}$$

Combining (A.6) and (A.7), we have

$$x(\alpha^T\mu l)=-C_i$$

Since $\alpha^T\mu \neq 0$,

$$x=-\frac{C_i}{\alpha^T\mu}=-\frac{C_s-\alpha^T p}{\alpha^T\mu} \tag{A.9}$$

Thus from (3.25),

$$p_0=p-x\mu=p+\frac{C_s-\alpha^T p}{\alpha^T\mu}\mu \tag{A.10}$$

Combining these results, we have

$$p_1=2p_0-p=p+2\frac{C_s-\alpha^T p}{\alpha^T\mu}\mu$$

Thus, the image transformation M(p) for point p is defined as

$$M(p)=p+2\frac{C_s-\alpha^T p}{\alpha^T\mu}\mu \tag{A.11}$$

Suppose the dependency vector T(p) at p ends at q, vector $T(p_1)$ at p_1 ends at q_1, where p_1 is the image of p. We have only to show that the image of q coincident with q_1, i.e, $M(q)=q_1$.

$$q_1=Dp_1+d=D[p+2\frac{C_s-\alpha^T p}{\alpha^T\mu}\mu]+d \tag{A.12}$$

$$q=Dp+d$$

From(A.11),

$$M(q)=q+2\frac{C_s-\alpha^T q}{\alpha^T\mu}\mu$$

Since q is an end point of dependency vector from $-C_i$, we know from lemma 3,

$$\alpha^T q=C_s+rC_i \tag{A.13}$$

or

$$C_s-\alpha^T q=rC_i=r(C_s-\alpha^T p) \tag{A.14}$$

$$M(q)=q+2\frac{r(C_s-\alpha^T p)}{\alpha^T\mu}\mu$$

$$=(Dp+d)+2\frac{r(C_s-\alpha^T p)}{\alpha^T \mu}\mu$$

$$=Dp+2\frac{C_s-\alpha^T p}{\alpha^T \mu}r\mu+d$$

Notice

$$\begin{aligned} r\mu &= (1+\alpha^T \mu)\mu \\ &= 1+\mu(\alpha^T \mu) \\ &= (I+\mu\alpha^T)\mu \\ &= D\mu \end{aligned}$$

M(q) can be simplified as

$$M(q)=Dp+2\frac{C_s-\alpha^T p}{\alpha^T \mu}D\mu+d$$

$$=D[p+2\frac{C_s-\alpha^T p}{\alpha^T \mu}\mu]+d$$

$$=q$$

[Q.E.D]