AN ABSTRACT OF THE DISSERTATION OF

Yue-Peng Zheng for the degree of Doctor of Philosophy in Electrical and Computer Engineering presented on May 27, 1994.

Title: Mapping of Recursive Algorithms onto Multi-Rate Arrays

Abstract approved: Redacted for Privacy

Sayfe Kiaei

In this dissertation, multi-rate array (MRA) architecture and its synthesis are proposed and developed. Using multi-coordinate systems (MCS), a unified theory for mapping algorithms from their original algorithmic specifications onto multi-rate arrays is developed.

A multi-rate array is a grid of processors in which each interconnection may have its own clock rate; operations with different complexities run at their own clock rate, thus increasing the throughput and efficiency.

A class of algorithms named directional affine recurrence equations (DARE) is defined. The dependence space of a DARE can be decomposed into uniform and non-uniform subspaces. When projected along the non-uniform subspace, the resultant array structure is regular. Limitations and restrictions of this approach are investigated and a procedure for mapping DARE onto MRA is developed.

To generalize this approach, synthesis theory is developed with initial specification as affine direct input output (ADIO) which aims at removing redundancies from algorithms. Most ADIO specifications are the original algorithmic specifications. A multi-coordinate systems (MCS) is used to present an algorithm's dependence structures. In a

MCS system, the index spaces of the variables in an algorithm are defined relative to their own coordinate systems. Most traditionally considered irregular algorithms present regular dependence structures under MCS technique. Procedures are provided for transforming algorithms from original algorithmic specifications to their regular specifications.

Multi-rate schedules and multi-rate timing functions are studied. The solution for multi-rate timing functions can be formulated as linear programming problems. Procedures are provided for mapping ADIOs onto multi-rate VLSI systems. Examples are provided to illustrate the synthesis of MRAs from DAREs and ADIOs.

The first major contribution of this dissertation is the development of the concrete, executable MRA architectures. The second is the introduction of MCS system and its application in the development of the theory for synthesizing MRAs from original algorithmic specifications.

# Mapping of Recursive Algorithms onto Multi-Rate Arrays

**By**

**Yue-Peng Zheng**

A Dissertation
Submitted to
Oregon State University

In partial fulfillment of
the requirements for the
degree of
Doctor of Philosophy

in

Electrical and Computer Engineering

Completed May 27, 1994
Commencement June 1995

APPROVED:

Redacted for Privacy

Associate 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 May 27, 1994

Typed by Yue-Peng Zheng for Yue-Peng Zheng

# CONTENTS

# List of Figures

# MAPPING OF RECURSIVE ALGORITHMS ONTO MULTI-RATE ARRAYS

## Chapter 1 Introduction

The first electronic computer ENIAC had a peak performance of about $10^3$ floating -point operations per second (flops) in 1946. Today, there are supercomputers capable of delivering hundreds of Giga-flops peak performance. Yet the demand for higher performance supercomputers continues to grow for applications such as weather forecasting, engineering, material science, plasma physics, economics, national defense, etc. [Mol93]. Performance is measured in two ways: peak performance and delivered performance. The second aspect is a more important criterion because most of today's supercomputers deliver only 5 to 15 percent of their peak performances to the users, except when hand optimization and assembly language programming are used on well suited programs [Kuc87, Mol93]. Higher performance computing can be achieved by using application specific computers that utilize Very-Large-Scale-Integrated (VLSI) circuit technology.

Such application-specific computers are the key components for many hard real-time applications such as in real-time speech and image codings, data communications, and robotic vision. Low design cost is an important criterion for application specific computers which requires automatic mapping of algorithms onto VLSI systems.

The systolic arrays shown in Figure 1.1 consist an important class of single rate arrays. The most distinguishing feature of the systolic architecture is that all operations in a system are controlled by a single clock signal. In other words, all the operations are per-

formed at the same rate throughout the entire array, regardless of the different complexities of these operations.

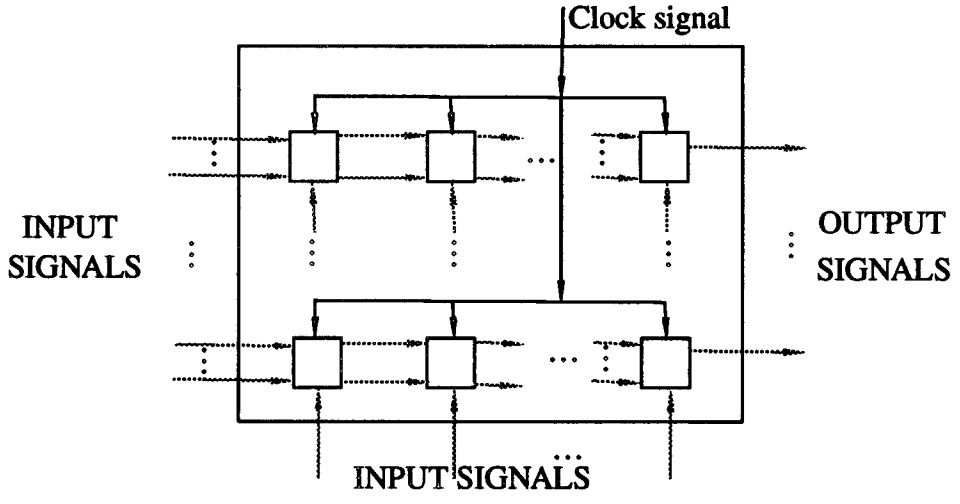



**Figure 1.1:** A *Single Rate Array* where the all operations of the array are controlled by a single clock signal.

---

Systolic and other application-specific arrays are well suited for matrix manipulations, filtering applications, and image codings, etc. Examples are FIR filters $y(i) = \sum_j h(j)\, x(i-j)$; and matrix multiplications $c(i,j) = \sum_{1 \le k \le n} a(i,k)\, b(k,j)$; etc. The systolic arrays can be a part of a larger supercomputing system shown in Figure 1.2.

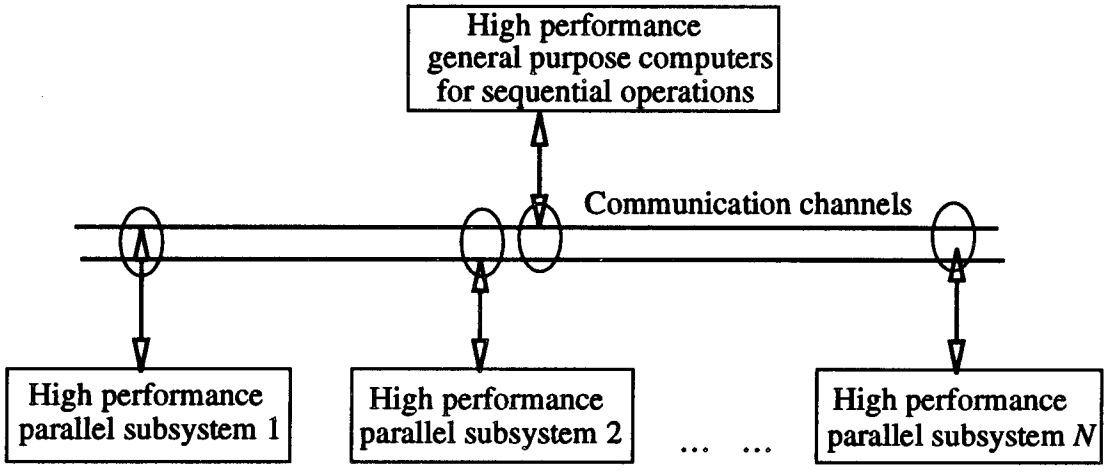



**Figure 1.2:** A possible supercomputer architecture consists of various application specific subsystems, each capable of delivering hundreds of Gflops.

Systolic array is a special parallel structure targeted for many applications in digital signal processing and matrix computation algorithms. Many synthesis tools are available for automatic mapping of algorithms onto systolic arrays. However, systolic arrays have limited scope of applications. In order to retain spatial regularity for VLSI implementations, redundancy often is introduced. Moreover, many algorithms cannot be synthesized onto systolic arrays using existing synthesis theories because they have irregular data dependence structures. For such algorithms, deriving a regular VLSI array is not a trivial task in any sense, nor is the task of performing design optimizations.

There exists a strong demand for developing a general synthesis method for the automatic mapping of a larger class of algorithms onto regular VLSI arrays. The ultimate objective is to synthesize algorithms from their original algorithmic specifications instead

of from their lower level specifications such as uniform recurrence equations (URE), affine recurrence equations (ARE), or regular iterative algorithms (RIA).

## 1.1 AN OVERVIEW OF THE DISSERTATION

This dissertation is devoted to the developments of multi-rate array structures and a synthesis theory for the automatic derivation of MRA architectures from the given algorithmic specifications.

Chapter 2 shows the Multi-Rate Array (MRA) as the target architecture for the growing class of algorithms. The spatial regularity requirement is indispensable for effective VLSI implementations, however, the temporal locality restriction is not necessary. The MRA concept is motivated by the fact that different operations may consume different amounts of time in a hardware system. By allocating different clocks to different operations, the efficiency of a system can be improved. The operations of an MRA architecture are controlled by multiple clock signals, as shown in Figure 1.3.

The ultimate objective in VLSI synthesis is to start the procedure from the original algorithmic specifications. The motivations are that algorithmic level optimization is the most important and effective step in application-specific VLSI system design. The selection of algorithms is an important decision affecting the degree of parallelism and the efficiency of parallel VLSI implementations. Also the ability to explore the full solution space of a given algorithm is necessary for architectural design optimization. Architectural optimization is also important for VLSI implementations.

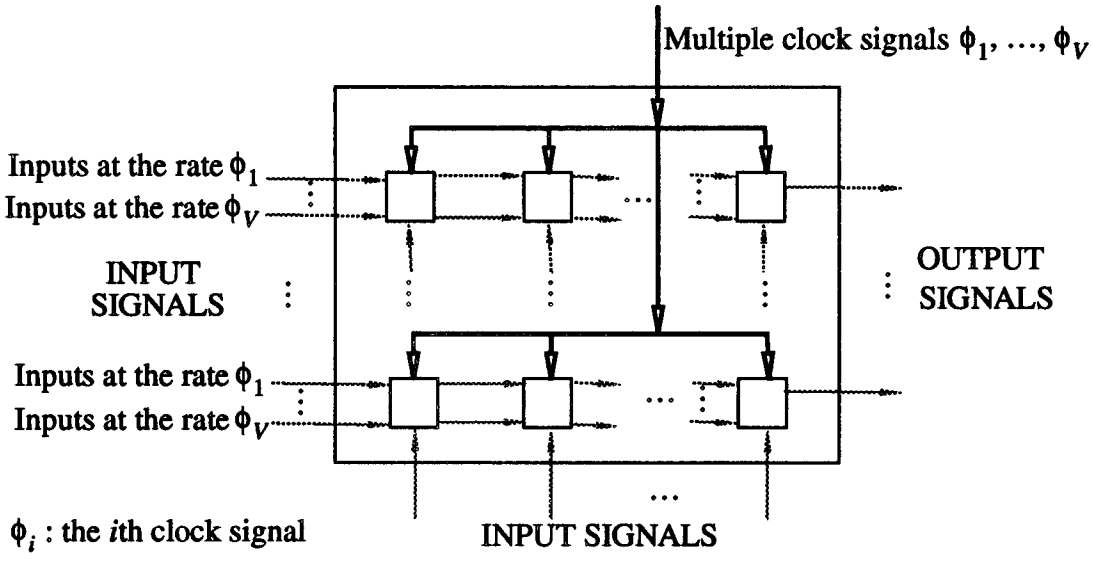



**Figure 1.3:** A *Multi-Rate Array* architecture where different operations are assigned with different amounts of time by the use of different clock signals.

---

Chapter 3 is concerned with data dependence localizations. It defines a new class of algorithms, affine direct input output (ADIO), to be studied and synthesized. Section 3.2 introduces the multi-coordinate systems (MCS). Definitions and the characterizations of MCS system will be provided. Under an MCS system, the index space of each variable may be defined relative to its own coordinate systems. The dependence structure of a given algorithm in general is not specified unless its index spaces have been defined. When a system of multi-coordinate systems is constructed for a given algorithm, the dependence structure of the algorithm has been specified. The uses of a single coordinate system for data dependence presentation and the use of a MCS system for dependence structure presentation of algorithms, in some sense, can be compared with the use of scalars and the use of vectors in number presentations.

Data dependence structure of ADIOs is investigated in section 3.3. The most important dependence properties of ADIOs are identified. By MCS technique, the right null space can be used effectively for dependence localizations. No other directional propagations will be required for most algorithms. Procedures will be provided for the construction of multi-coordinate systems for a given algorithm. Necessary and sufficient conditions will be provided in section 3.4. These conditions can be used to determine if a given ADIO can be transformed to regular specifications using null-space propagation under a MCS system. Section 3.5 provides a procedure for partitioning the computation domain for a variable whose index space must be defined in more than one coordinate system. In section 3.6, procedures are provided for converting ADIOs into their corresponding regular specifications. The chapter is concluded in section 3.7.

Chapter 4 deals with the scheduling issues for multi-rate VLSI systems. For the multi-rate array synthesis, it is necessary to derive a set of multi-rate timing functions from a given algorithm. This chapter defines and characterizes multi-rate schedules and multi-rate timing functions. It provides a procedure for constructing such schedules for a given algorithm. The derivation of a set of multi-rate schedules can be formulated as a linear programming problem.

In chapter 4, we also provide procedures for mapping ADIOs onto VLSI multi-rate systems. A procedure for the synthesis of ADIOs without the phase components will be provided first. Examples will show how the procedure works. The procedure is extended to allow the synthesis of ADIOs with the phase components. Again, examples will be studied.

Chapter 5 concludes the dissertation and points out some open problems.

## 1.2 LITERATURE REVIEW

The design of parallel computers can be traced back to the works of Unger [Ung58], Slotnick, et la [Slo62], and Golay [Gol65] for image processing applications. With the development of VLSI technology, parallel computer design has become an intensive research area since the late seventies. Processor regularity and nearest neighboring interconnection property are among the most important properties for effective VLSI implementations. Iterative logical arrays [Hen61, Hen68, Wai67] and cellular logic arrays [Duf78, Bat80] are two types of cellular automata [Bur70, Pre84]. Cellular automata are structurally iterative and regular which suit them for VLSI implementations. Two other important classes of cellular automata are systolic arrays [KL78, Kun82] and wavefront arrays [Kun84].

Since their introduction in the late seventies and early eighties, the design and synthesis of systolic/wavefront arrays has been one of the most active research area in the design of application-specific parallel computers, due to their structural regularity and local interconnection properties. Structural regularity can reduce the design cost while neighboring interconnections can improve system performance, especially in submicron VLSI systems where communication delays become the dominant factor.

Most of the early systolic architectures were designed by heuristic approaches. Such an *ad hoc* design approach is slow and error prone. It may require extensive simulations and the resulting designs are not guaranteed to be correct or optimal. Methodology was needed to reduce the design time and to guarantee the resulting systolic array is correct by construction. In the period of 1983 to 1986, much work was done on developing synthesis theories for automatically mapping systolic algorithms onto systolic architectures. The computational structures of systems of uniform recurrence equations (URE) defined by Karp, Miller and Winograd [KMW67] map extremely well onto systolic architectures,

This was presented by Quinton [Qui83, Qui84] and Chen and Mead [CM85, Che86], among others. Quinton developed a synthesis method where algorithm is initially represented as a set of UREs. An affine timing function is first determined which satisfies causality constraints imposed by the partial ordering in computation of the URE, and then an affine allocation function is selected to map the problem onto a systolic array. At the same time, others such as Cappello and Steiglitz [CS83, CS84], Miranker and Winkler [MW84] examined the systems. They pursued the linear mapping of uniform algorithms into space-time domain. These early works have developed a complete synthesis procedure based on URE as the initial specification of an algorithm.

Due to the restrictions arising from the URE specification of the initial algorithm, there has been an increasing effort towards the synthesis from a more general class of algorithms expressed in terms of Affine Recurrence Equations (ARE) and Regular Iterative Algorithms (RIA). The methods developed by Fortes and Moldovan [FM85], Delosme and Ipsen [DI86a, DI86b], Rao [Rao85], Rajopadhye [Raj86], Yaacoby and Cappello [YC88, YC89] focus on the initial specification of algorithms from AREs. AREs are transformed to UREs by localizing the global dependencies using null-space and directional propagations [FM84, Raj86, Roy88, Bu90,], or to Quasi-URE (QURE) [YC88] when the index mapping matrix $D$ is the root of $I$, that is $D^L = I$, where $I$ denotes the identity matrix. However, as it was pointed out in [RTRK88], the null space propagation technique works only under the condition that the index mapping matrix $D$ is idempotent, i.e., $D^2 = D$. Otherwise, multiple directional propagation technique must be employed to propagate the variable first along a direction not in parallel with the range space of the index mapping matrix. Then the variable can be propagated along the null space. This multiple directional propagation localization technique works, but it suffers from low efficiency in the sense that data must be propagated multiple times and through those index points where it is not used.

Efficiency has been one of the most important issues in the design of parallel computers. There has been a great deal of work on the systolic design optimizations [BK81, FP84, LW85, Rao85, DI85, CM85, Che86, LK88, SF88, WD85, WD89, WD92, BR90, Cap89, Cap92, SC92, ZK94]. These works all contribute to the design optimization of systolic arrays.

From the algorithmic point of view, the initial specifications of the above mentioned synthesis theories, the algorithms must be transformed from their original algorithmic specifications to ARE or RIA by *ad hoc* methods. To fully explore the parallel properties of algorithms for optimal solutions, it is necessary to start the synthesis procedure at higher level specifications. Weak Single Assignment Codes (WSAC) and WSAC-like specifications, proposed by Roychowdhury *et al* [RTRK88] and Rosseel *et al* [RCM92], allow the use of global operators such as summation $\Sigma$ and product $\Pi$, in the initial specification of algorithms. However, additional communication time and interconnections are required for such a dependence localization technique, which would degrades the performance of a system. Moreover, these works mainly focus on the synthesis of algorithms whose index mapping matrices are totally unimodular.

The advances in digital signal and image processing techniques in recent years have resulted in the wide use of multi-rate signal processing systems [CR83, Mal89, Vai90]. Sampling rate conversions (up- and down-sampling) are the key components in such systems. They can be found in applications such as subband speech coding, subband image coding, and wavelet analysis systems. Index mapping matrices in such applications are generally not idempotent nor unimodular. They do not have regular data dependencies as viewed traditionally. Temporal/spatial locality may not be achieved in such applications and existing synthesis theories cannot be employed to synthesize these algorithms onto regular VLSI array architectures.

In those earlier works done by many researchers, "all variables in the algorithm must have the same set of indices" as specified explicitly in [Rao85, p.272] or implicitly in [KMW67, Qui84, Qui89, Raj86, RTRK88, etc.]. In recent years, different index spaces for different variables have been used by some researchers [MQRS90]. However, the use of a coordinate system has been natural and has never become an issue in the development of synthesis theories. But in all research works published to date, the use of an implicit coordinate system is apparent in the definition of index spaces and computational domains.

The most serious problem for the development of synthesis theories for multi-rate (systolic) array architectures is the lack of a concrete multi-rate (systolic) array model and an executable MRA architecture which could show how such arrays might work. The concept of multi-rate (systolic) array and the motivation for developing such architectures are easily understood. However, how to design such arrays was not well understood. Without a concrete target architecture and an understanding of how such arrays work, all efforts would be in vain for developing synthesis theories for multi-rate (systolic) arrays.

The derivation of schedules for array processors has been an important issue in parallel computer design. Many works have been done on the scheduling of uniform algorithms onto systolic arrays [Qui83], [Qui84], [FP84], [DI86], [SF88], [Roy88], [YC89], [WD92], among others. However, these works are all concerned with the design of single rate schedules (for single rate arrays) and can not be employed for the determination of multi-rate schedules. For the design of multi-rate arrays, it is necessary to develop multi-rate schedules.

# Chapter 2 Multi-Rate Array and Directional Affine Recurrence Equations

Many algorithms do not have uniform dependencies and are linear functions of the indices. In this case, temporal/spatial locality may not be achieved for such algorithms. By relaxing the locality criteria and allowing data to propagate in multi-rate fashion, these non-uniform data dependencies can be propagated without hampering the regularity and pipeline structure of the array. A class of algorithms termed Directional Affine Recurrence Equation (DARE) is defined. The dependence structure of DAREs can be decomposed into uniform and non-uniform subspaces. When projected along the non-uniform subspace into the uniform subspace, the resultant array has regular structure.

## 2.1 NOTATIONS AND DEFINITIONS

This section provides the notations and definitions needed to make this dissertation self contained. New notations and definitions are to be introduced when they are needed in the following chapters.

- The Euclidean $n$-space is denoted by $R^n$, the set of integers in $R^n$ is denoted by $Z^n$, and the set of rational numbers in $R^n$ is denoted as $Q^n$. An integer lattice in $Z^n$ is denoted by $L^n$. $p, q \in Z^n$ denote the index points.
- $U_i$, $U_j$ denote variables in an algorithm.
- $\Omega_i$ denotes the computational domain of variable $U_i$.
- $\Gamma_i(p)$ denotes the index mapping function of the computed variable $U_i$ (on the left hand side of an equation). The index mapping functions for the input variables (on the right hand side of an equation) are denoted as $\Gamma_{ji}(p)$, $1 \le j \le V$. A variable $U_j$ may have more than one index mapping functions in the $i$-th equation $(1 \le i \le s)$. In this case, subscripts separated by coma will be used to distinguish these index mapping functions as $\Gamma_{ji,1}(p)$, $\Gamma_{ji,2}(p)$.

- $D_i \in Z^{m \times n}, d_i \in Z^m, 1 \le m \le n$, denote the linear part and the translation part of the affine index mapping function $\Gamma_i(p)$ for variable $U_i$ (in the $i$-th equation).
- $D_{ji} \in Z^{m \times n}, d_{ji} \in Z^m, 1 \le m \le n$ denote the linear part and the translation part for index mapping function $\Gamma_{ji}(p), 1 \le j \le s$. If $U_j$ has more than one index mapping function with different index mapping matrices in the $i$-th equation, subscripts separated by comma will be employed to distinguish them as: $D_{ji,1}, D_{ji,2}$.
- $I$ denotes the identity matrix of appropriate dimensions.
- $\aleph(D)$ denotes the null space of matrix $D$, and $\Re(D)$ denotes the range space of $D$,
- $U_i(p) \leftarrow U_j(q)$ denotes that $U_i(p)$ depends on $U_j(q)$, i.e., the evaluation of $U_i(p)$ uses the value of $U_j(q)$.

**Definition 2.1:** *Recurrence Equations*: A recurrence equation over a computational domain $\Omega$ is an equation defined as:

$$U(p) = f[\ldots, U(\Gamma(p)), \ldots] \tag{2.1}$$

where $p, \Gamma(p) \in \Omega$, $f$ is a single valued function which is strictly dependent on its arguments. A system of $s$-recurrence equations (SRE) over $\Omega$ is a family of $s$-REs defined as:

$$U_i(p) = f_i[\ldots, U_i(\Gamma_{ii}(p)), U_{ji}(\Gamma_j(p)), \ldots], 1 \le i, j \le s \tag{2.2}$$

**Definition 2.2:** *Affine Recurrence Equations*: An RE as defined in Eq.(2.1) is called an Affine Recurrence Equation (ARE) if all index mapping functions in it are affine functions, *i.e.* $\Gamma_{ji}(p) = D_{ji}p + d_{ji}; 1 \le j \le s$, where $D_{ji} \in Z^{m_j \times n}, d_{ji} \in Z^{m_j}, 1 \le j \le k, 1 \le m_j \le n$. If $\forall i, 1 \le i \le s$, Eq.(2.2) are AREs, then the SRE is called a system of affine recurrence equations (SARE). Moreover, if $\forall (i,j), D_{ji} = I$, then the system of equations specified becomes a system of uniform recurrence equations (URE).

Graph theory has been widely used for defining algorithms and also as a tool for the analysis of algorithms' dependence structures. For computability analysis and scheduling

of algorithms onto arrays, algorithms represented as Directed Acyclic Graphs allow us to see their dependencies more clearly. Therefore, they can help us in deriving better array structures.

**Definition 2.3:** *Directed Acyclic Graph*: The Directed Acyclic Graph (DAG) of an SRE is a directed graph defined by [*N, A*] where *N* is a set of nodes and *A* is a set of arcs. A pair of nodes, $n_1, n_2 \in N$, are connected by an arc, $a \in A$, from $n_1$ to $n_2$ if and only if the computation at $n_2$ uses the value from $n_1$. The DAG is related to a SRE so that the nodes are $N = \{a_i(p) \mid a_i(p)$ *is in SRE,* $p \in C_i\}$ and arcs *A* represent dependencies where $A = \{n_i(p) \leftarrow n_j(q) \mid q = M_{ij}(p)$*; are dependencies in SRE*$\}$.

## 2.2 MOTIVATION

**Example[2.1]:** Decimation and interpolation filters are the most important components in multi-rate digital signal processing applications [Cro83, Mal89, Vai90, Che93, Her93, Pri93]. An *N*-th order decimation filter with decimation factor *M* is defined as:

$$y(i) = \sum_{j=0}^{N-1} h(j)\, x(Mi - j) \tag{2.3}$$

where $h(j)$'s are the filter's coefficients and $x(i)$'s are the filter's inputs. The above can be expressed as an SARE as follows:

$$\begin{cases} For\ (0 \le i, 0 \le j \le N-1) \\ \quad y(i,j) = y(i,j-1) + h(j)\, x(Mi - j) \\ \quad y(i,-1) = 0 \end{cases} \tag{2.4}$$

The index point is $p = [i\ j]^T \in Z^2$ and the index mapping functions and index mapping matrices are:

$$\Gamma_y(p) = \begin{bmatrix} i \\ j-1 \end{bmatrix} \Rightarrow D_y = I, d_y = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$$

$$\Gamma_h(p) = \begin{bmatrix} 0 \\ j \end{bmatrix} \Rightarrow D_h = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \tag{2.5}$$

$$\Gamma_x(p) = \begin{bmatrix} Mi-j \\ 0 \end{bmatrix} \Rightarrow D_x = \begin{bmatrix} M & -1 \\ 0 & 0 \end{bmatrix}$$

The dependencies of variable $y$ $(\Gamma_y)$ are uniform, but dependencies of variables $h$ $(\Gamma_h)$ and $x$ $(\Gamma_x)$ are not uniform. Since $D_h$ is idempotent $(D_h^2 = D_h)$, these global dependencies of variable $h$ can be removed by propagating the $h$-variables along the null space of $D_h$, $\aleph(D_h) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$. However, the index mapping matrix $D_x$ is not idempotent. Conventional null space propagation technique cannot be used to localize these global dependencies [RTRK88]. The localization technique proposed by [RTRK88] for this class of algorithms is the multiple directional propagation technique. It first propagates the elements of $x$ along a direction $\eta$ which is not in the range space of $D$, i.e., $\eta \notin \Re(D)$, then it propagates these elements along another subspace. However, this technique does not yield a satisfactory solution here, since the length of propagation along $\eta$ is a linear function of the indices and increases with the indices in $\Re(D)$. Moreover, this technique increases the number of interconnections among processors, which are expensive in VLSI implementations. This SARE is not convertible to QURE $(D^L \neq I)$. Therefore, existing synthesis techniques are insufficient for deriving regular specifications from this class of algorithms. Existing synthesis theories are mostly concerned with the synthesis of algorithms with totally unimodular $(det(D) = \pm 1)$ index mapping matrices.

For sake of simplicity, let $N$=10 and $M$=3. Moreover, assume that those non-uniform dependencies associated with variable $h$ have already been localized by employing existing localization techniques. Then, Eq.(2.4) can be further specified as the following SARE:

$$
\begin{aligned}
&For\ (0 \le i, 0 \le j \le 9) \\
&\quad y(i,j) = y(i,j-1) + h(i,j)\,x(3i-j) \\
&\quad h(i,j) = h(i-1,j) \\
&\quad h(0,j) = h(j) \\
&\quad y(i,-1) = 0
\end{aligned}
\tag{2.6}
$$

The DAG for Eq.(2.6) is given in Figure 2.1. The global dependencies due to the broadcasting of $x_i$'s can not be removed by the use of conventional localization techniques since the index mapping matrix $D_x$ is not idempotent. Apparently, it is impossible to derive a regular array architecture from this DAG.

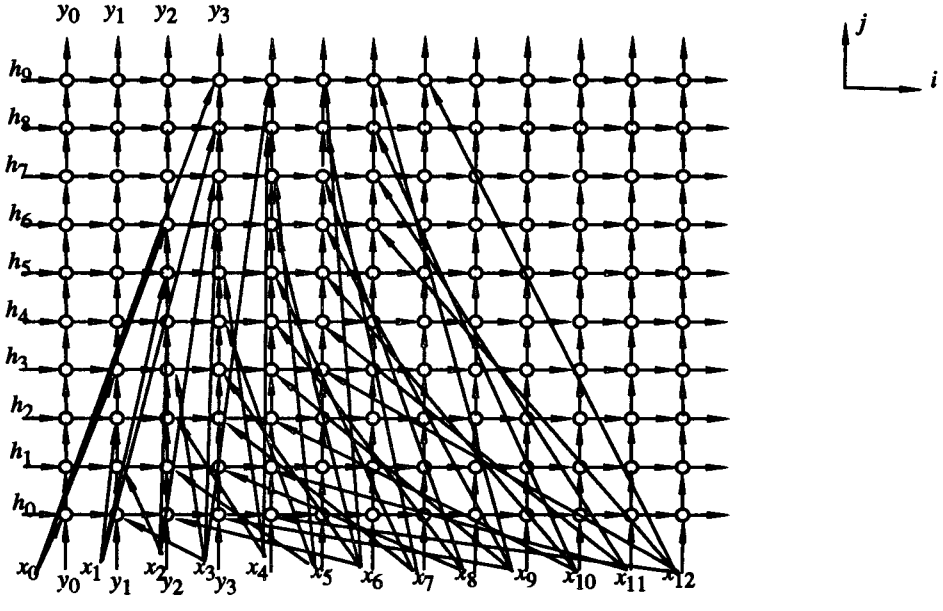



**Figure 2.1:** The DAG of the 10th order decimation filter with $M = 3$.

## 2.3 MULTI-RATE ARRAYS AND DIRECTIONAL AFFINE RECURRENCE EQUATIONS

**MRA:** A Multi-Rate Array is a grid of array processors where interconnections among processors may have different clock rates. In other words, each variable may be propagated at different clock rates.

**Definition 2.4:** A multi-rate array (MRA) is a grid of array processors characterized by the sets $[P, \Phi, U, D_p, N_l, F]$ where:

- $P$ is the processor space which is the set of all lattice points enclosed within a specified region in a *P-dimensional* Euclidean space. Each processor in this space may have more than one I/O link connecting it to other processors.
- $\Phi = \{\phi_i\}$ is the set of clock signals associated with each link $i$ where the clock frequencies could be different for different variables. If the set has only one element then the MRA is a systolic array.
- $U = \{U_1, U_2, ....U_V\}$ is the set of $V$ variables processed at a given time.
- $D_p$ is the set of processor displacements that defines the interconnection of the processor array. If $d \in D_p$ an interconnection exists between processor $p$ and $(p+d)$ irrespective of the particular value of $p$, except possibly at the boundary processors.
- $N_l$ is the number of delays in each processor associated with each link $l$ connecting processor $p$ to $(p + d)$. If $n \in N_l$, then a variable produced at processor $p$ will be passed to processor $(p+d)$ after $n$ clock beats (clock rate for each link is different), irrespective of the particular value of $p$ except possibly at the boundary processors.
- $F$ is the set of functional dependencies that relate the computation of a variable $x$ at processor $p$ during beat $\phi_x$ as a function of the variables computed during the previous beat at the neighboring processors.

The uniqueness of MRA architecture is that the temporal locality constraint does not exist and each data link may have its own clock frequency. Figure 2.2 shows a possible one-dimensional MRA architecture.

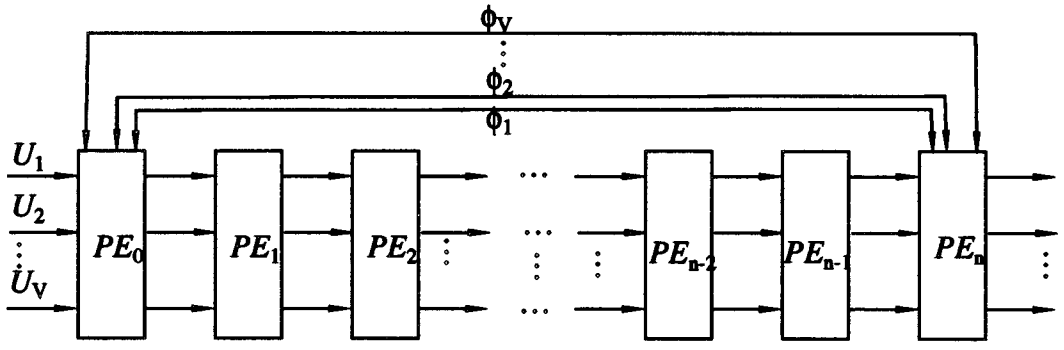



**Figure 2.2:** A possible one-dimensional MRA architecture with $V$ variables.

---

**Definition 2.5:** *Directional Affine Recurrence Equations* (DARE): A system of Directional Affine Recurrence Equations is a system of ARE whose index mapping matrices $D$ are such that $[D \pm I]$ is singular (where $I$ is the identity matrix). The rank of the matrix $[D \pm I]$, $m = \rho(D - I)$ is the dimension of the DARE, and the DARE is said to be an $m$-dimensional DARE.

A system of DARE is a system of ARE whose data dependencies can be decomposed into uniform and nonuniform components. An $m$-dimensional DARE implies that the dependence structure of this DARE has an $m$-dimensional non-uniform subspace and an $(n - m)$-dimensional uniform subspace. For a given 1-dimensional DARE, there exists a unique vector $\mu$. By projecting the dependence graph along this direction into an $(n - 1)$-dimensional subspace (hyperplane), the resultant array will have constant interconnections. However, projecting the dependence structure along any other direction results in an array architecture with varying interconnection lengths.

It should be noted that UREs, singular AREs, and QUREs are special cases of DAREs, because for UREs, singular ARE's and QURE the eigenvalues of the dependency

matrix $D$ are zero or $\pm 1$. However, for DARE only one of the eigenvalues of $D$ must be zero or $\pm 1$ and the remaining eigenvalues could be any integer number defining the multi-rate structure of the dependencies. For convenience, let $A = D - I$ and $T\{p,q\}$ denote the dependence vector for $U_i(p) \leftarrow U_j(\Gamma_j(p))$.

In this chapter, we will mainly investigate the properties of 1-dimensional DAREs, i.e., $D - I$ is rank one. Higher dimensional DAREs are simple generalizations of the 1-dimensional cases. The synthesis technique to be developed below can be modified to synthesize $m$-dimensional DAREs. For a given $m$-dimensional DARE, there exist $m$ projection vectors, $\{\mu_1, \mu_2, \ldots, \mu_m\}$, which can be used to project the dependencies $m$ times into the $(n-m)$-dimensional subspace with regular structures.

The decimation filter studied previously has a singular index mapping matrix $D_x = \begin{bmatrix} 3 & -1 \\ 0 & 0 \end{bmatrix}$. The global broadcasting dependency is due to the broadcasting of the variable $x$. It can be localized by propagating the variable $x$ along the $j$-axis, resulting in $D = \begin{bmatrix} 3 & -1 \\ 0 & 1 \end{bmatrix}$ with the translation vector $d = \begin{bmatrix} 0 & -3 \end{bmatrix}^T$. It is easy to show that $D - I$ is singular. Therefore it is a DARE.

### 2.3.1 Properties of DARE

This section studies the properties of one-dimensional DAREs and how to map DAREs onto the MRA structure based on these properties.

**Lemma 2.1:** For a given DARE, there exists a family of parallel hyperplanes $H_i$ such that the dependence vectors from hyperplane $H_i$ to a parallel hyperplane $H_j$ are constant, i.e., all dependence vectors $T\{p, q\}$ starting on nodes $p \in H_i$ are a constant vector and end on nodes $q \in H_j$.

**Proof.** The vector $T\{p,q\}$ is:

$$T = [Dp + d] - p = [D - I]p + d \tag{2.7}$$

Since *[D-I]* is singular and of rank one, its rows are linearly dependent and can be expressed as:

$$[D - I] = \begin{bmatrix} \alpha^T \\ K\alpha^T \end{bmatrix} = \mu\alpha^T \Rightarrow T = \mu\alpha^T p + d \tag{2.8}$$

where $\mu$ is a constant column vector. The product $\alpha^T p = C_i$ defines a *Uniform HyperPlane* (UHP) with constant dependencies in $n$-space. $\alpha^T$ is the normal direction of the hyperplanes with uniform dependencies $T = \mu C_i + d$, which is a *constant vector*. Therefore, all dependence vectors from hyperplane $H_i$ to $H_j$ are constant. ■

**Lemma 2.2:** Among the family of parallel UHPs, there is a *Symmetrical Hyperplane* $H_s$ on which the dependence vectors are invariant. This symmetrical UHP is defined as:

$$\{H_s | \alpha^T p = C_s = -\frac{\alpha^T d}{\alpha^T \mu}\} \text{ for } \alpha^T \mu \neq 0 \tag{2.9}$$

otherwise, $H_s$ is at infinity.

**Proof.** Assume there exist a symmetrical plane $H_s$. Any dependence vector $T_s$ on $H_s$ (i.e., both its starting and end points $p, q \in H_s$) should be orthogonal to $\alpha^T$, thus:

$$\alpha^T T_s = 0 \tag{2.10}$$

From Eqs.(3.5-6) we have:

$$\alpha^T T = \alpha^T \mu \alpha^T p + \alpha^T d \tag{2.11}$$

if $T \in H_s$, $\alpha^T p = C_s$, then

$$\alpha^T T = \alpha^T \mu C_s + \alpha^T d = 0 \Rightarrow C_s = -\frac{\alpha^T d}{\alpha^T \mu} \tag{2.12}$$

From Lemma.1 we know that the dependence vectors on a UHP are constant vectors, therefore, dependence vectors on $H_s$ are invariant. ■

From the invariant dependency property of $H_s$ we have the following theorem for the allocation function. It provides the condition for deriving an allocation function for a given DARE, which can map the DARE onto a VLSI array with constant length of interconnections.

**Theorem 2.1:** The processor space of a DARE belongs to the symmetrical hyperplane $H_s$, and there exists an affine allocation function *a(p)* which maps all the dependencies to this space domain, resulting in an array with constant interconnections.

**Proof.** The objective is to show that the projection of all dependencies onto this space is a constant vector. Let the affine allocation function be:

$$a(p) = \lambda_a p + \alpha_a \tag{2.13}$$

where $\alpha_a$ is a constant vector and $\lambda_a$ is the projection of dependency $T$ onto $H_s$ along the direction $\mu$:

$$\{\lambda_a | T \rightarrow H_s \, along \, \mu\} \tag{2.14}$$

from the definition of projection it follows that:

$$\lambda_a \mu = 0 \rightarrow \mu \in \aleph(\lambda_a)$$

Using Eqs.(3.5) and (3.6), we have:

$$\lambda_a T = (D - I)p = \lambda_a(\mu \alpha^T p + d) = \lambda_a \mu C_i + \lambda_a d = \lambda_a d \tag{2.15}$$

which is a constant interconnection vector in the space domain. ■

**Lemma 2.3:** For a given DARE the end points of the dependence vector $\{T_i: p \leftarrow q\}$ lie on the parallel hyperplanes defined as:

$$\alpha^T p = C_s + C_i,\ \alpha^T q = C_s + rC_i \tag{2.16}$$

where $r$ is a constant integer.

**Proof.** Since $q = \Gamma(p) = Dp + d$, then:

$$\alpha^T \Gamma(p) = \alpha^T (Dp + d) = \alpha^T ((D - I)p + p + d) \tag{2.17}$$

From (8-9) we have

$$\begin{aligned} \alpha^T \Gamma(p) &= \alpha^T [\mu \alpha^T p + d] + C_s + C_i = \alpha^T \mu (C_s + C_i) + \alpha^T d + C_s + C_i \\ \alpha^T \Gamma(p) &= \alpha^T \mu C_s + \alpha^T d + C_s + (1 + \alpha^T \mu) C_i \end{aligned} \tag{2.18}$$

Substituting $C_s$ from Eq.(2.9) results in:

$$\alpha^T \Gamma(p) = C_s + (1 + \alpha^T \mu) C_i = C_s + rC_i \rightarrow r = 1 + \alpha^T \mu \tag{2.19}$$

Therefore, the dependence vector *T[p,q]* has an index point on the hyperplane $\alpha^T p = C_s + C_i$. Its other index point $q = \Gamma(p)$ lies on the hyperplane parallel to $H_s$ separated by distance $rC_i$ as: $\alpha^T q = C_s + rC_i$. ■

For a given DARE, the dependence structure on either side of the symmetrical UHP $H_s$ are mirror-images and can be folded along $H_s$.

**Lemma 2.4:** For a given DARE with the dependence vectors on a UHP defined as $\alpha^T p = C_s + C_i$, there exist symmetrical dependencies (mirror images) that lie on the hyperplane defined as $\alpha^T p' = C_s - C_i$.

**Proof.** Assume $\alpha^T\mu \neq 0$, then there exists a symmetrical hyperplane $\{H_s: \alpha^T p = C_s\}$. The dependence vector $T_s$ on $H_s$ can be presented as:

$$T_s = q - p = Ap + d = \mu\alpha^T p + d = \mu C_s + d$$

$$\alpha^T T_s = \alpha^T \mu C_s + \alpha^T d = -\alpha^T d + \alpha^T d = 0$$

The dependence vector $T_s$ is perpendicular to $\alpha^T$ which is the normal direction of the hyperplane $H_s$. Furthermore, the dependency vector $T$ on the index point $p$ of the UHP defined as $\alpha^T p = C_s + C_i$ has its starting point $q$ on the UHP defined as $\alpha^T q = C_s + rC_i$. Thus we have a dependence vector $T^p$ defined as:

$$T^p = q - p = Ap + d = \mu\alpha^T p + d = \mu(C_s + C_i) + d \tag{2.20}$$

Correspondingly, there exists another dependence vector $T^n$ with an end point $p'$ on the hyperplane defined as $\alpha^T p' = C_s - C_i$, and its starting point $q'$ is on the hyperplane defined as $\alpha^T q' = C_s - rC_i$. Therefore, the dependence vector is presented as:

$$T^n = \mu(C_s - C_i) + d \tag{2.21}$$

The distance between the first two UHP's ($\alpha^T p = C_s + C_i$ and $\alpha^T q = C_s + rC_i$) is $dist(\alpha^T p - \alpha^T q)$ which is the same as the distance between the later two hyperplanes $\alpha^T p'$ and $\alpha^T q'$. If the vector $\alpha^T$ is used to project $T^p$ onto $H_s$, then the vector $-\alpha^T$ should be used to project $T^n$ onto $H_s$, since $T^p$ and $T^n$ are on the different sides of $H_s$. Therefore, the images of these vectors are presented below as:

$$\begin{aligned} \alpha^T T^p &= \alpha^T \mu(C_s + C_i) + \alpha^T d = \alpha^T \mu C_i \\ \alpha^T T^n &= -\alpha^T \mu(C_s - C_i) - \alpha^T d = \alpha^T \mu C \end{aligned} \tag{2.22}$$

Since the images of $T^p$ and $T^n$ vertical to are $H_s$ equal to each other (the distances of the hyperplanes of each pair) and the images on $H_s$ are also equal to each other, therefore, $T^n$ is the mirror image of $T^p$ and vice versa. ■

**Example[2.2]:** Consider a DARE with index mapping function $\Gamma(p) = Dp + d$ where $D = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$ and $d = \begin{bmatrix} 0 & -1 \end{bmatrix}^T$. The matrix $A = D - I = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ is singular. Decomposing $A$ as:

$$A = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \end{bmatrix} = \mu\alpha^T$$

This DARE has a DAG as shown in Figure 2.3. The symmetrical hyperplane is defined as $\{H_s | i + j = \frac{1}{2}\}$ and is shown in Figure 2.3 (along the direction of $\begin{bmatrix} -1 & 1 \end{bmatrix}^T$). The dependencies are invariant along this direction of $\begin{bmatrix} -1 & 1 \end{bmatrix}^T$. The only valid projection vector which can give us an array with constant interconnections is $\mu = \begin{bmatrix} 1 & 1 \end{bmatrix}^T$.

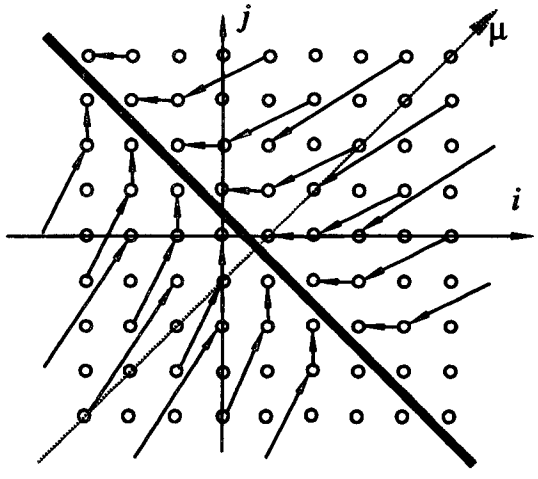



**Figure 2.3:** The DAG for example[2.2]. The dependencies are uniform only along the direction perpendicular to μ.

The projection vector $\mu$ and the symmetrical hyperplane $H_s$ in this example are perpendicular to each other. This may not be true in general, as will be shown in the following example. Under such circumstances, oblique projection may be used to project a DAG onto $H_s$.

**Example[2.3]:** Given a DARE $U_i(p) \leftarrow U_j(Dp+d)$, where $D = \begin{bmatrix} 3 & -1 \\ 0 & 1 \end{bmatrix}, d = \begin{bmatrix} 0 \\ -3 \end{bmatrix}$. This DARE has the following dependence matrix:

$$A = D - I = \begin{bmatrix} 2 & -1 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \begin{bmatrix} 2 & -1 \end{bmatrix} = \mu\alpha^T$$

The hyperplanes are defined by $2i - j = C_i + C_s$ and the symmetrical hyperplane is defined as $\{H_s | 2i - j = -3/2\}$. Figure 2.4 shows these hyperplanes (dotted lines) and the symmetrical hyperplane (wide heavy black line). The UHP's are parallel to each other. The dependence vectors from one hyperplane to another are constant vectors. These dependence vectors are symmetrical on both sides of $H_s$. From Thm.2.1, in order to obtain an array with constant interconnections, we need to use the vector $\mu = \begin{bmatrix} 1 & 0 \end{bmatrix}^T$ to project this DAG into the symmetrical hyperplane $H_s$. However, since such projections are not perpendicular, it may be easier to use orthogonal projections to obtain the allocation function.

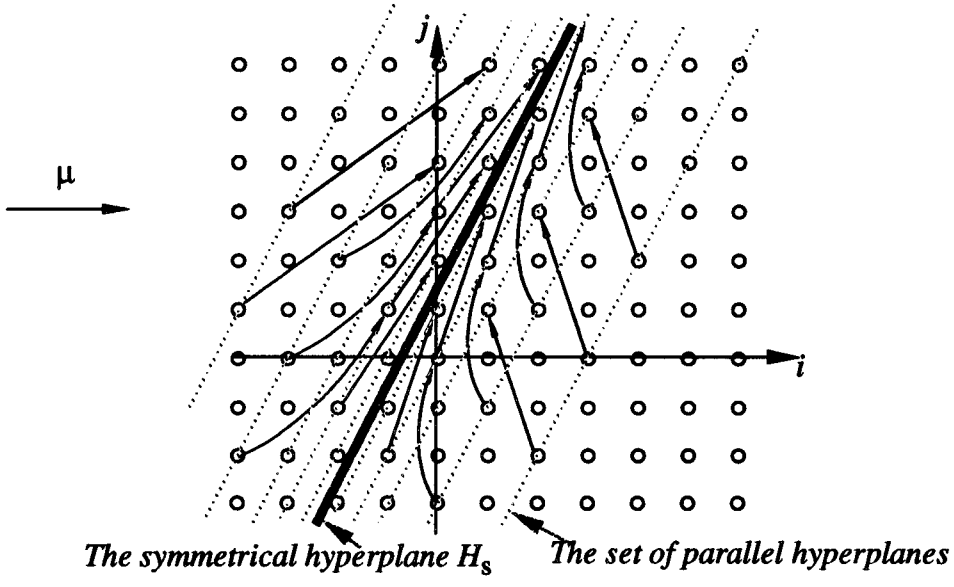



**Figure 2.4:** The family of parallel hyperplanes with uniform dependencies and the symmetrical hyperplane $H_s$ for Example[2.3].

---

**Theorem 2.2:** For a given DARE, a schedule for the multi-rate variable is a piecewise linear schedule divided into three regions by the symmetrical hyperplane $H_S$ as:

$$\begin{aligned} &\alpha^T p < C_s \Rightarrow \lambda_{t1} = \alpha^T \\ &\alpha^T p = C_s \Rightarrow \lambda_{t2} \perp \alpha^T \\ &\alpha^T p > C_s \Rightarrow \lambda_{t3} = -\alpha^T \end{aligned} \tag{2.23}$$

where the schedule function is $t(p) = \lambda_t p + \alpha_t$.

**Proof.** 1) For $\alpha^T p < C_s$:

Let $p$ be a node on the dependent hyperplane $\alpha^T p = C_i$. This node depends on $\Gamma(p) = Dp + d = (\mu\alpha^T + I)p + d$. A valid schedule for a recurrence equation must satisfy the causality requirement $\lambda_t p > \lambda_t \delta(p)$.

$$
\begin{aligned}
&\lambda_t p > \lambda_t \delta(p) \Rightarrow \\
&\alpha^T \Gamma(p) = \alpha^T(\mu\alpha^T + I)p + \alpha^T d \\
&\qquad = \alpha^T \mu \alpha^T p + \alpha^T p + \alpha^T d \\
&\qquad = \alpha^T C_i + C_i + \alpha^T d < \alpha^T p = C_i \\
&\Rightarrow \alpha^T \mu C_i + \alpha^T d < 0 \Rightarrow C_i < -\frac{\alpha^T d}{\alpha^T \mu} = C_s
\end{aligned}
\tag{2.24}
$$

Therefore $\alpha^T$ is a valid scheduling vector for the multi-rate variable for $C_i < C_s$.

2) For $\alpha^T p = C_s$:

In this case, both of the nodes $p$ and $q$ of the dependence vector are on the symmetrical hyperplane $\{H_s | \alpha^T p = C_s = \alpha^T \delta(p)\}$. Therefore, $\alpha^T$ can not be used as a schedule vector because $\lambda_t p > \lambda_t \delta(p)$. All other vectors which are not equal to $\alpha^T$ can be used as the schedule vector $\lambda_{t2}$. Here we choose $\lambda_{t2} \perp \alpha^T$.

3) For $\alpha^T p > C_s$:

The proof for this case is similar to the case of i). $-\alpha^T$ is qualified to be the schedule function for this case. ■

In general, there exist many vectors qualified as scheduling vectors. Therefore, the above theorem does not attempt to exclude other vectors to be used as scheduling vectors. The above selection is just a choice which reflects the computation ordering.

### 2.3.2 Mapping of DAREs onto MRA

***Procedure 2.1.*** Mapping DAREs on MRA architectures.

1. Represent the algorithm in terms of ARE and check if it is a DARE.
2. Extract the projection vector $\mu$ from the matrix $A = D - I = \mu\alpha^T$.

3. Compute the clock rate ratio $r$ as $r = \alpha^T\mu + 1$.

4. Base on the value of $r$ determine:

   *a)* $r > 1$. Use $\mu$ as the projection vector. Projecting the DAG onto the $(n-1)$ -dimensional space. The multi-rate schedule function has a ratio of $r$ in respect to the schedule function for the uniform dependencies.

   *b)* Since $r = 1$ implies $\alpha^T\mu = 0$. Thus, symmetrical hyperplane $H_s$ is at infinity. This is a URE.

   *c)* Since $r = 0 \Rightarrow det(D)=0$, use the null space propagation technique [Ray86].

   *d)* $r = -1$. The ARE can be made uniform by folding the computation domains along the symmetrical hyperplane $H_s$.

   *e)* $r < -1$. Fold the DAG along $H_s$ and convert it to an ARE with $r>1$.

**Example[2.4]:** *MRA implementation of the decimation filter:* Apply the synthesis procedure to the mapping of decimation filter studied in example[2.1] onto MRA array architecture. Let $N$=10 and $M$=3, as mentioned before. Propagating the variable $x$ along the $j$-axis, Eq.(2.6) can be transformed into the DARE specifications as defined below:

$$
\begin{aligned}
&For\ (0 \le i, 0 \le j \le 9) \\
&\quad y(i,j) = y(i,j-1) + h(i,j)\,x(3i-j, j-3) \\
&\quad h(i,j) = h(i-1,j) \\
&\quad h(0,j) = h(j) \\
&\quad x(i,j) = x(i,j-1) \\
&\quad x(i, j \le 0) = x(i) \\
&\quad y(i,-1) = 0
\end{aligned}
\tag{2.25}
$$

The computational domain of Eq.(2.25) is defined as $\Omega = \{[i\ j]^T \mid 0 \le i, 0 \le j \le 9\}$. The index mapping functions and matrices of this DARE are defined as:

$$\Gamma_y(p) = \begin{bmatrix} i \\ j-1 \end{bmatrix} \Rightarrow D_y = I, d_y = \begin{bmatrix} 0 \\ -1 \end{bmatrix}; \Gamma_h(p) = \begin{bmatrix} i-1 \\ j \end{bmatrix} \Rightarrow D_h = I, d_h = \begin{bmatrix} -1 \\ 0 \end{bmatrix}$$
$$\Gamma_x(p) = \begin{bmatrix} 3i-j \\ j-3 \end{bmatrix} \Rightarrow D_x = \begin{bmatrix} 3 & -1 \\ 0 & 1 \end{bmatrix}, d_x = \begin{bmatrix} 0 \\ -3 \end{bmatrix} \quad (2.26)$$

In Eq.(2.26), the only non-uniform index mapping function is $\Gamma_x(p)$, but $det[D_x - I] = 0$, therefore, the above synthesis procedure can be used for mapping this algorithm onto the MRA structure. Figure 2.5 shows the DAG of this algorithm. Since the $y$- and $h$-dependencies are uniform over the entire domain, we present them in gray lines in order to show the dependencies of variable $x$ clearly.

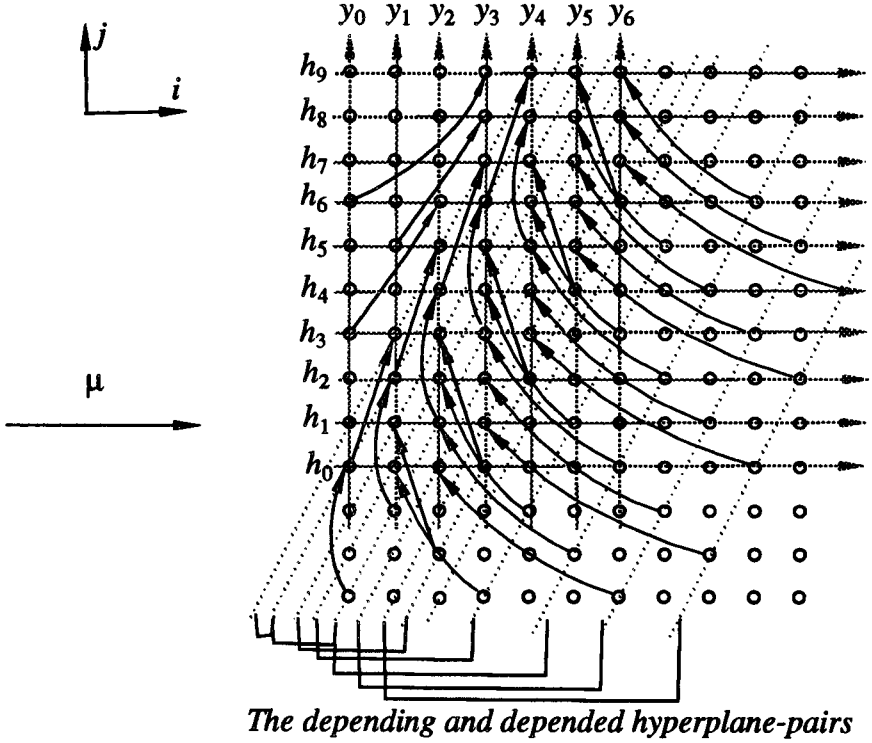


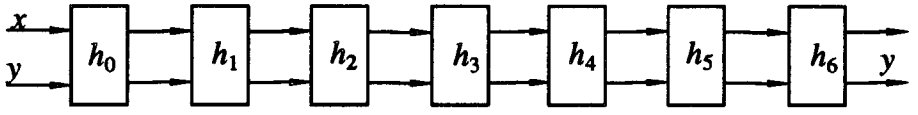

**Figure 2.5:** The DAG of the decimation filter with $N = 10$ and $M = 3$ as defined in Eq.(2.25).

Following the synthesis procedure presented above, we have:

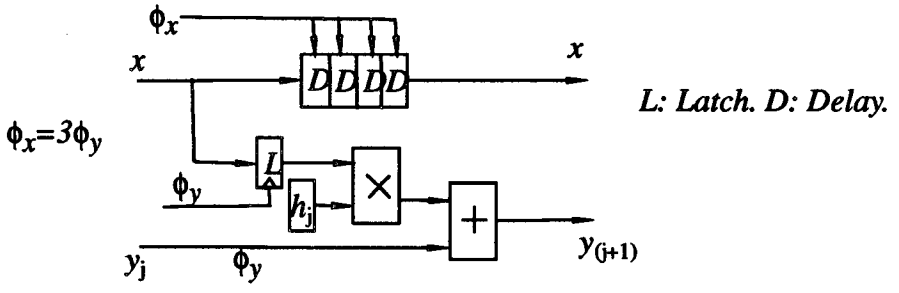
$$A = D_x - I = \begin{bmatrix} 2 & -1 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \begin{bmatrix} 2 & -1 \end{bmatrix} = \mu\alpha^T \tag{2.27}$$

Therefore, we have the allocation function $a(p) = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix}$, which give us a linear array structure as shown in Figure 2.6*a*. Now, let different variables travel at different clock rates and derive a structure for the algorithm. Let the clock signal for variables $y$ and $h$ have a rate $\phi_y$. The clock rate ratio is determined as $r = \alpha^T\mu + 1 = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} 2 \\ -1 \end{bmatrix} + 1 = 3$ for variable $x$. That is, the clock signal frequency for the variable $x$ should be three times the other variables' clock signal. Thus, we have $\phi_x = 3\phi_y$.

A good system design engineer would produce a linear array structure as shown in Figure 2.6a with internal structure as given in Figure 2.6b. The most significant difference between MRA architecture and single rate array structures is that in MRA array different operations may be allocated with different amounts of time. Complicated operations (e.g., multiplication-accumulation operations in this example) are always granted with more time to perform. Simpler operations (e.g., shifting data from register to register) are always traveling at faster clock rate. The amount of time allocated for performing the multiply-accumulate operation is three times the time given to the data shifting operations. This feature of MRA structure also will be illustrated later in the MRA implementation of an interpolation filter, a dual structure of the decimation filter.




*(a) Multi-rate array for the decimation filter*




*(b) Internal block diagram of the processing element*

**Figure 2.6:** An MRA implementation of the decimation filter. Two clock signals are employed.

---

The operation of the array is as follows. The input terminal $x$ (link-$x$) receives the input data from the signal source and propagates the data along link-$x$ at the clock rate $\phi_x$, while variable $y$ is computed at each PE and propagated along link-$y$ at the rate $\phi_y$. The filter's output is generated also at the rate $\phi_y$, where $\phi_x = 3\phi_y$. Such a design can improve the performance of the array since the time required for latching and shifting operations generally takes less time than $\tau_m$ or $\tau_a$ (multiplication and addition times). Assume that the shift registers and latches are rising edge triggered. Since the latch $L$ is clocked by $\phi_y$ and the frequency of $\phi_y$ is only one third of $\phi_x$, each PE captures only one third of the data propagating on link-$x$ for the computation. This MRA array, in effect, performs the decimation operation before the computation instead of post computation.

A VHDL model of this MRA architecture has been developed and simulated. Figure 2.7 shows the simulation output, the waveform of each signal. It shows only the waveform of the first four PEs due to the space restriction.

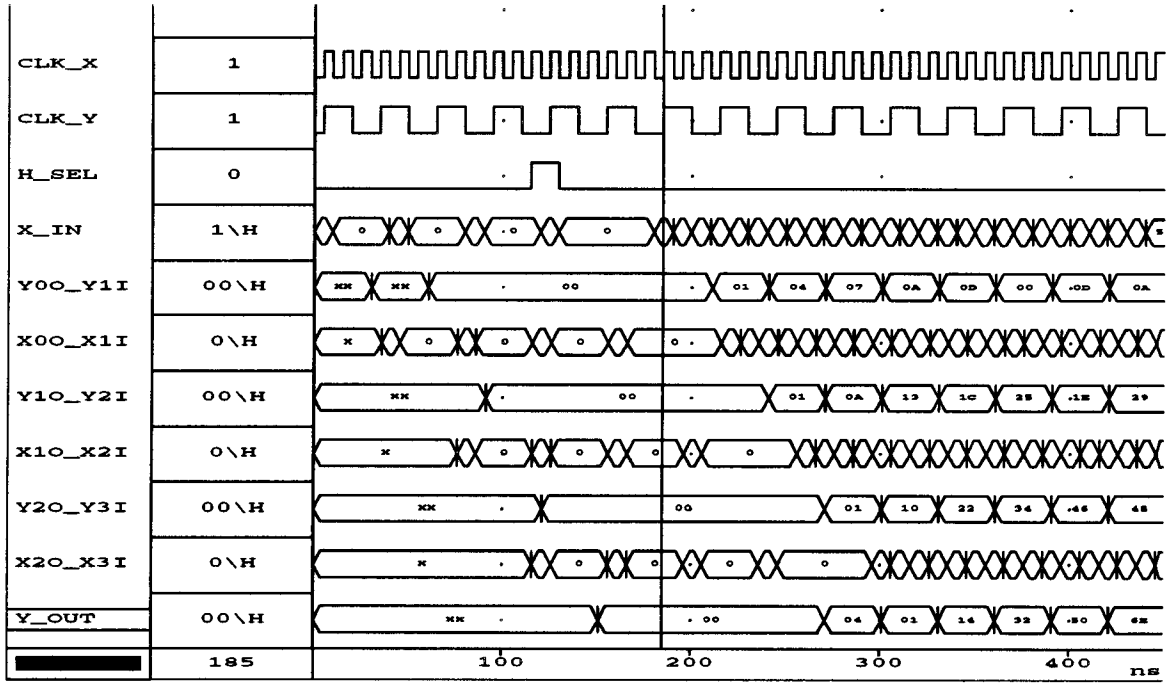



**Figure 2.7:** A VHDL simulation result of the MRA decimation filter shown in Figure 2.6.

The detailed operations of the first four PEs of the MRA array are given in the snapshots in Table 2.1. Again, only the operations of the first four PEs are given here due to space restriction.

**TABLE 2.1. Snap-shots of the operation of the MRA decimation filter.**

| $\phi_x$ | $\phi_y$ | *PE0* ($h_0$) | *PE1* ($h_1$) | *PE2* ($h_2$) | *PE3* ($h_3$) |
|---|---|---|---|---|---|
| 0 | 0 | $h_0x_0$ ($x_0$) | | | |
| 1 | | $h_0x_0$ ($x_1$ $x_0$) | | | |
| 2 | | $h_0x_0$ ($x_2$ $x_1$ $x_0$) | $h_0x_0$+0 | | |
| 3 | 1 | $h_0x_3$ ($x_3$ $x_2$ $x_1$ $x_0$) | $h_0x_0$+0 | | |
| 4 | | $h_0x_3$ ($x_4$ $x_3$ $x_2$ $x_1$) | $h_0x_0$ (x0) | | |
| 5 | | $h_0x_3$($x_5$ $x_4$ $x_3$ $x_2$) | $h_0x_0$ ($x_1$ $x_0$) | | |
| 6 | 2 | $h_0x_6$ ($x_6$ $x_5$ $x_4$ $x_3$) | $h_0x_3$+$h_1x_2$ ($x_2$ $x_1$ $x_0$) | $h_0x_0$ | |
| 7 | | $h_0x_6$ ($x_7$ $x_6$ $x_5$ $x_4$) | $h_0x_3$+$h_1x_2$ ($x_3$ $x_2$ $x_1$ $x_0$) | $h_0x_0$ | |
| 8 | | $h_0x_6$ ($x_8$ $x_7$ $x_6$ $x_5$) | $h_0x_3$+$h_1x_2$ ($x_4$ $x_3$ $x_2$ $x_1$) | $h_0x_0$ ($x_0$) | |
| 9 | 3 | $h_0x_9$ ($x_9$ $x_8$ $x_7$ $x_6$) | $h_0x_6$+$h_1x_5$ ($x_5$ $x_4$ $x_3$ $x_2$) | $h_0x_3$+$h_1x_2$+$h_2x_1$ ($x_1$ $x_0$) | $h_0x_0$ |
| 10 | | $h_0x_9$ ($x_{10}$ $x_9$ $x_8$ $x_7$) | $h_0x_6$+$h_1x_5$ ($x_6$ $x_5$ $x_4$ $x_3$) | $h_0x_3$+$h_1x_2$+$h_2x_1$ ($x_2$ $x_1$ $x_0$) | $h_0x_0$ |
| 11 | | $h_0x_9$ ($x_{11}$ $x_{10}$ $x_9$ $x_8$) | $h_0x_6$+$h_1x_5$ ($x_7$ $x_6$ $x_5$ $x_4$) | $h_0x_3$+$h_1x_2$+$h_2x_1$ ($x_3$ $x_2$ $x_1$ $x_0$) | $h_0x_0$ |
| 12 | 4 | $h_0x_{12}$ ($x_{12}$ $x_{11}$ $x_{10}$ $x_9$) | $h_0x_9$+$h_1x_8$ ($x_8$ $x_7$ $x_6$ $x_5$) | $h_0x_6$+$h_1x_5$+$h_2x_4$ ($x_4$ $x_3$ $x_2$ $x_1$) | $h_0x_3$+$h_1x_2$+$h_2x_1$ +$h_3x_0$ ($x_0$) |
| 13 | | $h_0x_{12}$ ($x_{13}$ $x_{12}$ $x_{11}$ $x_{10}$) | $h_0x_9$+$h_1x_8$ ($x_9$ $x_8$ $x_7$ $x_6$) | $h_0x_6$+$h_1x_5$+$h_2x_4$ ($x_5$ $x_4$ $x_3$ $x_2$) | $h_0x_3$+$h_1x_2$+$h_2x_1$ +$h_3x_0$ ($x_1$ $x_0$) |
| 14 | | $h_0x_{12}$ ($x_{14}$ $x_{13}$ $x_{12}$ $x_{11}$) | $h_0x_9$+$h_1x_8$ ($x_{10}$ $x_9$ $x_8$ $x_7$) | $h_0x_6$+$h_1x_5$+$h_2x_4$ ($x_6$ $x_5$ $x_4$ $x_3$) | $h_0x_3$+$h_1x_2$+$h_2x_1$ +$h_3x_0$ ($x_2$ $x_1$ $x_0$) |
| 15 | 5 | $h_0x_{15}$ ($x_{15}$ $x_{14}$ $x_{13}$ $x_{12}$) | $h_0x_{12}$+$h_1x_{11}$ ($x_{11}$ $x_{10}$ $x_9$ $x_8$) | $h_0x_9$+$h_1x_8$+$h_2x_7$ ($x_7$ $x_6$ $x_5$ $x_4$) | $h_0x_6$+$h_1x_5$+$h_2x_4$ +$h_3x_3$ ($x_3x_2x_1x_0$) |

The advantages of this MRA implementation are: (1) regular array structure with nearest neighboring interconnections; (2) high hardware utilization, the efficiency of this MRA decimation filter is one hundred percent, no computed result is compressed; (3) high throughput rate can be achieved; (4) high achievable speed, the speed of the MRA array, is in general not limited by the slowest operation.

## 2.4 PROPERTIES AND SYNTHESIS OF HIGHER DIMENSIONAL DARES

In the last section, we considered one-dimensional DAREs, i.e., $\rho(A) = 1$. In this section, we will generalize this study to algorithms with higher rank numbers, or $m$-dimensional DAREs (it is assumed that these algorithms themselves are $n$-dimensional). The dependence space of this class of algorithms can be decomposed into an $m$-dimensional non-uniform subspace and an $(n-m)$-dimensional uniform subspace. Let $m$ ($m<n$) be the rank of $A$, $m = \rho(A) = \rho(D-I)$. This means that there are only $m$ vectors in the matrix $A$ which are linearly independent. The other $n-m$ vectors are linearly dependent on these $m$ vectors. Therefore, we can decompose $A$ as:

$$A = \mu_1\alpha_1^T + \mu_2\alpha_2^T + \ldots + \mu_m\alpha_m^T \tag{2.28}$$

where all $\mu_i$'s and $\alpha_i$'s are linearly independent respectively. The dependence vector $T = (D-I)p + d$ can be expressed as:

$$T = \mu_1\alpha_1^T p + \mu_2\alpha_2^T p + \ldots + \mu_m\alpha_m^T p + d \tag{2.29}$$

**Lemma 2.5:** For a given $m$-dimensional DARE there exists a family of intersections of $(n-m)$-dimensional space $\hat{\mathbf{I}}$ formed by the $m$-sets of hyperplanes defined as:

$$\alpha_i^T p = C_{ij},\ i = 1, \ldots, m \tag{2.30}$$

Dependence vectors starting on one intersection $\hat{\mathbf{I}}_1$ have their end points on an parallel intersection $\hat{\mathbf{I}}_2$.

**Proof.** Since we know that the index point $p$ is on the intersection of the hyperplanes defined by Eq.(2.30), then $\forall i,\ 1 \le i \le m, \exists j \wedge C_{ij}$ such that index point $p$ satisfies all these equations of hyperplanes. Thus the dependence vector $T$ is an $n$-dimensional constant vector defined as:

$$T = \mu_1 C_{1j} + \mu_2 C_{2j} + \ldots + \mu_m C_{mj} + d \tag{2.31}$$

This means that all dependence vectors with their ending points on this intersection have their starting points on another intersection parallel to it. ■

**Lemma 2.6:** Among the family of parallel intersections, there exists a *Symmetrical Intersection* $\hat{\mathbf{I}}_s$ on which the dependence vectors are invariant. This symmetrical intersection is defined by the following hyperplanes:

$$H_s^i \Big| \alpha_i^T p = C_s^i = -\Big( \sum_{i \ne j,\ 1 \le j \le m} \alpha_i^T \mu_j \alpha_j^T p + \alpha_i^T d \Big) / (\alpha_i^T \mu_i) \tag{2.32}$$

for $\alpha_i^T \mu_i \ne 0;\quad 1 \le i \le m$, otherwise $\hat{\mathbf{I}}_s$ is at infinity.

**Proof.** Assume that there exist a symmetrical plane $\hat{\mathbf{I}}_s$. Let $T_s$ be a dependence vector on $\hat{\mathbf{I}}_s$, i.e., both its starting and end points $p, q \in \hat{\mathbf{I}}_s$. Moreover, since $p, q \in \hat{\mathbf{I}}_s \subseteq H_s^i,\ 1 \le i \le m$, therefore, any $T_s$ on $\hat{\mathbf{I}}_s$ should be orthogonal to all vectors $\alpha_i^T, \forall i,\ 1 \le i \le m$, that is:

$$\alpha_i^T T_s = 0, \forall i,\ 1 \le i \le m \tag{2.33}$$

From Eqs.(3.27-28), $\forall i,\ 1 \le i \le m$, we have:

$$\begin{aligned} \alpha_i^T T &= \alpha_i^T (\mu_1 \alpha_1^T p + \mu_2 \alpha_2^T p + \ldots + \mu_m \alpha_m^T p + d) \\ &= \alpha_i^T \mu_1 \alpha_1^T p + \alpha_i^T \mu_2 \alpha_2^T p + \ldots + \alpha_i^T \mu_m \alpha_m^T p + \alpha_i^T d \end{aligned} \tag{2.34}$$

if $T_s \in \hat{\mathbf{I}}_s \subseteq C_s^i,\ 1 \le i \le m$, it is orthogonal to all the hyperplanes $\alpha_i^T p = C_s^i$, therefore, we have the conclusion that:

$$\alpha_i^T T_s = \alpha_i^T \mu_i C_s^i + \sum_{j \neq i,\ 1 \le j \le m} \alpha_i^T \mu_j \alpha_j^T p + \alpha_i^T d = 0$$
$$\Rightarrow C_s^i = -\Big( \sum_{i \neq j,\ 1 \le j \le m} \alpha_i^T \mu_j \alpha_j^T p + \alpha_i^T d \Big) / (\alpha_i^T \mu_i) \tag{2.35}$$

From Lemma 3.5, we know that the dependence vectors on an intersection are constant vectors, therefore, dependence vectors on $\hat{\mathbf{I}}_s$ are invariant. ■

From the invariant dependence property of $\hat{\mathbf{I}}_s$, we conclude that the regular processor space of a given DARE belongs to $\hat{\mathbf{I}}_s$.

In order to implement such problems onto VLSI arrays with fixed length of interconnections, the allocation function $a(p) = \Lambda_a p + c_a$ must satisfy the condition $\Lambda_a T = C$, where $C$ is an $(n\text{-}m)$ dimensional column vector, $\Lambda_a$ is an $((n-m) \times n)$ matrix. Extending Thm.2.2 from 1-dimensional to $m$-dimensional, we have the following theorem for the solution of the allocation function.

**Theorem 2.3:** For a given $m$-dimensional DARE, its processor space belongs to the symmetrical interconnection $\hat{\mathbf{I}}_s$. There exists an affine allocation function $a(p)$ which maps all the dependencies to this space domain, resulting in an array with constant interconnections. Moreover, the linear part of the allocation function $\Lambda_a \in Z^{(n-m) \times n}$ must satisfy the conditions $\Lambda_a \mu_i = 0, \quad \forall i, 1 \le i \le m$.

**Proof.** The objective is to show that the projection of all dependencies onto this space is a constant vector. Let the affine allocation function be:

$$a(p) = \lambda_a p + \alpha_a \tag{2.36}$$

where $\alpha_a$ is a constant vector, $\lambda_a \in Z^{(n-m) \times n}$ is the allocation matrix. A dependence vector $T$ can be mapped onto $\hat{\mathbf{I}}_s$ by projecting it $m$ times along the directions $\mu_i$, $1 \le i \le m$. In order to map the dependence vector $T$ onto the $(n\text{-}m)$-

dimensional processor space as a constant interconnection, $\lambda_a \mu_i = 0$ must hold for $i$=1,2,...,$m$. This confirms that $\lambda_a$ lies in the null space of $\mu$. The rest of the proof is similar to that of Theorem 3.1. ■

The rank of $A$ plays an important role in the synthesis of MRAs based on DAREs. It determines the maximum possible dimension of a VLSI array with constant interconnecting links for a given problem.

The procedure for synthesizing a DARE is:

1) Determine $m = \rho[A]$ and the number of different clock signals.
2) Decompose the matrix $A$ as shown in Eq.(2.28).
3) Finding the allocation function $\lambda_a$
4) Find the proper timing function.

**Example[2.5]:** *Two-dimensional decimation filter.* A 2-dimensional decimation filter is defined as:

$$y(i,j) = \sum_{k=0}^{K-1} \sum_{l=0}^{L-1} h(k,l)\, x(M_1 i - k, M_2 j - l) \tag{2.37}$$

where $x(i,j)$'s and $y(i,j)$'s are the input and output signal arrays of dimensions $N_{x1} \times N_{x2}$ and $N_{y1} \times N_{y2}$ respectively; $h(k,l)$ is a $K \times L$ rectangular window array. $M_1$ and $M_2$ are the decimation factors. For the sake of simplicity, we will only consider the filter with parameters defined as: $M_1$=$M_2$=2 and $K$=$L$=4:

$$y(i,j) = \sum_{k=0}^{3} \sum_{l=0}^{3} h(k,l)\, x(2i - k, 2j - l) \tag{2.38}$$

We can express the above algorithm as a system of ARE as follows:

$$
\begin{aligned}
&For \quad 0 \le k \le 3 \\
&\quad For \quad 0 \le l \le 2 \\
&\quad\quad y(i,j,k,l) = y(i,j,k,l-1) + h(k,l)\, x(2i-k, 2j-l) \\
&\quad For \quad l = 3 \\
&\quad\quad y(i,j,k,l) = y(i,j,k,l-1) + y(i,j,k-1,l) + h(k,l)\, x(2i-k, 2j-l) \\
&\quad\quad y(i,j,k,-1) = 0 \\
&\quad\quad y(i,j,-1,3) = 0
\end{aligned}
\tag{2.39}
$$

By augmenting the lower dimension arrays $h$ and $x$ to the full dimension, we have the following system of DARE:

$$
\begin{aligned}
&For \quad 0 \le k \le 3 \\
&\quad For \quad 0 \le l \le 2 \\
&\quad\quad y(i,j,k,l) = y(i,j,k,l-1) + h(i,j,k,l)\, x(2i-k, 2j-l, k-2, l-2) \\
&\quad For \quad l = 3 \\
&\quad\quad y(i,j,k,l) = y(i,j,k,l-1) + y(i,j,k-1,l) \\
&\qquad\qquad\qquad + h(i,j,k,l)\, x(2i-k, 2j-l, k-2, l-2) \\
&\quad\quad y(i,j,k,-1) = 0 \\
&\quad\quad y(i,j,-1,3) = 0
\end{aligned}
\tag{2.40}
$$

The index mapping functions are given as follows:

$$
D_{y,1} = D_{y,2} = D_h = I, d_{y1} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -1 \end{bmatrix}, d_{y,2} = \begin{bmatrix} 0 \\ 0 \\ -1 \\ 0 \end{bmatrix}, d_h = \begin{bmatrix} -1 \\ -1 \\ 0 \\ 0 \end{bmatrix}
$$

$$
D_x = \begin{bmatrix} 2 & 0 & -1 & 0 \\ 0 & 2 & 0 & -1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, d_x = \begin{bmatrix} 0 \\ 0 \\ -2 \\ -2 \end{bmatrix}
$$

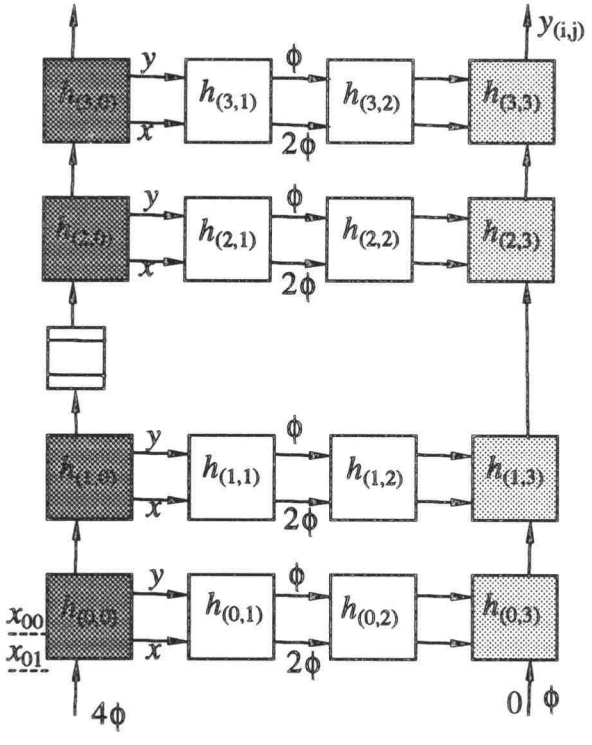
All dependencies are uniform except those of $D_x$. It is a two-dimensional DARE with the dependence matrix defined as:

$$A = D - I = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & -1 & 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & -1 \end{bmatrix} = \mu_1 \alpha_1^T + \mu_2 \alpha_2^T \quad (2.41)$$

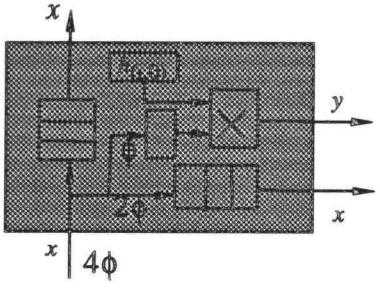
From Thm.2.3, the allocation function must be in the null space of $\mu$'s and it is defined as $\aleph(\mu) = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$. Even equipped with this information, the derivation of the two-dimensional MRA architecture is not a trivial task. Instead of projecting the DAG along one direction, we need to project the DAG twice along both vectors $\mu_1$ and $\mu_2$. Or equivalently, the allocation function is defined as $a(p) = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} p = \begin{bmatrix} k \\ l \end{bmatrix}$. As a result, each node $p = \begin{bmatrix} i & j & k & l \end{bmatrix}^T$ in the computational domain is mapped onto a 2-dimensional processor array.

The resulting multi-rate array is given in Figure 2.8. Three types of processing elements are used in the array. There are three clock rates in the operation of this MRA architecture. The first clock signal has the highest operating frequency denoted as $4\phi$. The second clock signal is $2\phi$, one half of the rate of the first one. The third is $\phi$, the slowest clock frequency in the array, which is one-fourth of $4\phi$. The original data element *x(i,j)* enters the array at the highest rate $4\phi$ and the final computed result *y(i,j)* leaves the array at the lowest rate at $\phi$. The internal block diagrams of these PEs are given in Figure 2.8b,c,d.

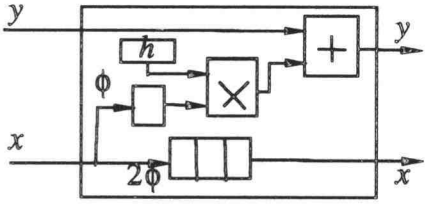
The efficiency of this array is 100% and the throughput rate is 1. This array is capable of producing one result at each clock cycle $\phi$ and receiving input samples at the clock rate of $4\phi$. The operation of this MRA array is similar to the one-dimensional MRA decimation filter. It performs decimation before other computations.



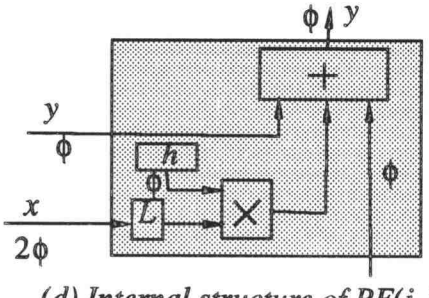

*(a) The MRA system architecture*




*(b) Internal block diagram of PE(i,0)*




*(c) Internal structure of non-boundary PEs*




*(d) Internal structure of PE(j, N)*

**Figure 2.8:** An MRA architecture for the 2-D decimation filter (a) and the internal block diagrams of the processing elements.

There are other algorithms such as the Fast Fourier Transform (FFT) which may also be considered as directional uniform algorithms. But the FFT algorithm is not a DARE because we cannot present it as a DARE. However, the concept of projecting the dependence structure along the non-uniform subspace into the uniform subspace can still be used to produce spatially regular VLSI arrays. But the design procedure may be more involved and can only be performed case by case, instead of following a systematic procedure. The internal structures of these processing elements may not be identical under such circumstances. For example, the $N$-point $(N = 2^n)$ FFT algorithm can be implemented on an array of $n$ PEs with constant interconnections, but the internal structure of each PE differs from one another. However, these differences are minor. Most functional blocks in each PE are the same, except that the number of delays in each PE is different, one from the other.

## 2.5 LIMITATIONS OF THE MULTI-RATE TRANSFORMATION

**CASE I:** Requirements for low level specifications.

Under most circumstances, DARE, like other low level specifications, is not the original algorithmic specification for many digital signal processing applications. Moreover, there does not exist a systematic procedure for transforming an algorithm from its original algorithmic specification into its corresponding DAREs. This has been performed by the use of *ad hoc* methods. Hence, it is impossible to determine if a given algorithm can be implemented onto regular VLSI architectures from its original algorithmic specification, until its corresponding DARE specification is derived. This makes it difficult for algorithmic level optimization. Moreover, the use of the heuristic method for deriving DAREs from their algorithmic specification reduces the solution space of an application. This also decreases the ability to perform architectural level optimizations. These are com-

mon problems for all synthesis methodologies starting with initial specifications such as ARE, URE, and RIA.

In the following, we present two examples in which the existence of the corresponding DARE specifications is not known. How to transform them into corresponding DARE specifications is not clear. These examples are interpolation filter and decimation filter with fractional factors.

**Example[2.6]:** Subband coding has found applications in speech coding, image coding, and other digital signal processing applications. There are two major sections consisting of a subband coding system, the analysis and the synthesis sections. Decimation filters are the main components of the analysis section, while at the synthesis section, interpolation filters play the role. Figure 2.9 shows a block diagram of a subband coding system. At the analysis section, the input signal is decomposed into three signal bands with different frequencies. At the synthesis section, these three subband signals are used to recover the original signal. In the diagram, $x$ denotes the input signal sequence, $y$ denotes the subband signals, $\bar{x}$ denotes the output sequence synthesized from the subband signals $y$.

An interpolation filter increases the sampling rate of the input signal by a factor $L$, which is the dual system of a decimation filter. The synthesis section is composed of decimation filter banks which can be implemented by using the MRA architecture shown in Figure 2.6. It is possible to optimize the design by combining all three decimation filters in one architecture. In the following, we are interested in the implementation of the synthesis section and the interpolation filter banks. The coder and decoder (or communication channel) are of no concern to us here.

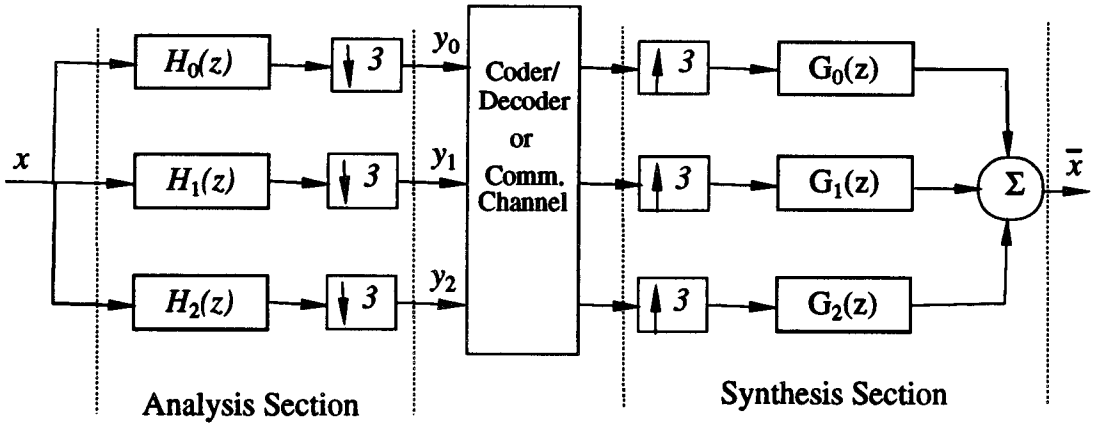



**Figure 2.9:** The block diagram of a three-band critically sampled subband coding system with decimation and interpolation filters.

---

An interpolation filter with interpolation factor $L$ will first expand the size of input data sequence $y$ by $L$ times. This is achieved by inserting $(L-1)$ zeros between any pair of the input samples, $y(i)$ and $y(i+1)$ to form a new data sequence $w$. This new signal sequence $w$ is filtered to obtain a smooth signal. An $N$th order interpolation filter with $L = 3$ is defined as follows:

$$w(i) = \begin{cases} x(\frac{i}{3}), & for \quad \frac{i}{3} = integer \\ 0, & otherwise \end{cases}$$
$$y(i) = \sum_{j=0}^{N-1} g(j)\, w(i-j) \qquad (2.42)$$

In the above expression, redundancy has been introduced through the intermediate variable $w$. The dependence structure of Eq.(2.42) is regular. A direct implementation of this algorithm is shown in Figure 2.10. This implementation suffers from low efficiency, low throughput rate, and low operating frequency, because all operations are performed,

including those multiplications and accumulations with zero valued operands. To improve the efficiency, different structures have been investigated by many researchers [Vai90]. Among them, polyphase implementation provides a fully efficient solution. Our interest here is to investigate how to express this algorithm with high efficiency, and if it is possible to express it in the format required for MRA implementations.

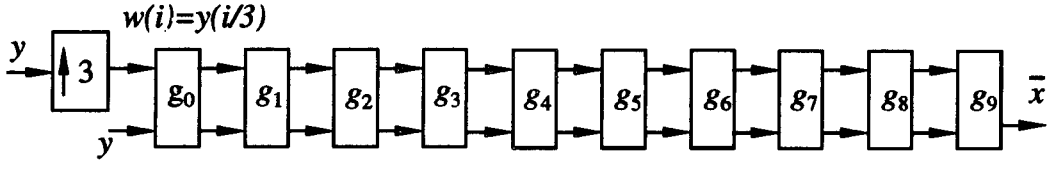



**Figure 2.10:** A single rate array for the 10th order interpolation filter.

---

To improve the efficiency of Eq.(2.42), it is necessary to eliminate those redundant computations, that is, those computations involve with zero-valued operands. We need to eliminate the intermediate variable $w$ and to relate the output variable $\bar{x}$ to the input variable $y$ directly. The following equation presents this relation with some nonlinear operators:

$$\bar{x}(i) = \sum_{l=0}^{\left\lfloor \frac{N-1}{L} \right\rfloor} g(jL + (i) \, mod L) \, y\left(\left\lfloor \frac{i}{L} \right\rfloor - j\right) \tag{2.43}$$

where $(i) \, modL$ is the modular operator, $\lfloor q \rfloor$ is the floor function which yields the greatest integer value which is less than or equal to the given number $q$. The process of deriving a DARE format of this algorithm becomes difficult, if not impossible. Let us denote

$(i)\ mod L$ as $\varphi$ and name it as the phase component of the index quantization. The following is an equation which directly relates the output signal $(\bar{x})$ to the input signal $(y)$:

$$0 \le \varphi \le L-1$$

$$\bar{x}(iL+\varphi) = \sum_{l=0}^{\left\lceil \frac{N-1}{L} \right\rceil} g(jL+\varphi)\, y(i-j) \tag{2.44}$$

The implementation of this equation may yield a fully efficient architecture. The key concern here is whether we have an MRA architecture which can perform such algorithm. From this equation, we do not have a systematic approach for deriving the corresponding DARE specifications. Therefore, the synthesis procedure based on DARE can not be employed for the implementation of this algorithm. This example illustrates the limitations for synthesis procedures starting with low level initial specifications. We do not know how to start the synthesis procedure until we find a corresponding DARE specification. But it is not clear if there exists a DARE expression for it.

**Example[2.7]:** In digital audio, different sampling rates have been used for different media. Non-integral sampling rate converters are used for copying materials from one medium to another. This is performed through the use of decimation filters with a fractional decimation factor. Decimation filter with fractional decimation factor $M/L$ can be described by the block diagram shown in Figure 2.11. The implementation of such filters has drawn much attention in recent DSP applications. It will be shown later in chapters 3 and 4 that this class of algorithms can also be implemented onto the MRA architectures.

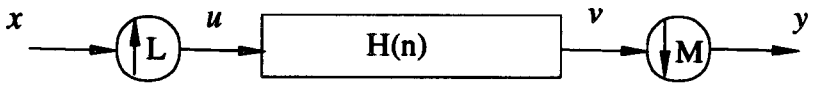



**Figure 2.11:** Decimation filter with a rational decimation factor $M/L$.

An $N$th order decimation filter with fractional decimation factor $M/L$ can be expressed as the following set of equations:

$$\begin{aligned} u(i) &= \begin{cases} x(\frac{i}{L}), \frac{i}{L} = \text{integer} \\ 0, \text{otherwise} \end{cases} \\ v(i) &= \sum_{j=0}^{N-1} h(j)\, u(i-j) \\ y(i) &= v(Mi) \end{aligned} \tag{2.45}$$

Again, intermediate variables have been introduced in this expression. The algorithm expressed in Eq.(2.45) has a regular dependence structure. Direct implementation of the block diagram in Figure 2.11 or Eq.(2.45) results in an array architecture with extremely low efficiency. Only one of $(L \times M)$ computed results must be computed. This efficiency is derived from two facts: first, there are $(L-1)$ out of $L$ computations involved with the multiplication of zero valued samples; secondly, only one of $M$ computed result is used in the output signal. Polyphase implementations [Vai90] present an efficient approach to improve efficiencies of these realizations, but the regular interconnection pattern has been greatly hampered by the continuous redrawing of the block diagrams. Moreover, these polyphase implementations are not programmable to suit different applications.

To utilize the synthesis technique for this algorithm, we again face the problem of transforming Eq.(2.45) into its corresponding DARE specification. There does not exist a known DARE specification for this algorithm. Therefore, the DARE specification becomes the barrier again. But, it is relatively easy to transform Eq.(2.45) into a high efficiency specification as shown below:

$$\begin{aligned} &for \quad (0 \le \phi \le L-1) \\ &y(Li+\phi) = \sum_{j=0}^{\left\lceil \frac{N-1}{L} \right\rceil} h(Lj+\phi)\, x(Mi-j) \end{aligned} \tag{2.46}$$

How to derive a highly efficient regular array architecture from this specification? The DARE approach can hardly help us in such applications.

**CASE II:** Conflicting Interests

In the synthesis procedure, the vector $\mu$ has been used as the projection vector to obtain a regular array architecture. However, in a given algorithm, if the computational domain has a ray $\gamma$ with direction different from the direction of $\mu$, then the synthesis technique developed for DARE cannot be used to derive an array with regular structure and a finite number of processing elements. It is impossible to implement an array with an infinite number of processing elements. Therefore, under such circumstances, if this synthesis technique is to be used, structural regularity must be sacrificed. This is an undesirable situation. Is it possible to develop a new synthesis theory which can produce regular array architectures with a finite number of processing elements for such algorithms?

The second conflict occurs where a given algorithm has two or more affine index mapping functions in an equation, say $\Gamma_1(p) = D_1 p + d_1$ and $\Gamma_2(p) = D_2 p + d_2$ where both dependence matrices $A_1 = D_1 - I$ and $A_2 = D_2 - I$ are singular. That is,

there are two DARE dependencies in the same equation. Let $A_1 = \mu_1 \alpha_1^T$ and $A_2 = \mu_2 \alpha_2^T$. If $\mu_1 \neq \mu_2$, under such circumstances, the synthesis procedure based on DARE cannot be employed to derive a regular MRA architecture for this algorithm, because projecting the DAG along either direction $\mu_1$ or $\mu_2$, the interconnections associated with the other dependency will always be irregular.

**Example[2.8]:** Let the following equation be in a given algorithm:

$$y(i,j) = y(i,j-1)\, w(2i+j, i+2j-1) + h(i,j)\, x(3i-j, j-3) \tag{2.47}$$

where the values of each variable are not important. There are two non-uniform index mapping functions which are defined as:

$$\begin{aligned} D_w &= \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}; \quad d_w = \begin{bmatrix} 0 \\ -1 \end{bmatrix}; \quad A_w = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \end{bmatrix} \Rightarrow \mu_{wy} = \begin{bmatrix} 1 & 1 \end{bmatrix}^T \\ D_x &= \begin{bmatrix} 3 & -1 \\ 0 & 1 \end{bmatrix}; \quad d_x = \begin{bmatrix} 0 \\ -3 \end{bmatrix}; \quad A_x = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \begin{bmatrix} 2 & -1 \end{bmatrix} \Rightarrow \mu_x = \begin{bmatrix} 1 & 0 \end{bmatrix}^T \end{aligned} \tag{2.48}$$

Those conventional techniques cannot be employed to synthesize this problem, since the index mapping matrices are not totally unimodular. Neither can the multi-rate transformation technique developed in this chapter be used to obtain array architectures with regular interconnections because $\mu_w = \begin{bmatrix} 1 & 1 \end{bmatrix}^T$, while $\mu_x = \begin{bmatrix} 1 & 0 \end{bmatrix}^T$. No matter which projection vector is selected, there are always non-constant interconnections.

# Chapter 3 Multi-Coordinate-Systems and Dependence Localization

Traditionally, the dependence structures of algorithms are often analyzed using a single coordinate system and synthesis theories are developed by expressing the computational domain and the index spaces by an $n$-dimensional orthogonal coordinate system [Jay91]. In many digital signal processing applications, the computational domain is defined in an $n$-dimensional Euclidean space $R^n$, but most variables are $m$-dimensional data arrays ($m < n$). Most existing synthesis theories are based on the initial specifications over the space $R^n$; thus the original algorithm index must be extended from $m$ to $n$. Furthermore, most synthesis methods require the dependence map $D$ to be idempotent. For algorithms with non-idempotent index mapping matrices, the approach employed by conventional techniques does not work well and more complicated control mechanisms.

These problems are directly related to the use of a single coordinate system for defining the index spaces and can be solved by expressing the algorithms in a system of multi-coordinate systems (MCS) by defining the initial specification as an *Affine Direct Input Output* (ADIO). The ADIO specification resembles the weakly single assignment codes (WSAC) specifications but is more general due to the introduction of the phase components and the index mapping matrices are $m \times n$, $m \leq n$. This allows the synthesis procedure to start directly from original algorithms (in WSAC index mapping matrices are defined to be $m \times n$ dimensional [RTRK88], but the synthesis procedure starts by extending the specifications to $n \times n$ dimensional by pre-multiplying index mapping matrices).

In an ADIO, the *direct input output* relationships in specifying algorithms are used to minimize the use of intermediate variables which would degrade the performance of

VLSI implementations. The purpose of using these intermediate variables is to obtain uniform data dependence structures. Using MCS, the index space of each variable may be defined in relative to its own coordinate systems. When presented in MCS system, most non-uniform iterative algorithms will present uniform dependence structures after the removal of global broadcasting dependencies. It will be shown that there exists a close relationship between MCS system and MRA structure. Procedures will be provided for the construction of MCS systems for given algorithms, and transforming these algorithms into their corresponding specifications with regular dependence structures. The next step is to find a valid affine allocation function and timing to map a given algorithm onto a VLSI array architecture. In summary, this chapter will provide:

- A definition of the initial specification of algorithms in ADIO.
- A characterization of Multi-Coordinate Systems, and procedures for constructing MCS systems for a given algorithm.
- A technique for deciding the necessary and sufficient conditions to present algorithms with regular dependence structures in MCS systems.
- A procedure for the removal of global dependencies and transforming the ADIOs to regular dependence structures.

Figure 3.1 shows the synthesis procedure for MRA architectures from given algorithms. The thrust of this chapter will be on the development of a systematic procedure to transform a given algorithm into its corresponding specifications with uniform dependence structures.


**Figure 3.1:** Synthesis procedure for mapping ADIOs onto multi-rate VLSI systems

## 3.1 SPECIFICATION OF ALGORITHMS

**Definition 3.1:** *Direct Input Output Specifications* (DIO): A direct input output specification for a set of $s$-equations is defined as:

$$U_i(\Gamma_i(p) + \varphi_i) = f\left[U_1[\Gamma_{1i}(p) + \varphi_{1i}], \ldots, U_{Vi}[\Gamma_V(p) + \varphi_{Vi}]\right] \qquad p \in \Omega_i (1 \le i \le s) \tag{3.1}$$

where:

- $\Omega_i \subseteq L^n$ is the computational domain for $U_i$.
- $U_i(\Gamma_i(p) + \varphi_i)$ is the output variable of the array; $\Gamma_i(p)$ is the index mapping function which maps point $p$ to $\Gamma_i(p)$.
- $U_j(\Gamma_{ji}(p) + \varphi_{ji})$, $1 \le j \le V$, are the input variables of the $i$-th array. Each input variable may be the output variable of the same or another array, *e.g.*, the output variable $U_i$ may also appears as an input variable on the right hand side of Eq.(3.1).
- $f$ is a function.
- $\varphi_i$ and $\varphi_{ji}$ are the phase components for the output and input variables respectively.
- The computation domains $\Omega_i$, $\Omega_j$,... need not be the same. ■

Note that the computed variable $U_i(\Gamma_i(p) + \varphi_i)$ may not appear on the right-hand side of the equation and in general, Eq.(3.1) may not be a recurrence equation.

**Definition 3.2:** *Affine Direct Input Output (ADIO) Specifications*: An ADIO is a DIO whose index mapping functions are affine (linear):

$$\Gamma(p) = Dp + d, \quad D \in Z^{m \times n}, d \in Z^m \tag{3.2}$$

where $Z^m$ is an integer lattice point in $R^m$, $D$ is an $m \times n$ index mapping matrix, $d$ is the translation part, an $m \times 1$ constant vector, and $1 \le m \le n$. ■

**Example[3.1]:** An IIR filter is defined as:

$$y(i) = \sum_{0 \le j \le N_1} a(j)\, x(i-j) + \sum_{1 \le j \le N_2} b(j)\, y(i-j) \tag{3.3}$$

The above is the ADIO specification of the algorithm. The computational domains are defined as $\Omega_a = \{p \mid 0 \le i, 0 \le j \le N_1\}$ and $\Omega_b = \{p \mid 0 \le i, 1 \le j \le N_2\}$ where the index point $p = [i\ j]^T \in Z^2$. The index mapping functions are:

$$\begin{aligned}
\Gamma_y(p) &= [1\ 0]p, & D_y &= [1\ 0] \\
\Gamma_{xy}(p) &= [1\ -1]p, & D_{yy} &= [1\ -1] \\
\Gamma_{xy}(p) &= [1\ -1]p, & D_{xy} &= [1\ -1] \\
\Gamma_{ay}(p) &= [0\ 1]p, & D_{ay} &= [0\ 1] \\
\Gamma_{by}(p) &= [0\ 1]p, & D_{by} &= [0\ 1]
\end{aligned} \tag{3.4}$$

There is no intermediate variable in the above specification.

**Example[3.2]:** Interpolation filters have many applications such as subband speech coding, subband image coding, wavelet analysis of images, etc. [Cro83, Mal89, Vai90]. An *N*th order FIR interpolation filter with upsampling factor $L$ can be expressed as follows:

$$\begin{cases} w(i) = \begin{cases} y(\frac{i}{L}), & for \quad \frac{i}{L} = integer \\ 0, otherwise \end{cases} \\ \bar{x}(i) = \sum_{j=0}^{N-1} g(j)\, w(i-j) \end{cases} \tag{3.5}$$

where $w$ is an intermediate variable used to relate the input and output variables $y$ and $\bar{x}$. The direct VLSI implementation of Eq.(3.5) is not efficient since some of the variable computations involve zero operand multiplications and additions. The ADIO specification of Eq.(4.5) is:

$$\begin{aligned} &for \quad 0 \le \varphi \le L-1 \\ &\qquad \bar{x}(iL+\varphi) = \sum_{l=0}^{\left\lfloor \frac{N-1}{L} \right\rfloor} g(jL+\varphi)\, y(i-j) \end{aligned} \tag{3.6}$$

The computational domain is $\Omega = \{p \mid 0 \le i, 0 \le j \le N-1\}, p = [i\ j]^T \in Z^2$. The index mapping functions are:

$$\begin{aligned} \Gamma_{\bar{x}}(p) &= [L\ 0]p, \quad \varphi_{\bar{x}} = \varphi, \quad D_{\bar{x}} = [L\ 0] \\ \Gamma_{g}(p) &= [0\ L]p, \quad \varphi_{g} = \varphi, \quad D_{g} = [0\ L] \\ \Gamma_{y}(p) &= [1\ -1]p, \quad \varphi_{y} = 0, \quad D_{y} = [1\ -1] \end{aligned} \tag{3.7}$$

**Example[3.3]:** An FIR decimation filter is defined as:

$$\begin{cases} w(i) = \sum_{j} h(j)\, x(i-j) \\ y(i) = w(Mi) \end{cases} \tag{3.8}$$

where $M$ is the decimation factor. The intermediate variable $w$ is computed at every index point but only one out of every $M$ computed samples is used as the output sample of the fil-

ter. The efficiency of this specification is $1/M$. The ADIO specification of Eq.(3.8) is as follows:

$$y(i) = \sum_{j=0}^{N-1} h(j)\, x(Mi-j) \tag{3.9}$$

The computational domain is $\Omega = \{p \mid 0 \le i, 0 \le j \le N-1\}, p = [i\ j]^T \in Z^2$. The index mapping functions are:

$$\begin{aligned} \Gamma_y(p) &= [1\ 0]p, \quad D_y = [1\ 0] \\ \Gamma_h(p) &= [0\ 1]p, \quad D_h = [0\ 1] \\ \Gamma_x(p) &= [M\ -1]p, \quad D_x = [M\ -1] \end{aligned} \tag{3.10}$$

The dependencies described by Eqs.(3.6) and (3.9) can not be localized by the use of conventional methodologies. The conventional approaches require the algorithms be initially expressed as URE, ARE or RIA and the index mapping matrices be unimodular. Transforming original algorithmic specifications to ARE and URE in general requires index space expansion.

The affine recurrence equation (ARE) is the expanded version of an ADIO specification where index space of the output variable has been expanded. For example, $y(i,j) = y(i,j-1) + h(j)\, x(Mi-j)$ is an ARE specification of the ADIO defined in Eq. (3.9), which is just one of many possible ARE formats derived from the ADIO specification in Eq.(3.9), which is just one of many possible ARE formats that can be derived from the ADIO.

## 3.2 MULTI-COORDINATE SYSTEMS

The concept of coordinate was first introduced by Descartes in 1637 [Ben58] and later expanded to rectangular cartesian coordinate, spherical polar coordinates, cylindrical coordinates, *etc.* It is possible to transform the specification of an algorithm from one coordinate system to another, however, representation of an algorithm in more than one coordinate systems is not common.

The $n$-dimensional coordinate system was first introduced by an Italian geometer Enrico Betti [Ros88], *"On spaces of an arbitrary number of dimensions"* (1871) [Bet03, *vol.2, pp. 273-290*] as:

> "Let $z_1, z_2, \ldots, z_n$ be $n$ variables that can take on all real values from $-\infty$ to $+\infty$. We will call the $n$-ply infinite field of systems of values of these variables a *space* of $n$ dimensions and denote it by $S_n$. A system $(z_1^0, z_2^0, \ldots, z_n^0)$ will define a *point* $L_0$ of this space; we will call $z_1^0, z_2^0, \ldots, z_n^0$ the *coordinates* of this point."

It is well known that an $n$-dimensional coordinate system is defined by $n$ independent $n \times 1$ base vectors and a reference point called the origin. Given a coordinate system and an $n$-tuple, or $n$-ply as in above, $(z_1, z_2, \ldots, z_n)$, any point can be uniquely defined. The scale coordinate was introduced by W. Bender [Ben58, Ben75] for the representation of mathematical quantizations of Riemannian geometry for applications in modern *quantum mechanics*. The coordinate system used for the analysis of data dependence structures for DSP algorithms is not exactly the same scale coordinate defined in [Ben58] since the physical dimensions of the scales are not of concern.

The simplest coordinate system in the $n$-space $R^n$ is called the *standard coordinate system* $C_s^n$ which is defined as:

**Definition 3.3:** *Standard Coordinate System* $C_s^n$*:* An $n$-dimensional standard coordinate system, $C_s^n$ is an $n$-dimensional orthogonal coordinate system defined by $n$ base vectors $\mathbf{e}_i = [0\ 0 \ldots\ 0\ 1\ 0\ \ldots\ 0]^T$, $1 \le i \le n$, with 1 is at its $i$th location. The axes of $C_s^n$ are labelled as $z_1, z_2, \ldots, z_n$ respectively. ■

Given a coordinate system, any vector $\mathbf{v}$ can be represented as a linear combination of its basis:

$$\mathbf{v} = c_1\mathbf{e}_1 + c_2\mathbf{e}_2 + \ldots + c_n\mathbf{e}_n$$

The $n$-tuple $(\mathbf{v})_{\mathbf{e}}$ is the vector of $\mathbf{v}$ represented basis $\{\mathbf{e}_1, \ldots, \mathbf{e}_n\}$. Hereafter, any vector $\mathbf{v}$ or point $p$ in $R^n$ is defined using $C_s^n$ will be represented as $\mathbf{v}$ or $p$ without the subscript. For other coordinate system $C_j^n$ with the basis $\mathbf{u}$ the subscript $(\mathbf{v})_{\mathbf{u}}$ will be used to notify the vector with the basis $\mathbf{u}$.

**Example[3.4]:** Let $\mathbf{v} = (2, 2)$; $\mathbf{v} \in R^2$ in the standard coordinate $C_s^2$. Consider the basis $\mathbf{u} = \{\mathbf{u}_1, \mathbf{u}_2\}$ with coordinate $C_1^2 \in R^2$, where $\mathbf{u}_1 = [1/2\ 0]^T$, $\mathbf{u}_2 = [-1/2\ 1]^T$ resulting in $(\mathbf{v})_{\mathbf{u}} = (6, 2)_{\mathbf{u}}$. The new basis $\mathbf{u}$ can be represented by transformation $\mathbf{T}$ of the standard basis in the form $[\mathbf{u}_1\ \mathbf{u}_2] = \mathbf{T}[\mathbf{e}_1\ \mathbf{e}_2]$ as:

$$\begin{aligned}[\mathbf{u}_1\ \mathbf{u}_2] &= \begin{bmatrix} 1/2 & -1/2 \\ 0 & 1 \end{bmatrix}[\mathbf{e}_1\ \mathbf{e}_2] \Rightarrow \mathbf{v} = \begin{bmatrix} 2 \\ 2 \end{bmatrix} = \left(\mathbf{T}^{-1}\begin{bmatrix} 2 \\ 2 \end{bmatrix}\right)_{\mathbf{u}} \\ &= \begin{bmatrix} 1/2 & -1/2 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 6 \\ 2 \end{bmatrix} \Rightarrow \begin{bmatrix} 2 \\ 2 \end{bmatrix} = \left(\begin{bmatrix} 6 \\ 2 \end{bmatrix}\right)_{\mathbf{u}}\end{aligned} \tag{3.11}$$

Figure 3.2 shows this vector defined relative to different coordinate systems.

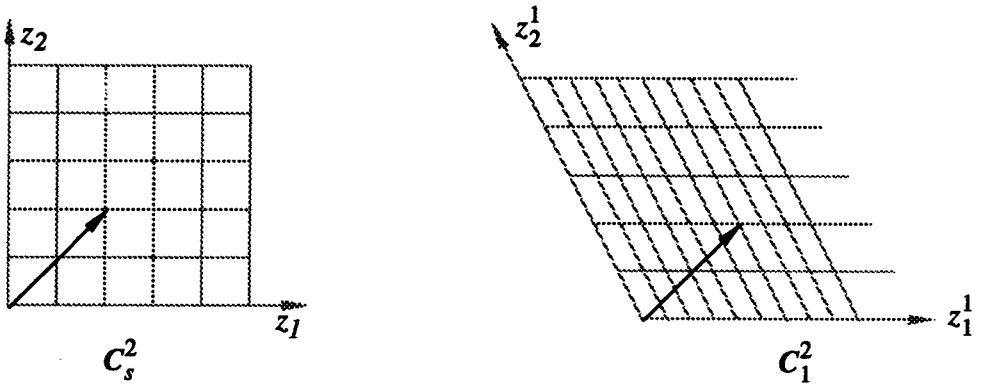



**Figure 3.2:** A vector is defined relative to different basis in $R^2$.

---

**Definition 3.4:** *Multi-Coordinate Systems* (MCS): A system of $M$-coordinate systems consists of coordinates $C_j^{m_j} \in R^{m_j}$ for $1 \le j \le M$ and the *s*tandard coordinates $C_s^n \in R^n$. The basis for each coordinate $C_j^{m_j}$ is denoted as $\mathbf{u}^j = \{\mathbf{u}_1^j, \mathbf{u}_2^j, \ldots, \mathbf{u}_m^j\}$ and the same origin as $C_s^n$ will be used if it is not defined otherwise. ■

Therefore, any vector $\mathbf{v}$ can be described in $C_j^{m_j}$ by the transformation from $C_s^n$ to $C_j^{m_j}$ as $\mathbf{T}_j : Z^{n \times n} \rightarrow Q^n$. For a given algorithm, the objective is to construct a proper MCS system with regular dependencies. First, let us introduce the format of algorithm specification under MCS system with which the dependence structure of a given algorithm is provided.

Consider an $n$-dimensional recurrence (original algorithmic expression):

$$U_i(p) = f[U_i(q_{ii}), U_j(q_{j1}), U_j(q_{j2}), \ldots] \tag{3.12}$$

The index points $p, q$ in this expression are relationship among the variables $U_i$, $U_j$and do not correspond to any computational domain or particular dependence graph

nodes. However, for the synthesis, it is necessary to embed the variables and its index into a relevant $n$-dimensional computational domain (i.e., assigning dependency flow). The procedure of embedding a variable array into the computational domain $\Omega$ is called *an embedding*.

An *embedding* of the variable index $U_j(q), q \in Z^m$ into the $n$-fractional space $Q^n$ is a mapping function $\sigma : Z^m \rightarrow Q^n$ which maps point $q$ to point $p_q \in Q^n$.

Eq.(3.12) provides no information on how to embed variables into the $n$-space and how to assign nodes and dependence structure. Note that different variable may have different dimensions and embedding all variables into the $n$-space according to a single coordinate system may not be desirable for dependence localization.

## 3.3 ADIO DEPENDENCE STRUCTURE

Let $U_o(D_o p + d_o) = f[\ldots, U_j((D_j p + d_j), \ldots)$ be an $n$-dimensional ADIO where $U_o$ and $U_j$ designate the output and input variables respectively. Since the translation part $d$ is a constant vector, it is not included in the analysis.

For algorithm dependence structure analysis, it is necessary to know the set of nodes where the element $U_j(q)$ are used. This sets of nodes will be defined as:

$$N_j(q) = \{p | \forall p \in \Omega \wedge W(p) \leftarrow U_j(q)\} \tag{3.13}$$

where some other variable $W(p), p \in \Omega$ requires elements of $U_j(q)$. Similarly, for the output variable, the set of nodes where the output nodes requires a value is denoted as:

$$N_o(q) = \{p | \forall p \in \Omega \wedge U_o(q) \leftarrow W(p)\} \tag{3.14}$$

Thus, the element $U_j(q)$ is sent to node $p$ if and only if $p \in N_j(q)$ ; and $U_o(q)$ requires an element from node $p$ if and only if $p \in N_o(q)$ .

The objective is to find the node sets $U_o(q)$ and $U_j(q)$ , and to use the intermediate variable to perform the data transfer from $U_j(q) \rightarrow U_o(q)$ via $W(p), p \in \Omega$. The objective is to find the relationship among $W(p)$ and the elements of variables in an ADIO. Using $W(p)$ , an ADIO can be expressed as follows:

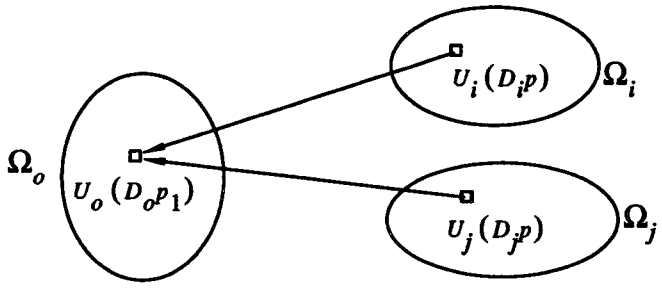
$$U_o(D_o p) = g(W(p)) = f[\ldots, U_j((D_j p), \ldots)] \tag{3.15}$$

The variable $W(p)$ decomposes the dependence relationships between $U_o$ and $U_j$ into the dependence relationships between first $W(p) \leftarrow U_j(D_j p)$ and then $U_o(D_o p) \leftarrow W(p)$ and $W(p)$ . The node sets $N_j(q) \subseteq \Omega$ and $N_o(q) \subseteq \Omega$ are as shown in Figure 3.3, nodes $N_i(q) = \{p_1, p_2\}$ require $U_i(D_i p)$ and nodes $N_j(q) = \{p_3, p_4\}$ require $U_j(D_j p)$ and the output $U_o(D_o p)$ requires values from $N_o(q) = \{p_1, p_3\}$ . Therefore:

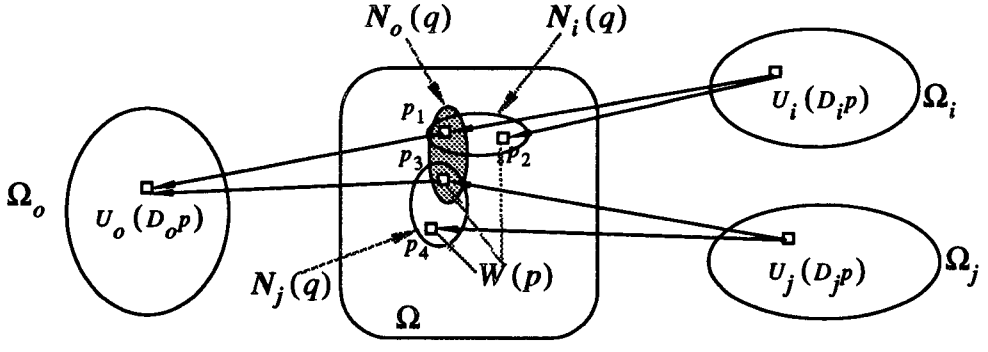
$$\left.\begin{array}{l} p_1 \leftarrow D_i p \\ p_2 \leftarrow D_i p \end{array}\right\} \Rightarrow D_i(p_1 - p_2) = 0 \Rightarrow p_1 - p_2 \in \aleph(D_i)$$

$$\left.\begin{array}{l} p_3 \leftarrow D_j p \\ p_4 \leftarrow D_j p \end{array}\right\} \Rightarrow D_j(p_3 - p_4) = 0 \Rightarrow p_3 - p_4 \in \aleph(D_j)$$

$$\left.\begin{array}{l} D_o p \leftarrow p_1 \\ D_o p \leftarrow p_3 \end{array}\right\} \Rightarrow D_o(p_1 - p_3) = 0 \Rightarrow p_1 - p_3 \in \aleph(D_o)$$




*(a) Direct input-output dependence relationships, $\Omega_o, \Omega_i, \Omega_j \subseteq \Omega$*




$$D_o(p_1 - p_3) = 0; \quad D_i(p_1 - p_2) = 0; \quad D_j(p_3 - p_4) = 0; \quad p_i \in \Omega$$

*(b) Decompose the input-output dependence relationships down to two separated parts*

**Figure 3.3:** Dependence relationships among input and output variables and $W(p)$.

---

The following lemmas provide the tools to determine $N_j(q)$ for $U_j(D_j p + d_j)$ and $N_o(q)$ for $U_o(q)$.

**Lemma 3.1:** If $W(p) \leftarrow U_j(D_j p + d_j)$, $W(p_1)$ and $W(p_2)$ require $U_j(q)$, $\exists q \in \Omega_j$, $\forall p \in \Omega$, if and only if $p_1 - p_2 \in \aleph(D_j)$.

**Proof.** If $p_1 - p_2 \in \aleph(D_j)$, from the definition of the right null space of a matrix, we have:

$$D_j(p_1 - p_2) = 0 \Rightarrow D_j p_1 = D_j p_2 \Rightarrow D_j p_1 + d_j = D_j p_2 + d_j$$

Therefore, $W(p_1)$ and $W(p_2)$ depends on the same data element $U_j(D_j p + d_j)$.

Only if: if $W(p_1)$ and $W(p_2)$ depends on the same data element of variable $U_j$:

$$D_j p_1 + d_j = D_j p_2 + d_j \Rightarrow D_j(p_1 - p_2) = 0 \Rightarrow p_1 - p_2 \in \aleph(D_j). \quad \blacksquare$$

In other words, if $\exists p_1 \in \Omega$ such that $W(p_1) \leftarrow U_j(q)$, then the node set $N_j(q)$ is defined as:

$$N_j(q) = \{p \mid \forall p \in \Omega, p - p_1 \in \aleph(D_j)\} \tag{3.16}$$

**Lemma 3.2:** If $U_o(D_o p + d_o) \leftarrow W(p)$, then $U_o(q), \forall q \in \Omega_o, \forall p \in \Omega$, requires $W(p_1)$ and $W(p_2), \exists p_1, p_2 \in \Omega$, if and only if $p_1 - p_2 \in \aleph(D_o)$.

**Proof.** The proof of this lemma is similar to lemma 3.1. $\blacksquare$

In other words, if $\exists q \in \Omega_o \wedge \exists p_1 \in \Omega, U_o(q) \leftarrow W(p_1)$, then the node set $N_o(q)$ is defined as:

$$N_o(q) = \{p \mid \forall p \in \Omega, p - p_1 \in \aleph(D_o)\} \tag{3.17}$$

Lemma 1 and 2 describe an important data dependence relationship in an ADIO. These properties are similar to the null space propagation technique [Raj86, RTRK89] but without the idempotent restriction, or $D_o D_j = D_j^2$ ($PQ = Q^2$ as given in [RTRK89]). The question now is how can the dependencies of an ADIO be localized by the use of null-space propagation even if the idempotent condition is not satisfied?

**Example[3.5]:** Consider the dependence relationships of an ADIO:

$$y(i) = \sum_{j=0}^{6} h(j)\,x(3i-j)$$

The index mapping matrices of this ADIO is given as:

$$D_y = [1\ 0], D_h = [0\ 1], D_x = [3\ -1]$$

where $y$ is the output variable, $h$ and $x$ are the input variables. The computational domain of this ADIO is $\Omega = \{ \begin{bmatrix} i \\ j \end{bmatrix} \mid 0 \le i, 0 \le j \le N \}$. Figure 3.4 shows the direct dependencies among the output and input variables without $W(p)$. Each line linking the elements $y(i)$, to the elements $h(j)$ or $x(k)$ represents that the evaluation of $y(i)$ requires the values of $h(j)$ and $x(k)$. The dependence structure is not well organized in Figure 3.4.

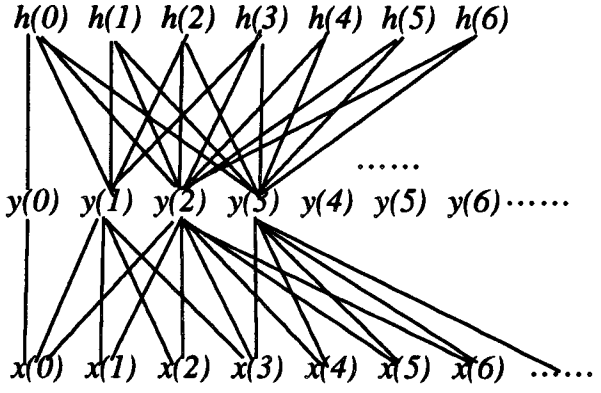



**Figure 3.4:** Direct dependence relationships between the output variable $y$ the input variables $x$, $h$ of the decimation filter.

---

Let the sets of nodes be denoted as $N_y(i), N_h((j), N_x(i))$ respectively. $W(p)$ is $W(i,j), \forall (i,j) \in \Omega$. To compute these node sets $N_y(i), N_h((j), N_x(i))$, we can

examine the computations performed in the ADIO directly. For example, let us compute a node set $N_y(2)$ for the output variable $y$:

$$y(2) = \sum_{j=0}^{6} h(j)x(6-j) \Rightarrow N_y(2) = \{(2,j) | (0 \le j \le 6)\}$$

This can be repeated for the determination of all node sets. However, it is not necessary to go through this procedure to determine the node sets. Let us use lemma 3.1 and 3.2 to compute these node sets. The null spaces of the index mapping matrices are:

$$\aleph(D_y) = \begin{bmatrix} 0 & 1 \end{bmatrix}, \aleph(D_h) = \begin{bmatrix} 1 & 0 \end{bmatrix}, \aleph(D_x) = \begin{bmatrix} 1 & 3 \end{bmatrix}$$

From lemma 3.2, $y(i) \leftarrow W(p_1), W(p_2)$ if and only if $p_1 - p_2 \in \aleph(D_y) = \begin{bmatrix} 0 & 1 \end{bmatrix}$. Thus, the node set of the output variable is defined as:

$$N_y(i) = \{(i,j) | 0 \le j \le 6\}, 0 \le i$$

Similarly, from lemma 3.1, $W(p_1), W(p_2) \leftarrow h(j)$ iff $p_1 - p_2 \in \aleph(D_h) = \begin{bmatrix} 1 & 0 \end{bmatrix}$. Thus the node set of the input variable h is defined as:

$$N_h(j) = \{(i,j) | 0 \le i\}, 0 \le j \le 6$$

Again, from lemma 3.1, $W(p_1), W(p_2) \leftarrow x(i)$ iff $p_1 - p_2 \in \aleph(D_x) = \begin{bmatrix} 1 & 3 \end{bmatrix}$. The node set $N_x(q)$ is defined as:

$$N_x(3i-j) = \{(i,j) | 0 \le i, 0 \le j \le 6\}$$

If node $(p_1 \in N_x(q)) \wedge (p_1 \pm \begin{bmatrix} 1 & 3 \end{bmatrix} \in \Omega)$ and $p_1 \pm \begin{bmatrix} 1 & 3 \end{bmatrix} \in \Omega$, then these nodes $(p_1 \pm \begin{bmatrix} 1 & 3 \end{bmatrix}) \in N_x(q)$. For example, the node $p_1 = \begin{bmatrix} 1 & 3 \end{bmatrix} \in N_x(0)$, then the node $p_2 = p_1 + \begin{bmatrix} 1 & 3 \end{bmatrix} = \begin{bmatrix} 2 & 6 \end{bmatrix} \in N_x(0)$. Similarly, $p_3 = p_1 - \begin{bmatrix} 1 & 3 \end{bmatrix} = \begin{bmatrix} 0 & 0 \end{bmatrix} \in N_x(0)$ is also true. The node set $N_x(1)$ contains these nodes $\{\begin{bmatrix} 1 & 2 \end{bmatrix}, \begin{bmatrix} 2 & 5 \end{bmatrix}\}$ and $N_x(2)$ contains these

nodes $\{[2\ 1], [3\ 4]\}$. All other node sets can be determined similarly. Note that the elements of each node set belong to the null space $\aleph(D_x)$.

The data dependence structure is shown in Figure 3.5. The circles denote the index points $W(p)$ (or the nodes) in the computational domain. The node sets are either surrounded by dashed lines (for $N_y(i)$ and $N_h(j)$) or simply struck through by a straight line (for $N_x(q)$) depending on the available space. From the node sets defined above, it is easy to see that the null space propagation technique can be employed even if the index mapping matrices are not idempotent. ■

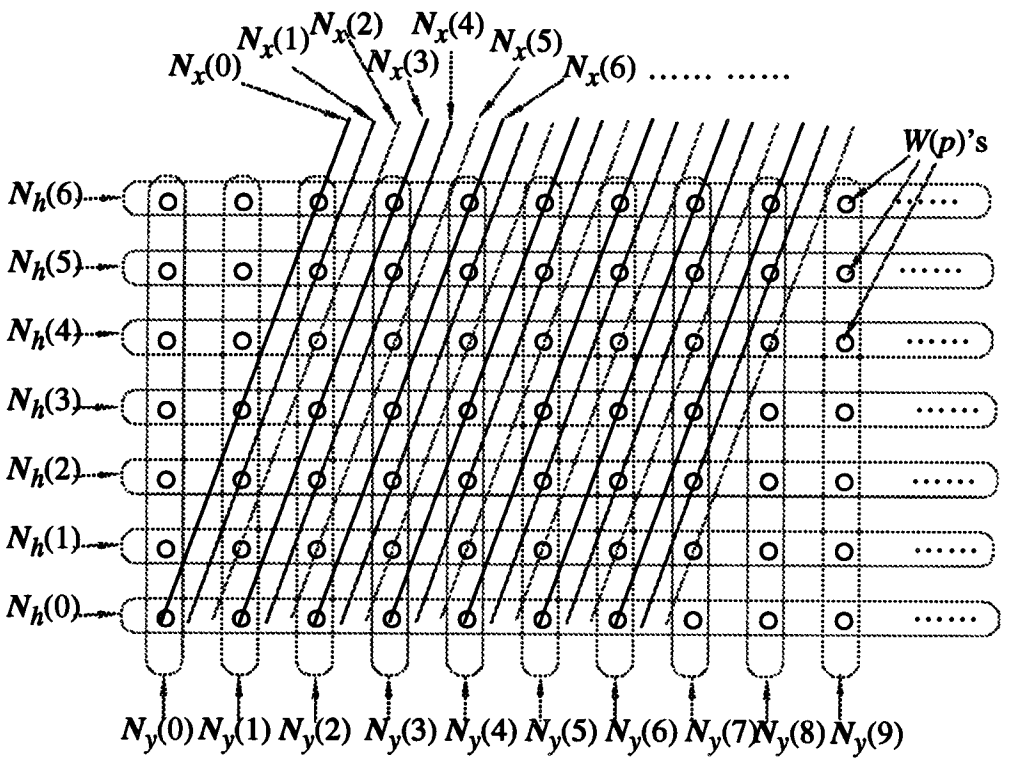



**Figure 3.5:** Decomposing the dependencies down to the dependencies between the output $y$ and $W(p)$; and between $W(p)$ and variables $x$ and $h$.

The objective of embedding is to assign variables $U_j(q)$ in an ADIO to points $p_q \in Q^n$ (i.e., assigning direction of data flow and index points in the DAG). For example, consider the decimation filter example in ADIO form:

$$y(i) = \sum h(j)\, x(3i-j)$$

For variable $x(3i-j)$, we need to assign point $q = [3i-j] = [3\ -1]p$ to a set of nodes $p_q \in Q^2$ in the computational domain. This set of points $p_q \in Q^2$ lies in the null space of $D_x$, $\aleph(D_x)$ such that $D_x(p-p_q) = 0$. For this example,

$$D_x(p-p_q) = [3\ -1](p-p_q) = 0 \Rightarrow p_q = \begin{bmatrix} i/3 \\ 0 \end{bmatrix}$$

For example, from the above example, we know that $N_x(1) = \{[1\ 2], [2\ 5]\}$, $N_x(2) = \{[2\ 1], [3\ 4]\}$, the elements $x(1)$ and $x(2)$ will be embedded into the nodes $p_{q1} = \begin{bmatrix} 1/3 \\ 0 \end{bmatrix}$ and $p_{q2} = \begin{bmatrix} 2/3 \\ 0 \end{bmatrix}$ respectively. Such embeddings satisfy the requirement

$$W(p) \leftarrow U_j(q) \Rightarrow D_j(p-p_q) = 0$$

Therefore, $U_j(q)$ can be propagated to $\forall p \in N_j(q)$ by using the null space propagation technique. Idempotent condition is not required for $D_j$.

**Lemma 3.3:** For the variable $U_j(q), q \in Z^m$ in an ADIO, its node $q$ can be embedded to an index point $p_q \in Q^n$ iff $p_q$ lies on the right null space of $D_j$ such that $\forall p \in N_j(q)$ $D_j(p-p_q) = 0$.

**Proof.** $\forall p \in N_j(q) \subset \Omega$, $W(p) \leftarrow U_j(q)$, if the condition $D_j(p-p_q) = 0$ is satisfied, then the null space $\aleph(D_j)$ can be used to propagate $U_j(q)$ from the index point $p_q$ to $\forall p \in N_j(q)$.

On the other hand, if the condition is not satisfied, i.e., $\exists p_1 \in N_j(q)$ such that $D_j(p_1-p_q) \neq 0 \Rightarrow p-p_q = \vec{v}_1 \notin \aleph(D_j)$. Then, in order to propagate $U_j(q)$

from node $p_q$ to node $p$, the propagation must be along the vector $\vec{v}_1 \notin \aleph(D_j)$. Moreover, from lemma 3.1, if $\exists p_1, p_2 \in \Omega, W(p_1), W(p_2) \leftarrow U_j(q)$, then $D_j(p_1 - p_2) = 0 \Rightarrow p_2 = p_1 + k\vec{\vartheta}, \vec{\vartheta} \in \aleph(D_j)$ for some integer $k$. Thus, $D_j(p_1 - p_q) \neq 0 \Rightarrow p_2 - p_q = k\vec{\vartheta} + \vec{v}_1 = \vec{v}_2 \in \aleph(D_j)$. This requires that the element $U_j(q)$ be propagated along $\vec{v}_2$ to the index point $p_2$. Consequently, multiple directional propagation technique must be employed to propagate the variable. However, as pointed out before, dependence localization by the multiple directional propagation technique requires multiple interconnections and complicated control mechanisms for VLSI implementation. Therefore, only if the node $q = D_j p$ is embedded to the index point $p_q \in Q^n$ such that $\forall p \in N_j(q), W(p) \leftarrow U_j(q), D_j(p - p_q) = 0$, then the null space $\aleph(D_j)$ can be used for dependence localization and no other directional propagation will be required. ■

Obviously, the embedding function $\sigma_j : Z^m \rightarrow Q^n$ is defined as $\sigma_j(q) = p_q$, $p_q \in Q^n$. The question now is how can we find $p_q \in Q^n$ for each variable $U_j(q)$ systematically? This can be achieved through the use of a system of multi-coordinate systems, i.e., by constructing a coordinate system for each $U_j$ itself.

### 3.3.1 Constructing coordinate systems

This section is devoted to the development of a procedure which allows us to determine index points $p_q$ systematically for embedding.

In this section, we are going to adopt the notations $\alpha$, $\beta$ given in [HJ85] (p.17) for defining submatrices. Let $D \in Z^{m \times n}$. "For index sets $\alpha \subseteq \{1, \ldots, m\}$ and $\beta \subseteq \{1, \ldots, n\}$, we denote the (sub)matrix that lies in the rows of $D$ index by $\alpha$ the columns indexed by $\beta$ as $D(\alpha, \beta)$. For example,

$$D = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \Rightarrow D(\{1,3\}, \{1,2,3\}) = \begin{bmatrix} 1 & 2 & 3 \\ 7 & 8 & 9 \end{bmatrix}$$

etc." (Note, the notation for a matrix has been changed from $A$ to $D$ for consistency with the rest of the dissertation).

Moreover, we will adopt the notation $|\beta|$ used in [CR81] for denoting the cardinality of the set $\beta$, that is $|\beta| = card(\beta)$ = the number of elements in $\beta$. (Note, the set is denoted as $A$ in [CR81] instead of $\beta$).

For simplicity, we will use the short hand notation $D(\beta)$ to denote a submatrix of $D$ which consists of $\beta$ columns of $D$. For example, if $\beta_1 = \{1,2\}$ and $\beta_2 = \{2,3\}$, then the submatrices of $D$ are defined as:

$$D = \begin{bmatrix} 1 & 2 & -1 \\ 1 & 0 & 2 \end{bmatrix} \Rightarrow D(\beta_1) = \begin{bmatrix} 1 & 2 \\ 1 & 0 \end{bmatrix}, D(\beta_2) = \begin{bmatrix} 2 & -1 \\ 0 & 2 \end{bmatrix}$$

For any given variable array $U(q)$ in an ADIO, can it be guaranteed that there exists a systematic procedure to determine $p_q$ for $U(q)$ ?

**Theorem 3.1:** For a given $m$-dimensional variable array $U(q)$, it is always possible to find a coordinate system $C^m$ and an embedding function $\sigma : Z^m \rightarrow Q^n$ which maps $U(q)$ to an index point $p_q \in Q^n$ such that $\forall p \in N(q), D(p_q - p) = 0$.

**Proof.** This theorem is proved by the following two cases.

> **Case I:** $\rho(D) = n$. The null space of this variable array is zero dimension and every element $U(q)$ will be used at most only at one location and no propagation of the elements will be required. To make regular dependencies, the linear transformation $T = D^{-1}$ is used to get the base vectors of the new coordinate system $C_j^n$ are defined as:

$\mathbf{u}_l^j = \mathbf{T}\mathbf{e}_l;\quad 1 \le l \le n$

If $U(q)$ is an output variable array with coordinate $C_j^n$, then the dependencies between $W(p)$ and $U(p)$ can be expressed as $U(q)_{\mathbf{u}} = U(p + D^{-1}d) = g(W(p))$. If $U(q)$ is an input variable array, then $g(W(p)) = U(q)_{\mathbf{u}} = U(p + D^{-1}d)$. These dependencies are constant over the entire computational domain and can be presented as a dependence vector $D^{-1}d$.

**Case II:** $\rho(D) = m, D: m \times n, m < n$.

For a rank $m$ matrix $A$, the following statements are equivalent [HJ85]:

(a) rank $A = m$;

(b) There exist $k$, and no more than $k$, columns of $A$ that constitute a linearly independent set;

(c) There exist a $k$-by-$k$ submatrix of $A$ with nonzero determinant, but all $(k+1)$-by-$(k+1)$ submatrices of $A$ have determinant 0.

The right null space $\aleph(D)$ is used for dependence localization using $W(p)$. For an $m$-dimensional data array, we need an $m$-dimensional coordinate system to define its index space. To find the $m$-dimensional coordinate $C_j^m$, we need to extract an $m \times m$ non-singular submatrix $D(\beta)$ from the $m \times n$ index mapping matrix $D$, where $\beta \subseteq \{1, 2, \ldots, n\}$ and $|\beta| = m$. Each selection of $\beta$ determines an $m$-subspace of $Q^n$ in which the coordinate system $C_j^m$ is constructed. If $k \in \beta$, then the $k$-th dimension of the $n$-space is selected as one dimension in the $m$-space. The selection of $\beta$ must be such that $D(\beta)$ is non-singular. Otherwise, the space spanned by $D(\beta)$ is not $m$-dimensional but

$(m-k)$-dimensional, where $1 \le k \le m$. An $(m-k)$-space is not sufficient to define an array of $m$-dimensional.

To construct the $m$-dimensional coordinate system $C_j^m$, we need to find the transformation T. The procedure for deriving T consists of the following steps:

1. Select a subspace to embed the variable array $U$, i.e., select $\beta$;
2. Select the data flow direction, or the iteration direction;
3. Construct the submatrix $D(\beta)$ which is non-singular, compute its inverse $D^{-1}(\beta)$;
4. Construct T as: $\forall k \notin \beta \wedge k \in \{1, \ldots, n\}$, fill the $k$-th row and column with 0's. Then $\forall k \in \beta$, fill those unoccupied positions in the $k$-th column with the corresponding elements of $D^{-1}(\beta)$.

Once the transformation T is determined, the coordinate system $C_j^m$ is defined by the basis $\mathbf{u}_i = \mathbf{T}\mathbf{e}_i$, $1 \le i \le n$.

Note that in $C_j^m$ defined above, only those $k$-th dimension, $\forall k \in \beta$, has nonzero base vector and the other dimensions reduce to zero. This puts the origin of $C_j^m$ at the same location as the origin of the standard coordinate. For different data flow directions, the origin of $C_j^m$ may be moved to other locations.

Embed the variable array $U(q)$ to $Q^n$ relative to its own coordinate system $C_j^m$. The physical location of $U(q)$ in relation to the standard coordinate system $C_s^n$ is given as $p_q = \mathbf{T}q_1$, $q_1 \in Z^n \wedge p_q \in Q^n$, where $q_1$ is an expansion of $q$ as follows: $\forall k \notin \beta \wedge k \in \{1, \ldots, n\}$, fill the $k$-th position of $q_1$ with zeros, then fill the rest positions by the elements of $q$. In other words, let $p'_q \in Q^m \wedge p'_q = D^{-1}(\beta)\, q$, we can expand $p'_q \in Q^m$ to $p_q \in Q^n$ by filling the $k$-th positions with 0, $\forall k \in \{1, 2, \ldots, n\} \wedge k \notin \beta$, and the other positions are filled by the elements of $p'_q$.

Since the $k$-th elements of $p_q \in Q^n$ are 0, $\forall k \in \{1, 2, \ldots, n\} \wedge k \notin \beta$, and the relation $q = D_j(\beta)p'_q, p'_q \in Q^m$, thus we have $q = D_j p_q$, $p_q \in Q^n$. This means that the dependency $W(p_q) \leftarrow U(Dp)$ exists. From lemmas 3.1 and 3.2, and $W(p) \leftarrow U(Dp)$, we have $D(p - p_q) = 0$. Therefore, when $D(\beta)$ is non-singular, the transformation T and the coordinate system constructed above can be used to find the embedding functions so that null space propagation can be used for dependence localization. ■

It should be pointed out here that if $C_j^m$ is allowed to be defined as a function of more than $m$-base vectors from $C_s^n$, then the number of possible coordinates is infinite.

**Example[3.6]:** Let an ADIO be defined as $y(i, j) \leftarrow x(2i - j, 2j)$. The index mapping matrices are specified as:

$$D_y = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, D_x = \begin{bmatrix} 2 & -1 \\ 0 & 2 \end{bmatrix} \tag{3.18}$$

The dependencies of this ADIO cannot be localized using existing methodologies. Its DAG under a single coordinate system is given as in Figure 3.6. Apparently, this DAG has non-uniform dependencies and it can not be mapped on regular array architectures.

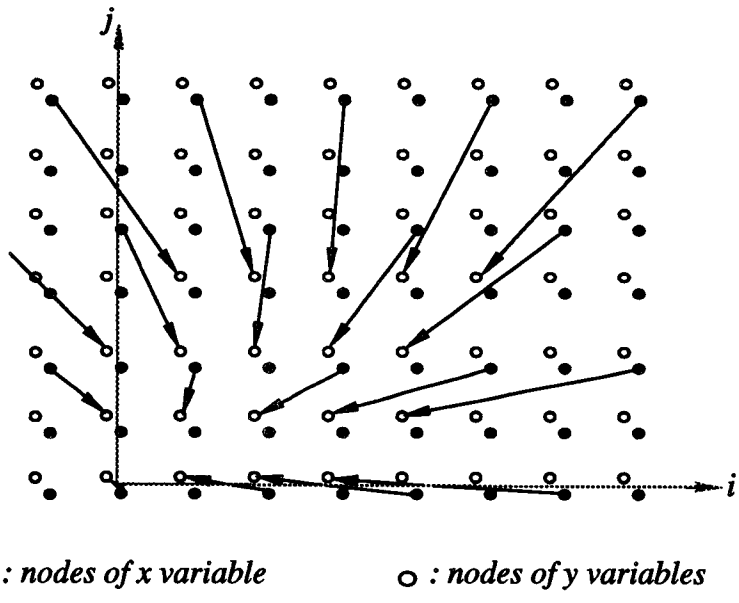


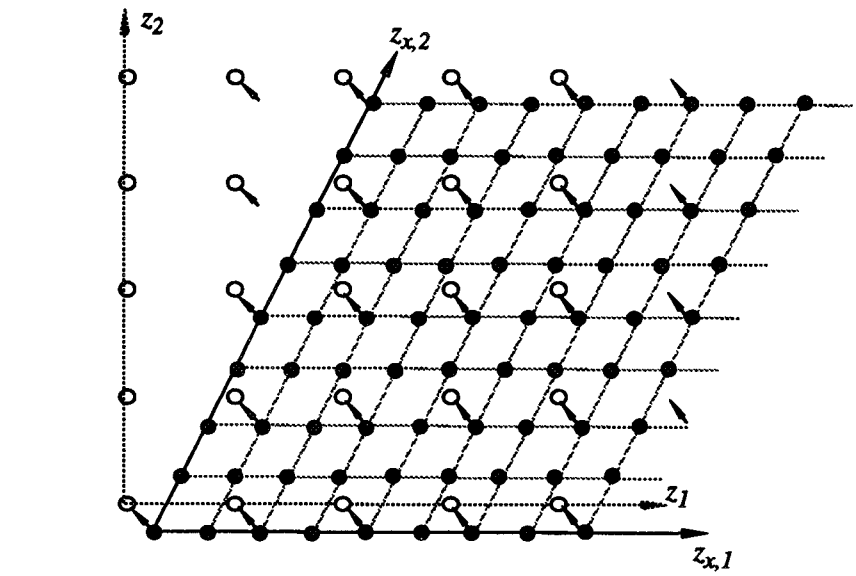

**Figure 3.6:** Dependence Acyclic Graph under a single coordinate system.

---

This ADIO has full rank index mapping matrices. Following the procedure of case-I the index space of variable $y$ is defined relative to the standard coordinate. However, variable $x$ must have its own coordinate system $C_x^2$. The transformation $\mathbf{T} = D_x^{-1}$ is given as:

$$\mathbf{T} = \begin{bmatrix} 1/2 & 1/4 \\ 0 & 1/2 \end{bmatrix} \Rightarrow C_x^2 = \{[1/2\ 0], [1/4\ 1/2]\} \tag{3.19}$$

The new dependence structure under MCS system is given in Figure 3.7. Note that there are many elements of $x$-array which have no dependence vectors originated from them, this means that they are not used in the computation. ■




● : *nodes of x variable* o : *nodes of y variables*

**Figure 3.7:** Directed Acyclic Graph under an MCS system.

---

**Example[3.7]:** Consider a three dimensional filter as:

$$U_o(i,j,k) = f[U_1(2i-k, k-j), U_2(i+2j-k, i+2k), U_3(i-j+k)$$

With the computational domain $\Omega = \{p|\ (0 \le i,j,k \le N)\}$. To embed this into the computational domain for $0 \le i,j,k \le N$. Let $p = [i\ j\ k]^T$ be the index point. This is the I/O specification (ADIO) and the index mapping matrices are:

$$D_o = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, D_1 = \begin{bmatrix} 2 & 0 & -1 \\ 0 & -1 & 1 \end{bmatrix}, D_2 = \begin{bmatrix} 1 & 2 & -1 \\ 1 & 0 & 2 \end{bmatrix}, D_3 = \begin{bmatrix} 1 & -1 & 1 \end{bmatrix}$$

The embedding of $U_o(p)$, $\sigma_o : Z^3 \rightarrow Z^3$ is given as $\sigma_o(i,j,k) = (z_1, z_2, z_3)$, where $i, j, k$ are not relative to any specific coordinate system, but, $z_1, z_2, z_3$ are in the standard coordinate $C_s^3$.

For $U_o(p) \leftarrow U_1(D_1 p)$, one possible embedding is $\sigma_1 : Z^2 \rightarrow Q^3$ which is defined as $\sigma_1(i,j) = (z_1, z_2, 0)$, that is, $U_1(q)$ is embedded into the 2-dimensional subspace $(z_1, z_2)$ with $z_3 = 0$ in $C_s^3$ as shown in Figure 3.8. Since the matrix $D_1$ is not unimodular, its dependencies are linear functions of the indices and grow with the increase of the indices at the rate of the eigenvalues in corresponding directions. A partial dependence structure of this dependence relation is given in Figure 3.8.

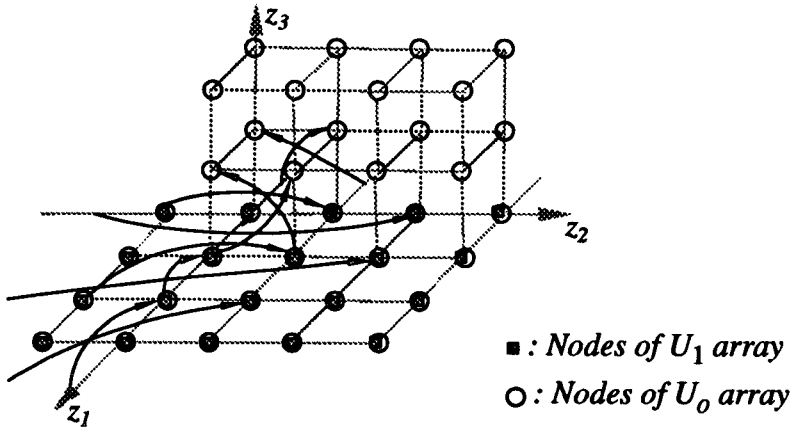



**Figure 3.8:** The dependencies of $U_o(p) \leftarrow U_1(D_1 p)$ under a single coordinate system. The $U_1$ array is embedded at the subspace $z_3 = 0$.

---

MCS representation: Following the procedure outlined above:

Step 1. Select the subspace to embed $U_1$. Let this subspace be $(z_1, z_2)$.

Step 2. Select the data flow direction. This is equivalent to the selection of the origin for the new coordinate system $C_1^2$. Let $C_1^2$ have the same origin as the standard coordinate.

Step 3. Construct the submatrix $D_1(\beta_1)$. Since the subspace is selected as $(z_1, z_2)$, therefore, $\beta_1 = \{1, 2\}$. Thus:

$D_1(\beta_1) = \begin{bmatrix} 2 & 0 & -1 \\ 0 & -1 & 1 \end{bmatrix}(\beta_1) = \begin{bmatrix} 2 & 0 \\ 0 & -1 \end{bmatrix}$, and the inverse of $D_1(\beta_1)$ is given as:

$$D_1^{-1}(\beta_1) = \begin{bmatrix} 1/2 & 0 \\ 0 & -1 \end{bmatrix}$$

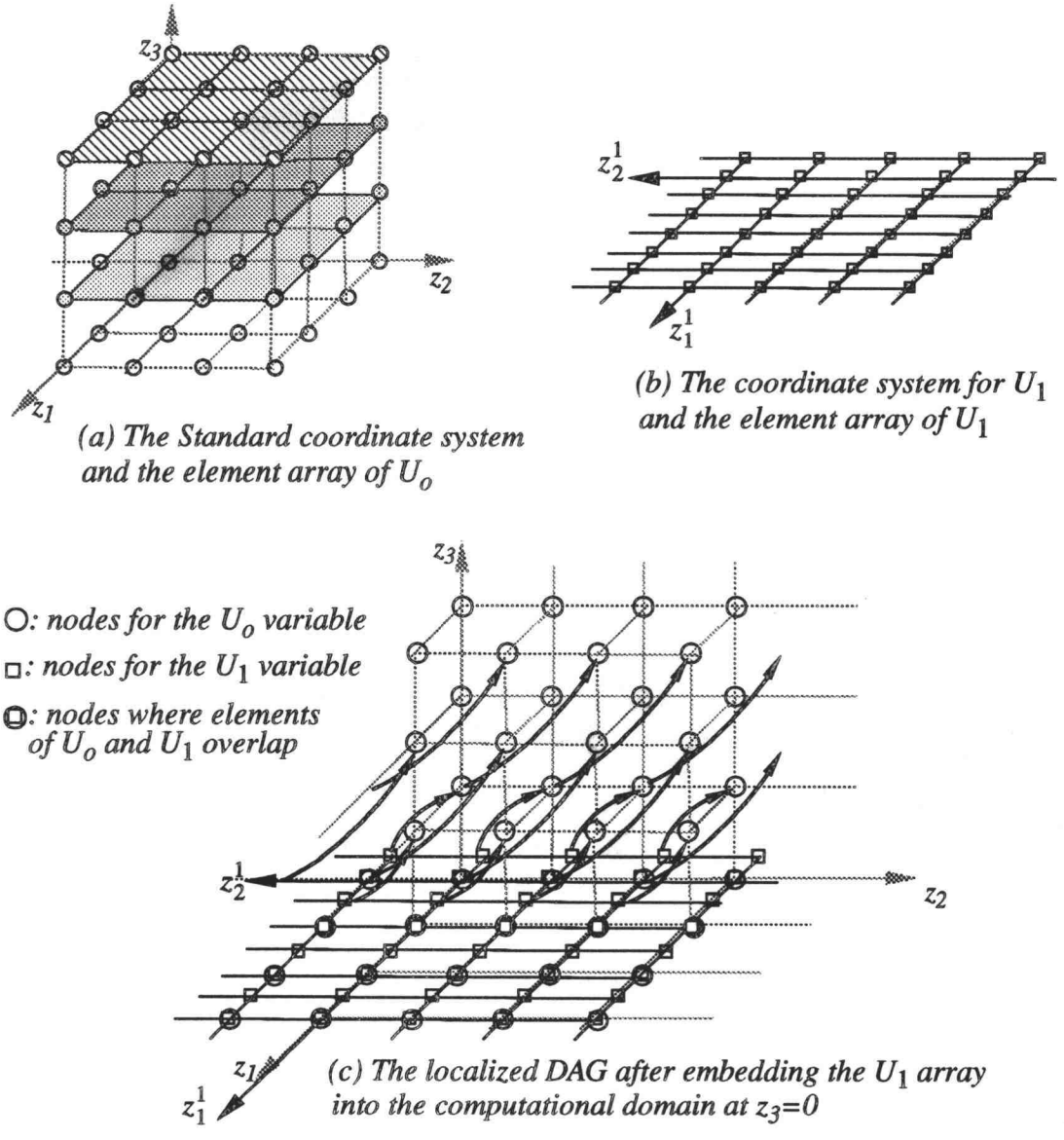
Step 4: Construct the transformation $\mathbf{T}_1 \in Z^{3\times 3}$. First, fill the third column and the third row of $\mathbf{T}_1$; then fill columns one and two with the columns of the matrix $D_1^{-1}(\beta_1)$. Thus the transformation $\mathbf{T}_1$ is given as:

$$\mathbf{T}_1 = \begin{bmatrix} 1/2 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{3.20}$$

with this transformation, the coordinate system $C_1^2$ is defined as $C_1^2 = \mathbf{T}_1 C_s^3$ with basis $\{\mathbf{u}_1^1, \mathbf{u}_2^1\}$; where $\mathbf{u}_1^1 = [1/2\ 0\ 0]^T$, $\mathbf{u}_2^1 = [0\ -1\ 0]^T$. $C_1^2$ is shown in Figure 3.9b. The superscripts of $\mathbf{u}_1^1, \mathbf{u}_2^1$ denote that these base vectors are associated with the first coordinate system $C_1^2$ for variable $U_1$. Using $C_1^2$, the embedding is:

$$\sigma_1(i,j) = \begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ 0 \end{bmatrix}\begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} 1/2 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = (\frac{z_1}{2}, -z_2, 0), for \begin{bmatrix} i \\ j \end{bmatrix} \rightarrow \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} \tag{3.21}$$

The dependence graph is partially given in Figure 3.9c where the global broadcasting dependencies have been removed by the null space propagation. Figure 3.10 shows this DAG with different layers.




*(a) The Standard coordinate system and the element array of $U_o$*

*(b) The coordinate system for $U_1$ and the element array of $U_1$*

*(c) The localized DAG after embedding the $U_1$ array into the computational domain at $z_3$=0*

**Figure 3.9:** The dependencies of $U_o(p) \leftarrow U_1(D_1 p)$ under a MCS system. The axes of $C_s^3$ are labeled as $(z_1, z_2, z_3)$ and of $C_1^2$ for $U_1$ as $(z_1^1, z_2^1)$.

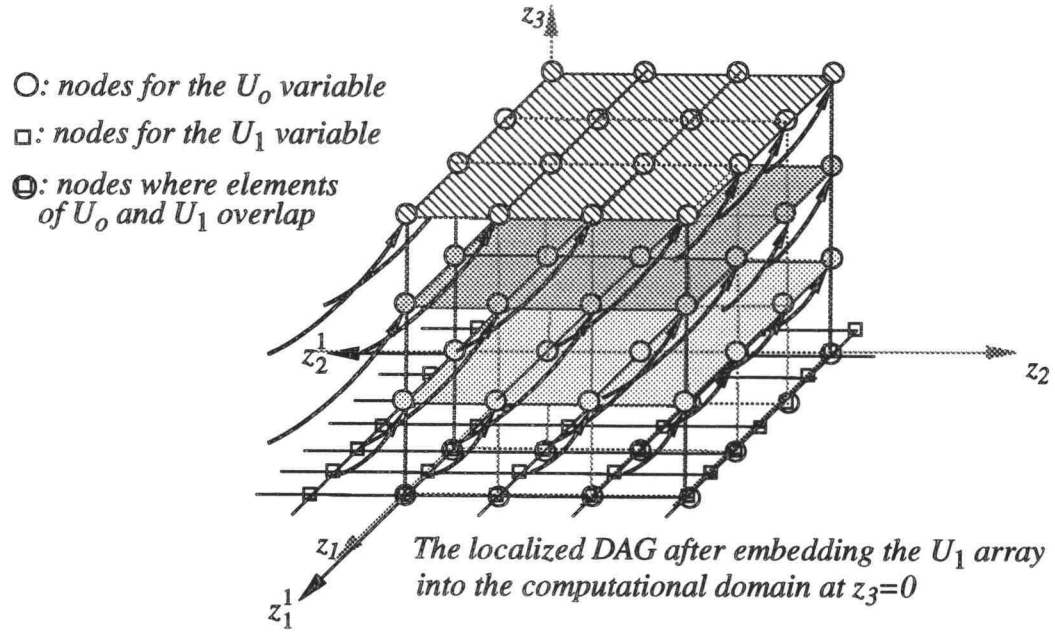



**Figure 3.10:** The DAG showing the dependencies at different layers.

To complete the dependence analysis, it is necessary to define the coordinate systems for variables $U_2$ and $U_3$. Let the subspace for embedding $U_2$ be the $(z_2, z_3)$ and the subspace for $U_3$ be the $(z_3)$. Correspondingly, following the same procedure, the coordinate systems for $U_2$ and $U_3$ are defined by the transformations $\mathbf{T} = \{\mathbf{T}_2, \mathbf{T}_3\}$:

$$\mathbf{T}_2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1/2 & -1/2 \\ 0 & 0 & 1 \end{bmatrix}, \mathbf{T}_3 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.22}$$

$$\begin{aligned} C_2^2 &= \{\mathbf{u}_1^2, \mathbf{u}_2^2\}, \mathbf{u}_1^2 = [0\ 1/2\ 0], \mathbf{u}_2^2 = [0\ -1/2\ 1] \\ C_3^1 &= \{\mathbf{u}_1^3\}, \mathbf{u}_1^3 = [0\ 0\ 1] \end{aligned} \tag{3.23}$$

These coordinate systems are illustrated in Figure 3.11. For clarity, each coordinate system is illustrated separately from $C_s^3$. ■

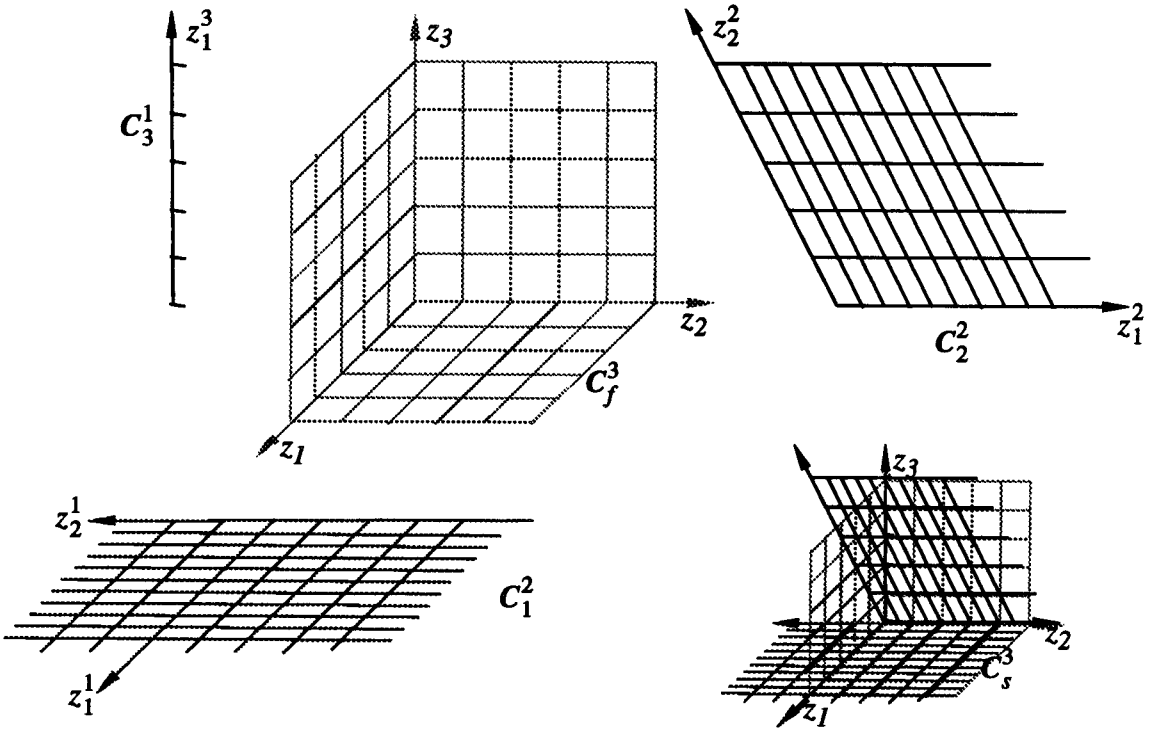



**Figure 3.11:** A possible 3-coordinate systems defined in $R^3$.

To summarize, to derive a regular dependence structure from a given algorithm, we need to propagate the output dependencies using the intermediate variable $W(p)$ into uniform dependencies and transform the dependencies between $W(p)$ and the input variables into uniform dependencies.

Assume that for each variable $U_j(q)$ in a given ADIO, there is one and only one index mapping matrix $D_j$ associated with it. Based on this assumption, the following is

developed for constructing an MCS system for such ADIOs. This assumption is too restrictive for many applications. It will be removed in next section.

Let $m_j = rank(D_j)$ ; the coordinate system for variable $U_j$ be $m_j$-dimensional; and $V$ be the number of variables in an ADIO. The construction of an MCS system for a given ADIO is described by Procedure 3.1:

**Procedure 3.1**:

For $1 \le j \le V$, DO

If $m_j = n$, *for* $l = 1$ *to* $n$, compute basis $\mathbf{u}_l^j = D_j^{-1}\mathbf{e}_l$.

else

If $m_j < n$, DO

select an $m_j$-subspace to embed the variable $U_j$ to the computational domain by selecting $\beta$. $D_j(\beta)$ must be nonsingular;

For $l = 1$ *to* $m_j$, DO

compute basis $\mathbf{u}_l^j = D_j(\beta)^{-1}\mathbf{e}_l^m; l \in \beta$; where $\mathbf{e}_l^m$ denotes the $l$-th base vector of the standard coordinate system with the other $k$-elements been deleted, where $k \in \{1, 2, \ldots, n\} \wedge k \notin \beta$.

End

End

End

In the above procedure we have restricted ourselves to the selection of coordinate systems which can be defined by $m$ base vectors and ruled out the use of other $(n-m)$ base vectors. By doing so, we have reduced the number of possible coordinate systems to:

$$\tilde{N}_n^{m_j} = \frac{n(n-1)\ldots[n-(m_j-1)]}{1 \cdot 2 \cdot 3 \cdot \ldots \cdot m_j} \tag{3.24}$$

This number is derived from the possible combinations of $m$ out of $n$ elements. For most algorithms, such an MCS system would be adequate to present algorithms with regular dependence structures, using the null space propagation. However, if there are two dependence relationships associated with the same variable $U_j$ ($U_j(D_{j,1}p)$ and $U_j(D_{j,2}p)$, where $D_{j,1} \neq D_{j,2}$), which $D_j$ should we use in Procedure 3.1? What are the characteristics of an ADIO which make it possible to be transformed into regular specifications using null space propagation? These questions are investigated in next section.

## 3.4 CONVERTIBILITY

This section studies if a given ADIO can be transformed into a system of specifications with regular dependence structures. First, the sufficient condition for convertibility is provided and then the necessary and sufficient condition is formulated.

### 3.4.1 Motivations

This section provides some motivation examples for removing the assumption made in last section, i.e., each variable $U_j$ is associated with one and only one index mapping matrix $D_j$. For many applications, this assumption does not hold. Therefore, it is necessary to remove this assumption to allow the synthesis of a larger class of algorithms.

**Example[3.8]:** A Lyapunov matrix equation [Che84, Roy88] is given by: $AX + XB = C$ where $A$ is an $n \times n$ non-singular lower triangular matrix, $B$ is an $n \times n$ non-singular upper triangular matrix. The objective is to solve the $n \times n$ matrix $X$. the above equation can be represented in detail as:

$$for \quad 1 \leq i,j \leq n$$

$$c(i,j) = \sum_{k=1}^{i} a(i,k)\,x(k,j) + \sum_{k=1}^{j} b(k,j)\,x(i,k)$$

This equation can be easily represented as an ADIO defined as follows:

$$for \quad 1 \le i,j \le n$$

$$x(i,j) = \frac{\left(c(i,j) - \sum_{k=1}^{i-1} a(i,k)\,x(k,j) + \sum_{k=1}^{j-1} b(k,j)\,x(i,k)\right)}{a(i,i) + b(j,j)}$$

The computational domain of this ADIO is $\Omega = \{p \mid 1 \le i,j \le n,\ 1 \le k < i,j\}$. To the index mapping matrices are given as:

$$\begin{aligned} D_x &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, D_c = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, D_{a1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, D_{x1} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \\ D_{b1} &= \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, D_{x2} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, D_{a2} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}, D_{b2} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix} \end{aligned} \tag{3.25}$$

The variables $x$, $a$ and $b$ have multiple index mapping matrices.

**Example[3.9]:** The algebraic path problem (APP):

$$f(i,j,k) = \begin{cases} a(i,j), & k = 0 \\ f(i,j,k-1), & i = j = k \\ f(i,k,k) \otimes f(i,j,k-1), & i = k \wedge j \neq k \\ f(i,j,k-1) \otimes f(k,j,k), & j = k \wedge i \neq k \\ f(i,j,k-1) \oplus f(i,k,k) \otimes f(k,j,k-1), & i \neq k \wedge j \neq k \end{cases} \tag{3.26}$$

The index mapping matrices are given as:

$$D_a = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, D_{f,1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, D_{f,2} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}, D_{f,3} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Another example in which a variable has more than one index mapping matrix is the Toeplitz factorization algorithm [DI86]. The procedure provided in last section apparently cannot be used to construct MCS systems for these algorithms. For such algorithms, can

the MCS technique be used to localize their dependencies using the null space propagation?

### 3.4.2 Sufficient condition

Using MCS, dependencies among $U_j$ and $U_k$ for $j \neq k$ are not important to the analysis of the dependence structure because each of them may be indexed in relation to different coordinate systems. However, if there are two dependencies for a given variable as $U_j(D_{j,1}p)$ and $U_j(D_{j,2}p)$, the properties of every linear part is important to the others. Let $\Delta_j$ be the set of index mapping matrices for $U_j(q)$ as $\Delta_j = \{D_{j,1}, D_{j,2}, \ldots\}$:

**Definition 3.5:** *Set of Index Mapping Matrices* $\Delta_j$: The set of index mapping matrices $\Delta_j$ is defined as $\Delta_j = \{D_{ji,l} \mid \forall l \Rightarrow U_j(D_{ji,l}p + d_{ji,l})$ appears in the ADIO$\}$, i.e., the elements of $\Delta_j$ are the distinct linear parts of all index mapping functions associated with $U_j$ in an ADIO. $|\Delta_j|$ denotes the cardinality of $\Delta_j$, or the number of elements in set $\Delta_j$. ■

For example, for $U_1(i,j) = f[\ldots, U_2(3i-j), U_2(-3i+j), \ldots]$, then these sets are defined as $\Delta_1 = \{\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\}$ and $\Delta_2 = \{\begin{bmatrix} 3 & -1 \end{bmatrix}, \begin{bmatrix} -3 & 1 \end{bmatrix}\}$.

For a given $n$-dimensional ADIO with $V$ distinct variables $U_j$, $1 \leq j \leq V$, their corresponding sets of index mappings are $\Delta_j$, $1 \leq j \leq V$.

**Theorem 3.2:** *(Sufficient condition for RMCS convertibility)*: Given a system of $n$-dimensional ADIO with $V$ distinct variable arrays $U_j$, $1 \leq j \leq V$. Let its $i$th equation be defined as:

$$\begin{aligned} &U_i(D_i p + d_i) = \\ &\quad f[U_i(D_{ii}p + d_{i,j}), U_j(D_{ji,1}p + d_{ji,1}), U_j(D_{ji,2}p + d_{ji,2}), \ldots] \end{aligned} \tag{3.27}$$

Let $\Delta_j$, $1 \le j \le V$ be the sets of index mapping matrices and each set have $|\Delta_j|$ elements $D_{ji,k} \in Z^{m_j \times n}$, $1 \le k \le |\Delta_j|$. $\forall j$, $1 \le j \le V$, if the following condition is satisfied:

$$\begin{aligned} &\exists \beta \subseteq \{1, 2, \ldots, n\} \wedge (|\beta| = m_j) \wedge (\forall l, 2 \le k \le |\Delta_j|) \\ &\qquad \Rightarrow (D_{ji,1}(\beta))^{-1} D_{ji,k}(\beta) = I \end{aligned} \tag{3.28}$$

then the ADIO is RMCS convertible.

**Proof.** The correctness of this theorem is apparent because if we construct a transformation $\mathbf{T}_j$ (or the coordinate system $C_j^{m_j}$) from $D_{j,1}(\beta)$ for each variable $U_j$, from theorem 3.1, we know that the dependencies of $U_j$ can be localized using the null space propagation. Each index mapping function may have its own null space. Therefore, the elements of $U_j$ may be propagated along $|\Delta_j|$ directions each direction is determined as $\aleph(D_{j,k})$; $1 \le k \le |\Delta_j|$. ■

From this theorem, for a given algorithm with $V$ variables, if $\forall j$, $1 \le j \le V$, $|\Delta_j| = 1$, then the algorithm can be represented in specifications with regular dependence structures. However, this is only a special case depicted by Eq.(3.26). The class of algorithms defined in Eq.(3.26) is characterized by one property, that is, these algorithms can be transformed to regular specifications using one coordinate system for each variable. In other words, to present these algorithms as regular specifications, each variable in a given ADIO needs only one coordinate system to define its index space. This is not always true. In the more general class of algorithms, the index space of each variable may need to be partitioned and defined relative to different coordinate systems.

**Example[3.10]:** An IIR filter is defined as:

$$y(i) = \sum_{0}^{3} a(j)\, x(i-j) + \sum_{k=1}^{3} b(j)\, y(i-j) \tag{3.29}$$

$$
\begin{aligned}
\Gamma_y(p) &= [1\ 0]p, & D_y &= [1\ 0] \\
\Gamma_{xy}(p) &= [1\ -1]p, & D_{yy} &= [1\ -1] \\
\Gamma_{xy}(p) &= [1\ -1]p, & D_{xy} &= [1\ -1] \\
\Gamma_{ay}(p) &= [0\ 1]p, & D_{ay} &= [0\ 1] \\
\Gamma_{by}(p) &= [0\ 1]p, & D_{by} &= [0\ 1]
\end{aligned}
\tag{3.30}
$$

The sets of index mapping matrices are defined as:

$$
\Delta_y = \{[1\ 0], [1\ -1]\}, \Delta_x = \{[1\ -1]\}, \Delta_a = \{[0\ 1]\}, \Delta_b = \{[0\ 1]\} \tag{3.31}
$$

All other variables have only one linear part in their sets except variable $y$. Therefore, the convertibility of Eq.(3.27) depends on the dependence properties of variable $y$. $\Delta_y$ has two distinct linear parts which satisfy Eq.(3.26) if $\beta = \{1\} \subseteq \{1, 2\}$ is selected, thus $D_y(\beta) = D_{yy}(\beta) = [1]$. Therefore, Eq.(3.27) can be transformed to regular specifications using the null-space propagation.

Let the subspaces for embedding be: $(z_1, 4)$ for $y$; $(z_1, 0)$ for $x$; $(0, z_2)$ for $a$ and $(0, z_2)$ for $b$. The transformations are given as $\mathbf{T}_y = \mathbf{T}_x = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$, $\mathbf{T}_a = \mathbf{T}_b = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$ with the origins $O_y = (0, 4)$, $O_x = (0, 0)$, $O_a = (0, 0)$, and $O_b = (0, 4)$. The localized DAG is given in Figure 3.12. ■

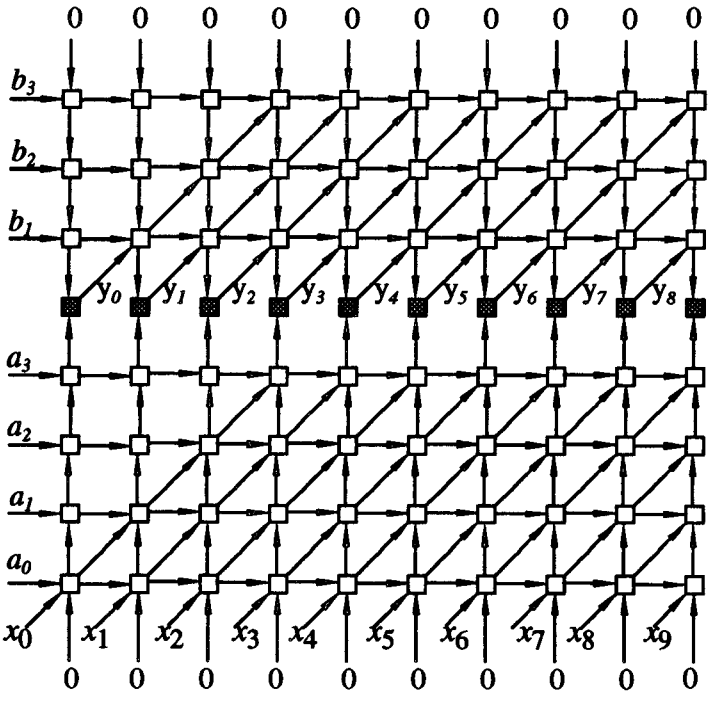



**Figure 3.12:** The localized DAG for the IIR filter algorithm.

---

When an ADIO satisfies Eq.(3.26), it can be transformed to a system of RMCS using the null space propagation. However, if Eq.(3.26) can not be satisfied, can a given ADIO be localized using the null space propagation? This problem is studied in next section.

### 3.4.3 Sufficient and necessary condition

When a given algorithm fails to satisfy Eq.(3.26), one coordinate system for each variable is insufficient to transform it into a regular specification using the null-space propagation. Then, can it be transformed to a regular specification using the null space propagation? This is investigated in this section.

This following example illustrates the case where the use of a single coordinate system for a variable is no longer sufficient to present it with uniform dependencies, and more than one coordinate systems must be used to define its index spaces in different regions in order to transform it into RMCS.

**Example[3.11]:** Let an ADIO be $U_1(i,j) = f[..., U_2(3i-j), U_2(-3i+j), ....$ The index mapping matrices are $D_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, D_{2,1} = [3\ -1], D_{2,2} = [-1\ 1]$ and the sets of index mapping matrix are defined as:

$$\Delta_1 = \{\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\}, \Delta_2 = \{[3\ -1], [-3\ 1]\} \tag{3.32}$$

In this example, $\Delta_1$ has only one element which is the identity matrix, therefore, its dependencies can be expressed regularly. Actually, the index space of variable $U_1$ can be defined in relative to the standard coordinate system since $D_1$ is the identity matrix. However, $\Delta_2$ has two elements and the condition in Eq. (3.24) is not satisfied, thus the algorithm can not be transformed into a regular specification if only one coordinate system is used to define the index space for variable $U_2$. Let $\beta = \{1\}$, then the coordinate system $C_2^1$ constructed from $D_{2,1}(\beta) = [3]$ is defined by the basis $\mathbf{u} = [1/3\ 0]^T$. If this coordinate system $C_2^1$ is used to define the entire index space for $U_2$, the resultant dependence structure is illustrated in Figure 3.9, which is not regular.

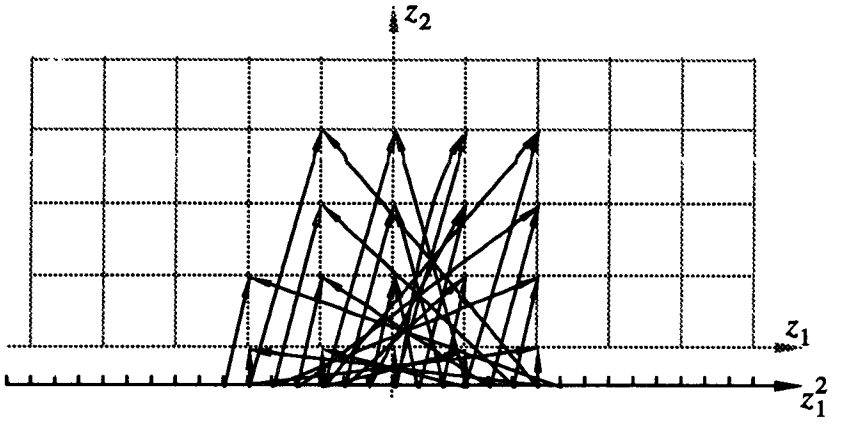



**Figure 3.13:** When condition (3.26) is not satisfied, one coordinate system for a variable is not sufficient to present an ADIO with regular dependencies.

---

Note that even though $(D_{2,1}(\beta))^{-1}D_{2,2}(\beta) \neq I$, $((D_{2,1}(\beta))^{-1}D_{2,2}(\beta))^2 = I$. This means that $(D_{2,1}(\beta))^{-1}D_{2,2}(\beta)$ is a 2-root of $I$. This property is important since it has been proved that if a matrix is a root of $I$, then its dependencies can be represented as quasi-regular dependence structures [Yaa88]. By quasi-regular it is meant that the dependencies are regular over the entire domain except possible at the boundaries. For these algorithms, the strategy is to use the "general fold" method to transform them into quasi-regular specifications, which is equivalent to the use of different coordinate systems to define the index spaces in different regions.

Instead of defining the entire index space relative to the coordinate system $C_2^1$, let the index space of $U_2$ be partitioned into two regions each to be defined in its own coordinate system. How to partition the computational domain for each variable will be investigated in next section. For this example, the domain is partitioned as into two regions, either $\Omega_1 = \{(i,j) | 0 \leq i\}$ and $\Omega_2 = \{(i,j) | 0 > i\}$, or $\Omega'_1 = \{(i,j) | 0 \leq j\}$ and $\Omega'_2 = \{(i,j) | 0 > j\}$ respectively, depending on how $\beta$ is selected to construct the sub-

matrices for constructing the coordinate systems. When $\beta = \{1\}$, $D_{2,1}(\beta) = [3]$ and $D_{2,2}(\beta) = [-3]$ are selected as the constructing submatrices, then the first partitioning strategy must be employed, i.e., $\Omega_1 = \{(i,j) | 0 \leq i\}$ and $\Omega_2 = \{(i,j) | 0 > i\}$. The corresponding coordinate systems for $U_2$ can be constructed by using Procedure 3.1, and their basis are given as $\mathbf{u}_{2,1} = \{[1/3\ 0]\}$ and $\mathbf{u}_{2,1} = \{[-1/3\ 0]\}$ for regions $\Omega_1$ and $\Omega_2$ respectively. If, on the other hand, $\beta = \{2\}$, then $[1]$ and $[-1]$ are selected as the constructing submatrices, the coordinate systems are then defined by the basis as $\mathbf{u}_{2,1} = \{[0\ 1]\}$ and $\mathbf{u}_{2,1} = \{[0\ -1]\}$ respectively for regions $\Omega'_1$ and $\Omega'_2$. The corresponding partitioning and the coordinate systems are illustrated in Figure 3.10.

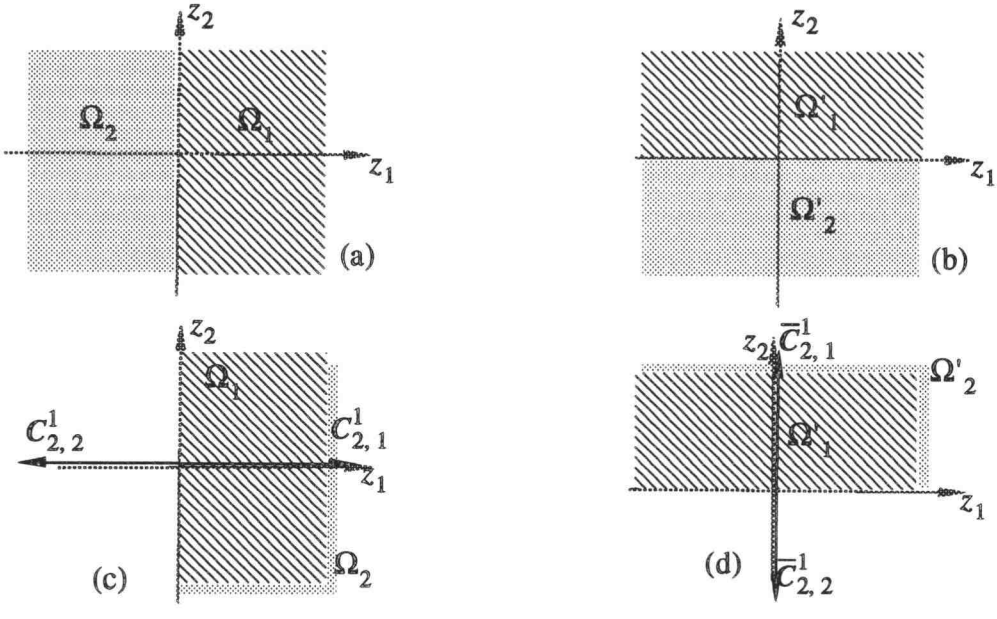



**Figure 3.14:** Different partitioning of the computational domain and the effect of using different coordinate systems for different regions.

In Figure 3.14a and b, the computational domain is partitioned into two regions but it is defined in one coordinate system. In Figure 3.14c and d, these regions are defined in different coordinate systems whose actual effects are folding these regions along the $z_2$- and $z_1$-axis respectively. It should be pointed out that not only the domain for variable $U_2$ is so partitioned and defined, but also the domains of the other variables. If the first partitioning strategy is employed, the resultant dependence structure of the ADIO is as given in Figure 3.15. The new dependence structure is regular over the entire domain. ■

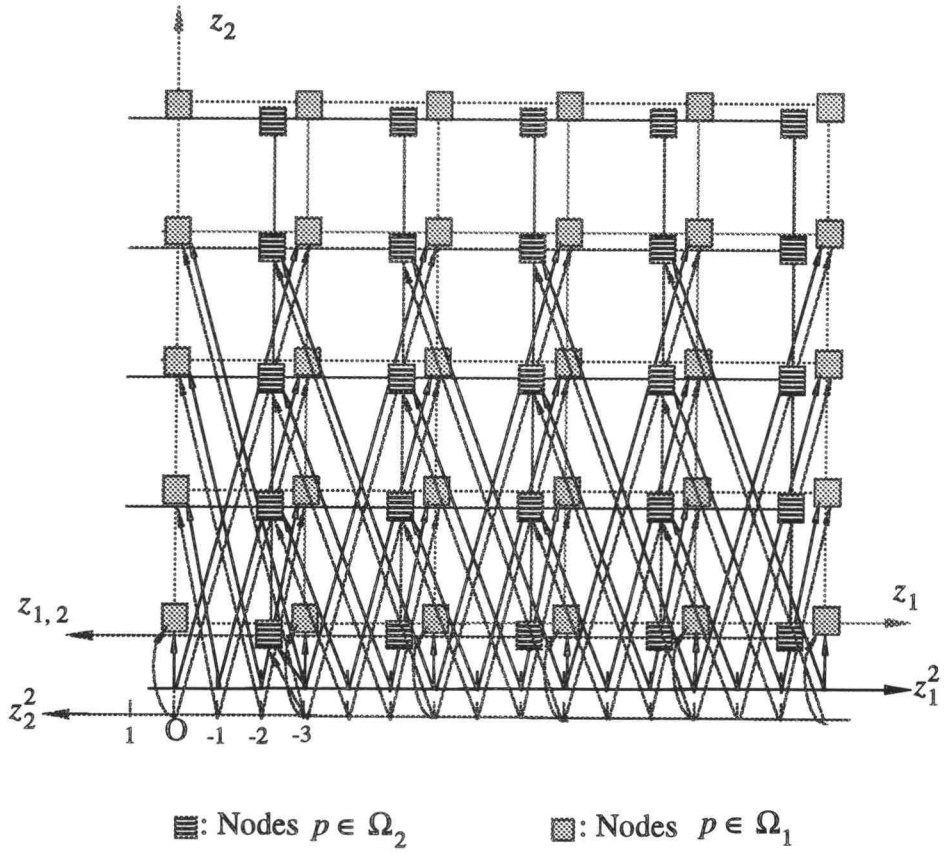



**Figure 3.15:** Two coordinate systems are used to define the index regions $\Omega_1$ and $\Omega_2$ of the index space for each variable.

The question now is that, for a given ADIO, what are the conditions which it should satisfy to be transformed into a system of regular specifications or quasi-regular specifications?

Let $D_{j,1}, D_{j,2} \in \Delta_j$ be two different linear parts of the index mapping functions associated with variable $U_j$, and both of them have the same rank $m = rank(D)$, this is reasonable because they both define the index vectors for the same $m$-dimensional variable array. However, if the rank of a matrix, say $D_{j,2}$, is less than $m$, then $D_{j,2}$ should not be considered and $D_{j,1}$, $(\rho(D_{j,1}) = m)$ should be used, because only the matrix of rank-$m$ can be used to index an $m$-dimensional array. If all the index mapping matrices associated with variable $U_j$ have rank less than $m$, then $U_j$ should not be an $m$-dimensional array.

If $(D_{ji,1}(\beta))^{-1} D_{ji,2}(\beta) \neq I$, but the condition $((D_{ji,1}(\beta))^{-1} D_{ji,2}(\beta))^{L_{ji}} = I$ is satisfied for some integer $L_{ji}$, the algorithm still can be transformed into a new specification with regular dependencies in the entire domain, except possibly at boundaries of the those partitioned regions of the computational domain. The computational domain is partitioned into $L_{ji}$ regions. This case is similar to the "general folding" technique proposed by Yaacoby [Yaa88] but it is more general because only the index mapping matrices associated with the same variable $U_j$ are considered. But in [Yaa88], all index mapping matrices in an equation must be considered. However, the method developed in [Yaa88] for computing a fundamental region can be employed to compute a partitioning for a variable. This will be discussed later for the computation of partitioning.

For a given ADIO, whether it can be transformed to a regular specification using null space propagation is determined as follows:

**Theorem 3.3:** *Necessary and Sufficient condition for RMCS convertibility:* Let an ADIO have $V$ variables as defined in Eq.(3.25), $D_{ji,1}, D_{ji,2} \in \Delta_j$ be two different linear parts

associated with variable $U_j$ and their rank be $m_j = rank(D)$. $\forall j, 1 \le j \le V$, if the following condition is satisfied:

$$\begin{aligned} &\exists \beta \in \{1, \ldots, n\} \wedge (|\beta| = m_j) \wedge (\exists L_{ji}) \Rightarrow \forall k, 2 \le k \le |\Delta_{ji}|, \\ &((D_{ji,1}(\beta))^{-1} D_{ji,2}(\beta))^{L_{ji}} = I \wedge ((D_{ji,1}(\beta))^{-1} D_{ji,k}(\beta) \quad is \ integral) \end{aligned} \tag{3.33}$$

where $L_{ji}$ is an integer number, then the ADIO is RMCS convertible. Its dependencies can be localized using $L_{ji}$ coordinate systems to define the index space of $U_j$. And the computational domain $\Omega_j$ is partitioned into $L_{ji}$ polytopes $P_{jl}$, $1 \le l \le L_{ji}$.

**Proof.** Assume that we have already constructed a coordinate system $C_{j1}^m$ for $U_j$ based on $D_{ji,1}(\beta)$, therefore the dependencies related to $D_{ji,1}$ are uniform. As mentioned before, we restrict ourselves to the embeddings of an $m$-dimensional array into an $m$-dimensional subspace in $Q^n$ with the coordinates of other dimensions been set to some constants. For simplicity, let the notations for $D_{ji,1}(\beta)$ and $D_{ji,2}(\beta)$ be denoted as $D_1$ and $D_2$ respectively ($D_1$ and $D_2$ are full rank $m \times m$ matrices). Now, let us study the dependence relation $W(p) \leftarrow U(D_1 p), U(D_2 p)$; $p \in Q^m$.

When $L = 1$, from Thm.3.1, we know that one coordinate system is sufficient to make the dependencies uniform. Assume $D_1 \neq D_2$ and $(D_1^{-1} D_2)^L = I$, for $1 < L$. First, we will show that the use of a single coordinate system $C_{j1}^m$ for the indexing of $U_j$ is not sufficient. Let

$$\begin{aligned} &W(p_1) \leftarrow U(D_1 p_1), U(D_2 p_1) \\ &W(p_2) \leftarrow U(D_1 p_2), U(D_2 p_2); \qquad D_2 p_1 = D_1 p_2 \\ &\qquad\qquad \ldots \\ &W(p_{L+1}) \leftarrow U(D_1 p_{L+1}), U(D_2 p_{L+1}); \quad D_2 p_L = D_1 p_{L+1} \end{aligned} \tag{3.34}$$

The dependence vectors associated with $D_2$ are defined as $(D_2 - I)p_1; (D_2 - I)p_2; \ldots$. The objective is to transform these dependencies to uniform ones, that is

$$(D_2 - I)p_1 = (D_2 - I)p_2 = \ldots = (D_2 - I)p_L = (D_2 - I)p_{L+1} \tag{3.35}$$

From Eq.(3.34), we have

$$\begin{aligned} p_2 &= D_1^{-1}D_2 p_1 \\ p_3 &= D_1^{-1}D_2 p_2 = (D_1^{-1}D_2)^2 p_1 \\ &\ldots \\ p_{L+1} &= D_1^{-1}D_2 p_L = (D_1^{-1}D_2)^L p_1 \end{aligned} \tag{3.36}$$

Eq.(3.35) is translated into

$$(D_2 - I) = (D_2 - I)D_1^{-1}D_2 = \ldots = (D_2 - I)\,(D_1^{-1}D_2)^L \tag{3.37}$$

This equation holds if and only if $D_2 = I$ or $D_1^{-1}D_2 = I$. Following the same procedure, if $D_2 = I$ holds, we can easily derive that $D_1 = I$, which is contradict to our assumption and this case can be easily presented with uniform dependencies. If the condition $D_1^{-1}D_2 = I$ holds, then this is also contradict to the assumption that $D_1^{-1}D_2 \neq I$. Therefore, the use of only one coordinate system is not sufficient to make the dependencies uniform.

In order to have uniform dependencies, Eq.(3.37) must hold. This is possible only if $p_j$, $2 \leq j \leq L$, is embedded to the same physical location as $p_1$. This can be achieved using other coordinate systems $C_{ij}^m$, $2 \leq j \leq L$ to define the index spaces which contain nodes $p_j$, $2 \leq j \leq L$:

$$\mathbf{u}_{ij} = [(D_1^{-1}D_2)^{j-1}]^{-1}\mathbf{u}_{i1}; 1 \le k \le m \tag{3.38}$$

When $p_j$, $2 \le j \le L$, are indexed relative to their own coordinates, their physical locations are located at:

$$[(D_1^{-1}D_2)^{j-1}]^{-1}p_j = [(D_1^{-1}D_2)^{j-1}]^{-1}(D_1^{-1}D_2)^{j-1}p_1 = p_1 \tag{3.39}$$

Substitute Eq.(3.39) into Eq.(3.37), the equation holds and the dependencies are constant. This proves that when $((D_{i,1}(\beta))^{-1}D_{i,2}(\beta))^{L_i} = I$, $L_i$ coordinate systems can be used sufficiently to present the dependencies uniform.

On the other hand, if this condition does not hold, i.e., there is no such an integer $L_i$ such that $((D_{i,1}(\beta))^{-1}D_{i,2}(\beta))^{L_i} = I$, or $L_i \to \infty$, then Eq.(3.24) can not be made uniform, or we need infinite number of coordinate systems to index $U_i$, which is not a interesting case since by then all nodes in the entire domain are reduced to a single point and no parallel computation can be achieved. This proves that $((D_{i,1}(\beta))^{-1}D_{i,2}(\beta))^{L_i} = I$ is a necessary condition to present these dependencies uniform under $L_i$ coordinate systems.

For the dependencies over the entire $n$-dimensional computational domain, $W(p) \leftarrow U(D_{ji,1}p), U(D_{ji,2}p)$; $p \in Z^n$, from lemmas 3.1 and 3.2, nullspaces $\aleph(D_{ji,1})$ and $\aleph(D_{ji,2})$ must be used to propagate a variable to the index points where it is required. Therefore, if $|\Delta_{ji}| = K$, then at least $k$-directional propagation (all along the null spaces $\aleph(D_{ji,k})$, $1 \le k \le K$) must be used. ■

**Remarks:** The conclusion drawn from theorem 3.4 can be considered as the generalization of the "generalized fold" method reported by Yaacoby [Yaa88], where the index spaces of all variables are defined relative to a single coordinate system (implicitly

defined). In contrast, Eq.(3.33) is concerned with each individual variable $U_j$ with its index space defined relative to the coordinate system constructed from $(D_{j,1}(\beta))^{-1}$. On this behalf, if we consider all $D_{j,l}(\beta)$ $(\forall D_{j,l} \in \Delta_j)$ as all the linear parts of dependence mappings in an SARE, then the theorems and procedures developed in [Yaa88] can be employed for the analysis and transformation of the dependencies of variable $U_j$. The number $L_{ji}$ has been determined as $L = lcm(r_j)$ (lcm: least common multiple), where $\sum \phi(r_j) \leq m_j$. Moreover, $L \leq f(m_j)$, where $f: N \rightarrow N$ (Theorem 3.6, p47, [Yaa88]). For $m_j = 1$, the values that $L$ can take are 1, 2. For $m_j = 2$ and 3, the values that $L$ can take are 1, 2, 3, 4, 6. For $m_j = 4$ and 5, the values that $L$ can take are: 1, 2, 3, 4, 5, 6, 8, 10, 12. Note that the number used to determine $L$ in [Yaa88] is the dimension of the algorithm, $n$, which is generally greater than $m_j$. In next section, a procedure will be provided which incorporates some procedures from [Yaa88] for the computation of a partitioning of the computational domain $\Omega_j$.

The following algorithm is a summary of the foregoing discussions. For a given algorithm with $V$ distinct variables, it determines the necessary and sufficient condition for transforming it into an equivalent specification with uniform dependence structures. If the condition in Eq.(3.33) is satisfied, the given algorithm is RMCS convertible; otherwise, the algorithm can not be transformed to RMCS using null space propagation only. To transform it to a regular specification, other directional propagation may be required. This would require additional interconnections for VLSI implementation.

**ALGORITHM 3.1: Decide RMCS Convertibility**

BEGIN

FOR $1 \le j \le V$,

Extract all index mapping matrices $D_{ji}$ from the algorithm, calculate the set $\Delta_j$;

IF $|\Delta_j| > 1$,

Determine if $\exists \beta \in \{1, 2, \ldots, n\}$ and $\exists L_{ji}$ such that $\forall l, 2 \le l \le |\Delta_j|$,

$$((D_{ji,1}(\beta))^{-1} D_{ji,l}(\beta))^{L_{ji}} = I \wedge ((D_{ji,1}(\beta))^{-1} D_{ji,l}(\beta) \; is \; integral)$$

If this condition is not satisfied by any $D_{ji,l}(\beta)$, then the algorithm is not RMCS convertible;

RETURN;

END

END

The algorithm is RMCS convertible;

RETURN;

END

In practical applications, the computational domains are generally not higher than four dimensions while most variables are one- and two-dimensional arrays, such as speech and image signals are 1-dimensional and 2-dimensional variables respectively. This means that most $m_j$ equal to one or two and $\tilde{N}_n^{m_i}$ is generally not greater than 6. Therefore, the computation of this algorithm should not become too complex.

Procedure 3.1 constructs a system of MCS for algorithms which satisfy Eq.(3.26). For algorithms which do not satisfy Eq. (3.26) but satisfy Eq.(3.31), it is necessary to construct multiple coordinate systems for $U_j$. This procedure is given by Procedure 3.2 below:

**Procedure 3.2: Construct MCS for variable $U_j$.**

1. Invoke Procedure 3.1 to compute a system of MCS of the given ADIO.

2. For $1 \le j \le V$, if $|\Delta_j| > 1$, then

   2.1. $\forall l, 2 \le l \le |\Delta_j|$, if $(D_{ji,1}(\beta))^{-1} D_{ji,l}(\beta) = D_l(\beta) \neq I$, invoke Procedure 3.3, compute a partitioning $P_{jl}$ for $\Omega_j$.

   2.2. Corresponding to each region (polytopes) $P_{jl}^{(k)}$, $1 \le k \le L_{ji} - 1$, apply the linear transformation $\mathbf{T}_{j,l} = (D_l(\beta))^{-1}$ to base vectors of the coordinate system $C_j^{m_j}$, which constructs a new coordinate system $C_{j,l}^{m_j}$ for defining the index region $P_{j,l}$.

**Remarks:** The operation of step 2.2 in the above procedure is equivalent to applying the linear transformation $\mathbf{T}_{j,l} = (D_l(\beta))^{-1}$ to each polytopes $P_{jl}^{(k)}$, $1 \le k \le L_{ji} - 1$, which maps all these regions to $P_{jl}^0$. Therefore, this operation is equivalent to the "generalized folding" method given in [Yaa88].

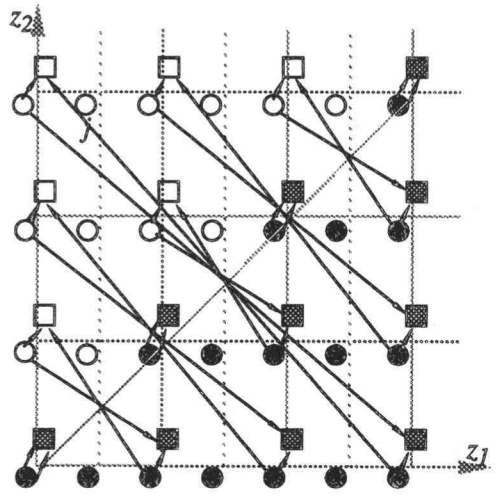
The following example serves to illustrate the application of theorem 3.4 in dependence localization where a variable requires more than one coordinate systems for the definition of its index spaces in order to present its dependencies as uniform ones.

**Example[3.12]:** Let an ADIO be given as $W(p) = f[U(D_{1,1}p), U(D_{1,2}p), \ldots]$, where $D_1 = \begin{bmatrix} 2 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}$, $D_2 = \begin{bmatrix} 0 & 2 & -1 \\ 1 & 0 & -1 \end{bmatrix}$, and $p^T = [i\ j\ k]$. Let $\beta = \{1, 2\} \in \{1, 2, 3\}$ be selected for defining the submatrices, which are given as:

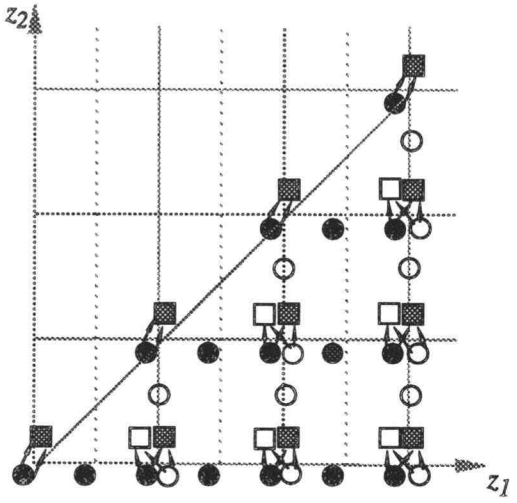
$$D_{1,1}(\beta) = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \text{ and } D_{1,2}(\beta) = \begin{bmatrix} 0 & 2 \\ 1 & 0 \end{bmatrix}$$

To construct an MCS system from these matrices, let $D_{1,1}(\beta)$ be used in Procedure 3.1. Thus we have $D_{1,1}(\beta)^{-1} = \begin{bmatrix} 1/2 & 0 \\ 0 & 1 \end{bmatrix}$ and $((D_1(\beta))^{-1} D_2(\beta))^2 = I$. This means that we can present the dependencies with uniform structure using two coordinates for $U$.

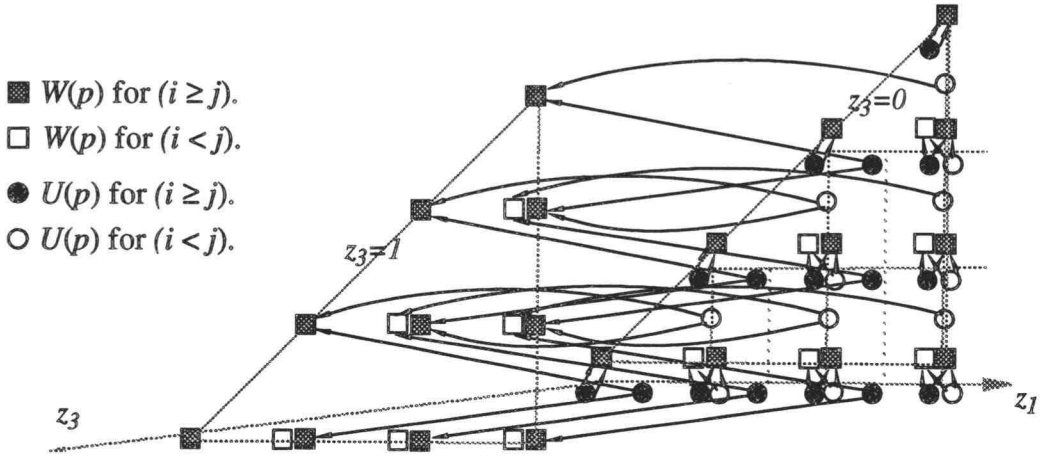
The first coordinate system $C_1^2 = \{[1/2\ 0\ 0], [0\ 1\ 0]\}$ is derived from $D_1(\beta)$. The dependencies of this algorithm indexed under this coordinate system is not uniform as illustrated in Figure 3.16*a* (note that the standard coordinate system is always used as a reference coordinate system, it is also used for indexing *W*). In order to have a uniform DAG, we need to partition the computational domain into two regions using two coordinate systems $C_1^2$ (for the half plane $z_1 \geq z_2$) and $C_2^2$ (for the other half plane $z_1 < z_2$) for the indexing of *U*, where $C_2^2 = \{[0\ 1/2\ 0], [1\ 0\ 0]\}$. Correspondingly, the domain of *W* is also partitioned into two regions and each requires a coordinate system of its own. For the index points of *W* in $z_1 \geq z_2$, $C_s^3$ is used; while if $z_1 < z_2$ then the coordinate system is defined as $C_s^3 = \{[0\ 1\ 0], [1\ 0\ 0], [0\ 0\ 1]\}$. Figure 3.16b shows the dependencies at $z_3 = 0$ while Figure 3.16c shows the dependencies at both $z_3 = 0$ and $z_3 = 1$.




*(a) DAG at $z_3$=0 when U is indexed under $C_1^2$*




*(b) DAG at $z_3$=0 when U is indexed under two coordinate systems $C_1^2$ and $C_2^2$*




*(c) MCS_DAG at $z_3$=0, 1 when U is indexed under $C_1^2$ and $C_2^2$*
*All dependencies are constant vectors*

**Figure 3.16:** The DAGs for Example[3.9] under different MCS systems. (a) The DAG is non-uniform in which the variable $U$ is indexed under $C_1^2$ only at $z_3 = 0$; (b) $C_1^2$ and $C_2^2$ are employed for $U$ ($C_1^2$ for $i \geq j$ and $C_2^2$ for $i < j$) at $z_3 = 0$; (c) DAG at $z_3 = 0$ and $z_3 = 1$.

How to compute a partitioning of the computational domain and construct a system of MCS for a given system of ADIO which satisfies condition Eq.(3.33)? These problems are to be studied in next section.

## 3.5 COMPUTING A PARTITIONING

When a given algorithm dose not satisfy condition Eq. (3.24) but satisfies condition Eq.(3.28), multiple coordinate systems must be employed to define the index space of each variable, each coordinate system is used to define one of several regions of the computational domain. The computational domain is partitioned into $L$ non-overlap polytopes. The computation of a partitioning of the computational domain has been studied by Yaacoby [Yac88], even though the procedures developed were used for computing a fundamental region for SAREs with index mapping matrices $D$ all are roots of $I$, they can be modified to compute a partitioning here for ADIOs.

The following procedure outlines the steps required for computing a partitioning:

**Procedure 3.3: Compute a Partition for $\Omega_j$**

1. Construct $\Delta_j$ from the algorithm.

2. If $|\Delta_j| = 1$, there is only one linear part associated with $U_j$, no partitioning of $\Omega_j$ is required. Return.

3. Determine the $m_j$-subspace (of $n$-space) which $U_j$ is to be embedded in according to specifications, and therefore the corresponding submatrix $D_{ji,1}(\beta)$ from $D_{ji,1} \in \Delta_j$.

4. For $2 \le l \le |\Delta_j|$, construct $D_{ji,l}(\beta)$ and compute $D_l = (D_{ji,1}(\beta))^{-1} D_{ji,l}(\beta)$. If $D_l = I$, compute next submatrix; otherwise, compute as follows:

   4.1 Let $G_l = \{D_i\}_{i=0}^{|G|-1}$ be the group generated by $D_l$, where $D_0 = I$.

   4.2 Construct point $p \in R^{m_j}$ such that $D^T p \neq p$, $\forall D \in G_l - I$.

   4.3 Construct matrix $A_0$, whose $row_i = p^T (D_i - I)$, $\forall D_i \in G_l - I$.

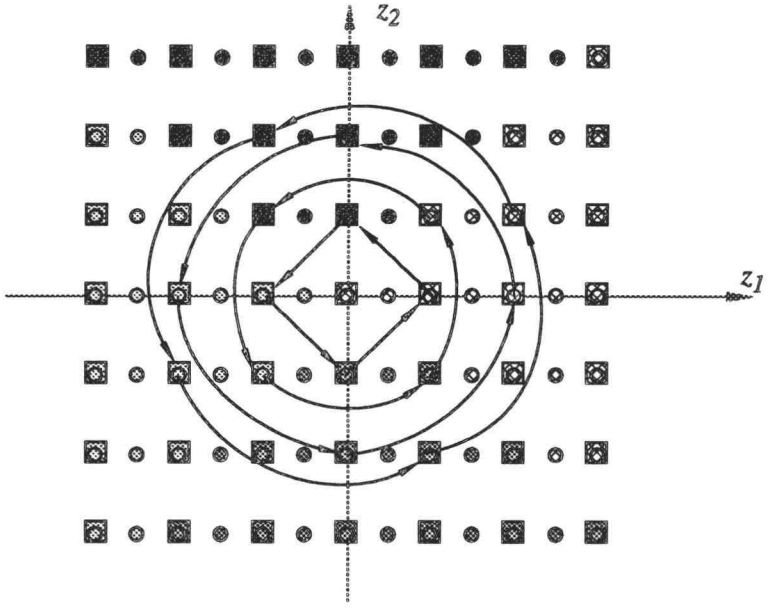
   4.4 $\forall D_i \in G_l$, form the matrix $A_i = A_0 D_i^{-1}$.

4.5 For $0 \le i < |G_l|$, define $F_i = \{x | A_i(x \le 0)\}$.

4.6 Construct $q \in F_0^i$ such that for $0 \le i < |G_l|$, $1 \le j < |G_l|$, $[A_i]_j q \ne 0$.

4.7 Construct the linear constraints for $P_i$ from those of $F_i$ as follows. If $[A_i]_j q < 0$, then use $[A_i]_j x \le 0$; else use $[A_i]_j x < 0$. These polytopes are the $P_i$.

**Example[3.13]:** Let $W(p) = f[U(D_1 p), U(D_2 p), \ldots]$, where $p = [i\ j\ k]^T$ and $D_1 = \begin{bmatrix} 2 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}, D_2 = \begin{bmatrix} 0 & 2 & -1 \\ -1 & 0 & 1 \end{bmatrix}$. Let the variable array $U$ be embedded in the $(z_1, z_2)$ subspace, that is, let $\beta = \{1, 2\}$, and so $D_1(\beta) = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$ be used to construct the coordinate system for variable $U$. The dependencies at $z_3 = 0$ are given as in Figure 3.13.




□ : *Index points for W*
o : *Index points for U*
◘ : *Node of W depends on node of U through the first dependence D1.*

**Figure 3.17:** The index points for variable W at $z_3 = 0$. The two dimensional array $U$ is embedded into this plane at $z_3 = 0$ according to its own coordinate system $C^2_{U,1}$. The DAG has non-uniform data dependencies.

In Figure 3.13, the elements of $W$ are indexed under the Standard coordinate system and the elements of $U$ are index under the coordinate system $C^{(2)}_{U,1}$ which is defined as:

$$C^{(2)}_{U,1} = \{ ([1\ 0\ 0], 1/2), ([0\ 1\ 0], 1) \} \tag{3.40}$$

The first dependence relation defined by $D_1$ is local and illustrated by the connected squares and circles. However, the dependence relation defined by the second index mapping function $D_2$ is not uniform and they are the function of the index points. To transform these non-uniform dependencies into uniform ones, the use of multiple coordinate systems for the variable array $U$ is required. Thus the computational domain is to be partitioned into different regions.

Let us select the first two columns for constructing the submatrices, that is, let $\beta = \{1, 2\}$, then $D_1(\beta) = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$ and $D_2(\beta) = \begin{bmatrix} 0 & 2 \\ -1 & 0 \end{bmatrix}$ for constructing the MCS system. Thus we have $(D_1(\beta))^{-1} = \begin{bmatrix} 1/2 & 0 \\ 0 & 1 \end{bmatrix}$ and $((D_1(\beta))^{-1} D_2(\beta))^4 = I$ which means that we need to partition the domain into four regions and therefore four coordinate systems are needed for defining the index space of the variable array $U$. The computational domain $\Omega$ is partitioned into the following four regions defined as in Eq.(3.36) with their coordinate systems Eq.(3.37):

$$\begin{aligned} P_1 &= \{p \mid i \geq 0, i \geq j, i > -j\} \\ P_2 &= \{p \mid j < 0, i \leq -j, i > j\} \\ P_3 &= \{p \mid i < 0, j \geq i, j < -i\} \\ P_4 &= \{p \mid j > 0, j \geq -i, j > i\} \end{aligned} \tag{3.41}$$

$$
\begin{aligned}
C^2_{U,1} &= \{[1/2\ 0\ 0], [0\ 1\ 0]\} \\
C^2_{U,2} &= \{[0\ 1/2\ 0], [-1\ 0\ 0]\} \\
C^2_{U,3} &= \{[-1/2\ 0\ 0], [0\ -1\ 0]\} \\
C^2_{U,4} &= \{[0\ -1/2\ 0], [1\ 0\ 0]\}
\end{aligned}
\tag{3.42}
$$

These partitions of the computational domain are illustrated in the Figure 3.18.

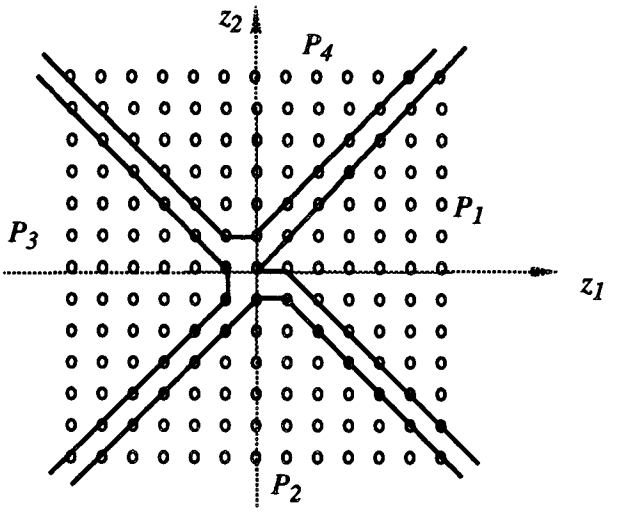
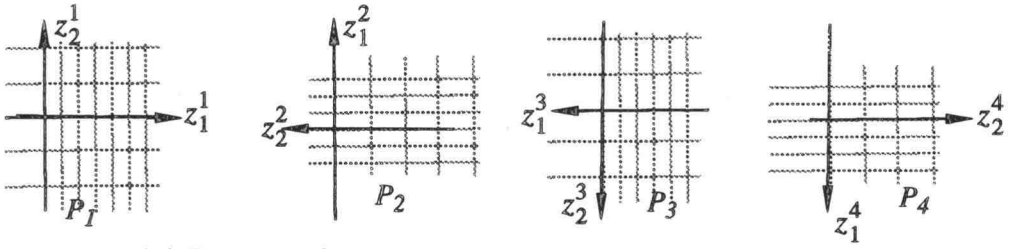


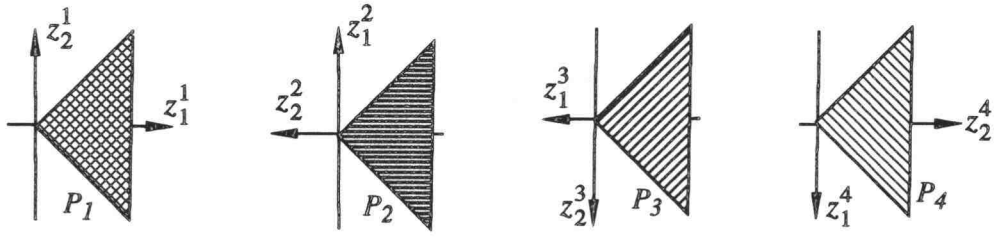

**Figure 3.18:** One possible partitioning of the computational domain.

---

These coordinate systems defined as in Eq.(3.37) are illustrated in Figure 3.19*a* with their associated indexing regions are presented in Figure 3.19*b*.




*(a) Four coordinate systems each a partition $P_i$*




*(b) Each region is indexed under its own coordinate system*

**Figure 3.19:** The coordinate systems for variable U and their index regions as expressed in them.

---

Corresponding to this partitioning, the computational domain for $W$ is also partitioned into four regions. These regions are similarly defined. Figure 3.20 illustrates the dependence structure after embedding the $U$ and $W$ variable arrays into the computational domain according to the MCS system.

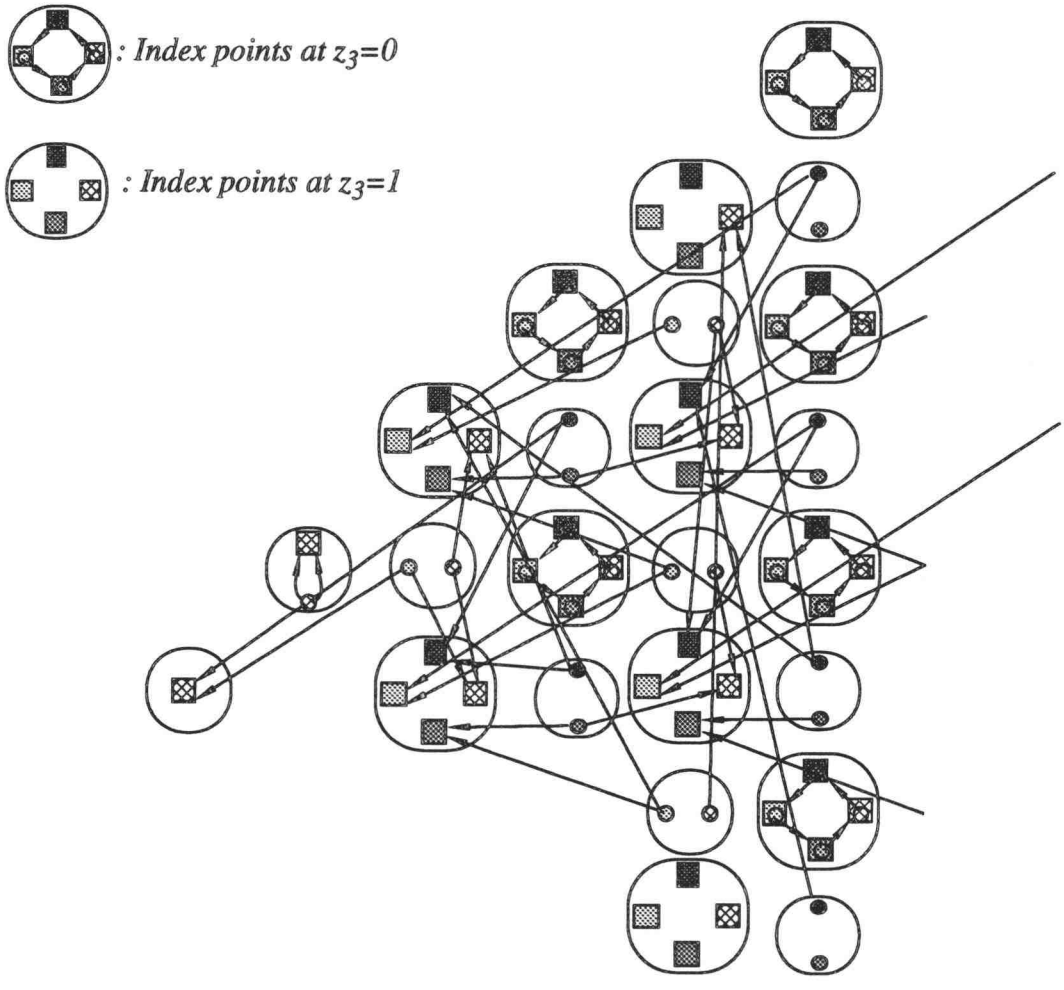



**Figure 3.20:** The dependencies at $z_3 = 0$ and $z_3 = 1$. All dependencies at $z_3 = 0$ are uniform (except those inside a node), while the dependencies at $z_3 = 1$ are non-uniform on the boundary of the regions but are uniform inside.

## 3.6 CONVERTING ADIOS TO REGULAR SPECIFICATIONS

For a given algorithm, the ultimate objective is to map it onto a regular VLSI array architecture. This can be achieved if and only if the algorithm can be presented as specifications with regular data dependence structures. This section is concerned with how to

transform an ADIO to its corresponding specifications with regular dependence structures. The structurally well defined specification in Eq.(3.12) provides all informations regarding how the index space of each variable is defined in relation to its own coordinate systems. However, in general, the dependencies in Eq.(3.12) are not uniform. Global broadcasting dependencies need to be localized to transform it into a specification with regular dependence structures.

Single assignment property is necessary for pipeline implementations. Only computable algorithms are concerned here.

- *Single Assignment Property*: For each output variable $U_o$ in a system of ADIOs, after its index space has been completely expanded from $m$- to $n$-dimensions, any instance of $U_o$, $U_o(p), p \in Z^n$, appears at most once on the left hand side of a recurrence equation.
- *Computability*: After the index spaces of all variables in an algorithm have been completely expanded, there exists a partial ordering of the equations such that any variable instance appearing on the right hand side of an equation appears earlier on the left hand side of some equation in the partial ordering, except those variables input from outside of the computational domain of the algorithm.
- *Thorough Distribution Property*: A data distribution mechanism is said to possess thorough distribution property if it can distribute all elements $U_j(q)$ of a variable array $U_j$ to their corresponding node sets $N_j(q)$, and it is said that $U_j$ will be thoroughly distributed.

Transforming ADIOs to their corresponding specifications with regular dependence structures consists of the following steps:

1) Decide the convertibility of a given ADIO.

2) Construct a system of multi-coordinate systems for the ADIO.

3) Find the embedding functions for each variable in the ADIO

4) Determine the iteration space for each variable.

5) Perform *index space expansion* for each variable.

Steps 1, 2 and 3 have already been studied in previous sections. This section will determine the iteration space and how to expand the index space of each variable in a given ADIO to localize the global dependencies.

### 3.6.1 The iteration space

In order to perform pipelining implementation for a given algorithm, single assignment property is necessary for the algorithm specification. however, in ADIO specifications, multiple assignment code may exist. Therefore, a procedure must be performed to transform an ADIO with multiple assignment property into a corresponding specification with single assignment property. This requires the operation called *index space expansion.* The index space expansion transforms a lower dimensional $m$-array into a corresponding higher dimensional $n$-array by expanding the index space of the $m$-array from $m$- to $n$-dimensions. However, before the index space expansion can be performed, it is necessary to determine subspace along which the expansion can be performed, which is called the *iteration space*, and the iteration orientation such as along the $+1$ or $-1$ direction along some the iteration space.

The iteration space is the $(n-m)$ -subspace of $\Omega$ along which the $m$-array $U_j(q)$ $(q \in Z^m)$ is expanded into an $n$-array $U_j(p)$ $(p \in Z^n)$. Formally, the iteration space is defined as:

**Definition 3.6:** *Iteration Space IS*: The iteration space of a variable $U_j(q)$ $(q \in Z^m)$ is the $(n-m)$ -subspace of the computational domain $\Omega \in Z^n$ along which the index space of $U_j$ is expanded to transform $U_j$ into its corresponding iterative format with single assignment property specified as:

$$U_j(p) = g(U_j(p-d), \ldots), p \in \Omega$$

where $g$ is a mapping function, $d$ is the dependence vector along the iteration space. Furthermore, $d$'s must fill the entire iteration space such that $U_j(q)$ is thoroughly distributed.

**Lemma 3.4:** The iteration space $IS_j$ of variable $U_j(Dp+d)$ in an ADIO belongs to the *null space* of $D$, i.e., $IS_j \subseteq \aleph(D)$.

**Proof.** Assume that $IS_i$ does not belong to $\aleph(D)$, that is, let $\vec{\vartheta} \in IS_i \wedge \vec{\vartheta} \notin \aleph(D)$. If $\vec{\vartheta}$ is used as the iteration vector, then every node $U_i(p)$ is propagated to $U_i(p+\vec{\vartheta})$ (here $U_i$ has already been expanded so that $p \in \Omega$ rather than $\Omega_i$). Let $p_2 = p_1 + \vec{\vartheta}$, since $\vec{\vartheta} \notin \aleph(D)$, we have $D\vec{\vartheta} \neq 0$. Therefore, $Dp_2 \neq Dp_1$, this means that $U(Dp_2)$ and $U(Dp_1)$ are different data elements in the $U_i$ array. But, $\forall p \in \Omega \wedge p = p_2 + a\vec{\vartheta}, 0 < a \in Z$, then each node $p$ will be assigned twice by $U(Dp_2)$ and $U(Dp_1)$ respectively, which violate the single assignment property. Therefore, the iteration space of $U$ must belong to $\aleph(D)$. ■

**Lemma 3.5:** The iteration space $IS_j$ of variable $U_j(Dp+d)$ in an ADIO possesses thorough distribution property only if it contains the *null space* of $D$, i.e., $\aleph(D) \subseteq IS_j$.

**Proof.** Assume that $IS_j$ does not contain $\aleph(D)$, that is, $p_1 \notin IS_j \wedge p_1 \in \aleph(D)$. Let $W(p) \leftarrow U_j(q) = U_j(Dp)$ and $D(p-p_1) = 0$. Then we have also the dependence relation $W(p_1) \leftarrow U_j(q)$. But $p_1 \notin IS_j$, therefore, the element $U_j(q)$ would not be sent to $p_1$. This means that the iteration space $IS_j$ does not possesses the thorough distribution property. Therefore, $\aleph(D) \subseteq IS_j$ must be respected. ■

From Lemmas 3.3 and 3.4, we have the following theorem for the determination of the iteration space for a variable in an ADIO.

**Theorem 3.4:** Given an ADIO, any variable array $U(Dp+d), D \in Z^{m \times n}, m \leq n$, in it can be expanded to an ARE which satisfies the single assignment property and thorough distribution property if and only if the space of expansion is the *null space of D.*

**Proof.** Lemma 3.3 provides the necessary condition which guarantees that the ARE has single assignment property. Lemma 3.4 ensures that each element of $U_i$, $U_i(q)$, is propagated to all the elements of $N_i(q)$. ■

By expanding the output variable solely, an ADIO is transformed into a system of ARE. For example, let an ADIO be $y(i) = \sum_j h(j)\, x(3i-j)$, the iteration space for variable $y$ is defined as $IS_y = [0\ 1]^T$. Let the coordinate system for variable $y$ be defined as $\mathbf{u}_y = [1\ 0]^T$ and let the iteration orientation be $[1\ 1]$. Then the ADIO is transformed into a system of ARE defined as:

$$
\begin{aligned}
y(i,j) &= y(i,j-1) + h(j)\, x(3i-j) \\
y(i,-1) &= 0 \\
y(i) &= y(i,N-1)
\end{aligned}
$$

The index space of y is expanded from 1-dimensional to 2-dimensional which transforms variable $y$ into a two dimensional array from a 1-dimensional array.

### 3.6.2 Expanding an *m*-array to an *n*-array

Index space expansion generates a higher dimensional array from a lower dimensional one, which is necessary for transforming a given ADIO to its ARE or regular specifications.

**Definition 3.7:** *Index Space Expansion:* Let $U_j$ be an $m$-dimensional variable array in a

given $n$-dimensional ADIO with domain $\Omega$ $(m < n)$, and the index space of $U_j$ be $\Omega_j \subseteq \Omega$. The index space expansion generates a new set of index vectors $p \in \Omega$ for $U_j$ from the old index vectors $q \in \Omega_j$ such that $U_j$ is expanded from an $m$-dimensional array to an $n$-dimensional array.

What is the format and characteristics of a regular specification under a MCS system? The characteristics of a regular specification under an MCS system is that the specification has uniform dependencies over the entire computational domain $\Omega$ characterized by a dependence vector or a set of dependence vectors $d_j \in \aleph(D_j)$ for variable $U_j$, except that the norms of $d_j$ may be less than others in a subdomain $\Omega_j^{\varepsilon}$ called the vicinity domain of $\Omega_j$, where $\Omega_j \subseteq \Omega_j^{\varepsilon} \subseteq \Omega$, while the orientation of all $d_j$ should be the same over the entire domain $\Omega$. Formally, the regular specification is defined as:

**Definition 3.8:** *Regular Specifications Multi-Coordinate-Systems* (RMCS): A recurrence equation is said to be an RMCS if it has uniform dependencies throughout its entire computational domain under an MCS system as specified as:

$$U_i(p) = f[U_1(p), \ldots, U_j(p + d_{ji}), \ldots, U_V(p + d_{Vi})], \forall p \in \Omega_i$$

$$U_j(p) = \begin{cases} U_j(p + d_j), \forall p \notin \Omega_j^{\varepsilon} \\ U_j(D_j p + d'_j)_{\mathbf{u}_j}, \forall p \in \Omega_j^{\varepsilon} \end{cases} \tag{3.43}$$

where $\Omega_j^{\varepsilon}$ denotes the vicinity domain of the computational domain $\Omega_j$. $\forall p \notin \Omega_j^{\varepsilon}$ means that index point $p$ does not depend on any point $q \in \Omega_j$ directly, or those points $q$ which are on the other side of the domain crossing the domain $\Omega_j$, such that the dependence vector will not intersect with the domain $\Omega_j$. $U_j(D_j p + d'_j)_{\mathbf{u}_j}$ indicates that the index space of variable $U_j$ is defined in relative to the basis $\mathbf{u}_j$, or the coordinate system $C_j^{m_j}$. ■

The vicinity domain is defined as follows. Let $W(p) \leftarrow U_j(D_j p), D_j \in Z^m$, and let the m-array $U_j(q'), q' \in Z^m$ be embedded into the m-subspace of $R^n$ such that the other $(n-m)$ coordinates are all zero, that is, $z_k = 0, \forall k \in \{1, 2, \ldots, n\} \wedge k \notin \beta$. Moreover, without loss of generality, let the computational domain $\Omega$ be defined in such a way that $0 \leq z_j, \forall j \in \{1, 2, \ldots, n\}$. Such a assumption can be justified because, for all computable algorithms, their computational domains must be bounded or semi-infinite only along certain coordinate and bounded along others.

Applying index space expansion to the $m$-array $U_j(q'), q' \in Z^m$ will expand it into an $n$-array $U_j(q), q \in Z^n$ by propagation along the right null space of $D_j$. Then we have all the dependencies characterized by $d_j \in \aleph(D_j)$ and $U_j(q) = U_j(q - d_j)$. Then the vicinity domain $\Omega_j^\varepsilon$ where $\Omega_j \subseteq \Omega_j^\varepsilon \subseteq \Omega$ is defined as:

$$\Omega_j^\varepsilon = \{p \mid W(p) \leftarrow U_j(q), \mathbf{e}_k^T q \leq 0;\ p, q \in Z^n, \forall k \in \{1, 2, \ldots, n\} \wedge k \notin \beta\}$$

The vicinity domain $\Omega_j^\varepsilon$ is illustrated by the diagram shown in Figure 3.17. Note that the domain $\Omega_j$ may be just a subdomain of the entire computational domain and it is defined in relative to its own coordinate system $C_j^{m_j}$. The dependencies in an RMCS are uniform in the sense that there is only a limited number of dependence vectors in the entire domain $\Omega_i$ and this number is not a function of the domain size parameters. Whenever $p \in \Omega$ but $p \notin \Omega_j^\varepsilon$, then the dependence vector is always equal to $d_j$. However, if $p \in \Omega_j^\varepsilon$, then the dependence vector may have different norms but its orientation should remain the same.

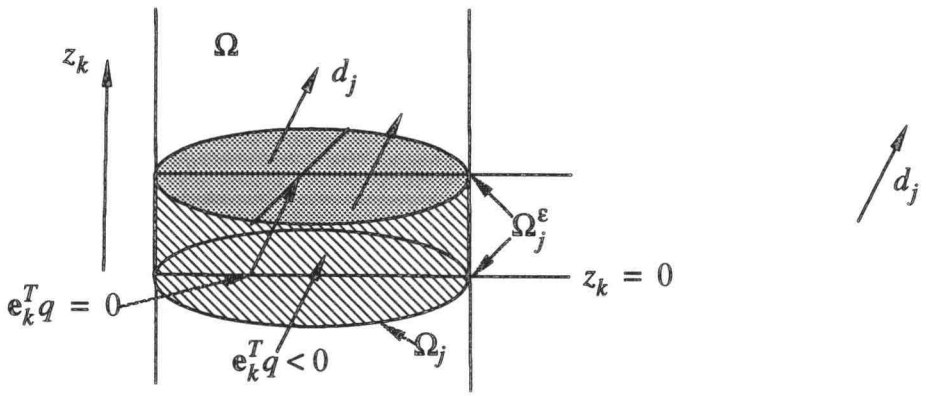



**Figure 3.21:** Illustration of the vicinity domain $\Omega_j^\varepsilon$ of $\Omega_j$, $\Omega_j \subseteq \Omega_j^\varepsilon \subseteq \Omega$.

The application of MCS technique for the analysis of algorithms' dependence structures can best be illustrated through examples. The following examples show dependence structures under single coordinate systems and under MCS systems.

**Example[3.14]:** Let us now consider again the dependence $U_o(p) \leftarrow U_1(D_1 p)$ studied in Example[3.5] to see how the vicinity domain of $\Omega_1$ is defined. Let the computational domain be $\Omega = \{p | 0 \le z_1, 0 \le z_2 \le N_1, 0 \le z_3 \le N_2\}$. Let $U_1$ be again embedded into the subspace $(z_1, z_2)$ in relative to the basis $\mathbf{u}_1^1 = [1/2\ 0\ 0]^T, \mathbf{u}_2^1 = [0\ -1\ 0]^T$, or the coordinate system $C_1^2 = \{\mathbf{u}_1^1, \mathbf{u}_2^1\}$. The global dependencies can be removed by propagating the elements of $U_1$ in the right null space of $D_1$ to all index points where it is used. The dependence vector is the right null space of $D_1$ which is given as:

$$D_1 = \begin{bmatrix} 2 & 0 & -1 \\ 0 & -1 & 1 \end{bmatrix}, d_j = \aleph(D_1) = [1\ 2\ 2]^T$$

Since $\beta = \{1, 2\} \subseteq \{1, 2, 3\}$ and $D_1(\beta)$ were selected to construct the coordinate system for $U_1$, Therefore, $\mathbf{e}_3$ is used to compute the vicinity domain of $\Omega_x$ because $3 \in \{1, 2, 3\} \wedge 3 \notin \{1, 2\}$. Accordingly,

$$\mathbf{e}_3^T d_j = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix} = 2 \Rightarrow z_3 \leq 2$$

while the other parameters are defined the same as before, thus the vicinity domain is defined as $\Omega_x^{\varepsilon} = \{ (z_1, z_2, z_3) \mid 0 \leq z_1, 0 \leq z_2 \leq N_1, 0 \leq z_3 \leq 2 \}$. The dependence graph is given in Figure 3.22 where the global broadcasting dependencies have been removed by pipelining.

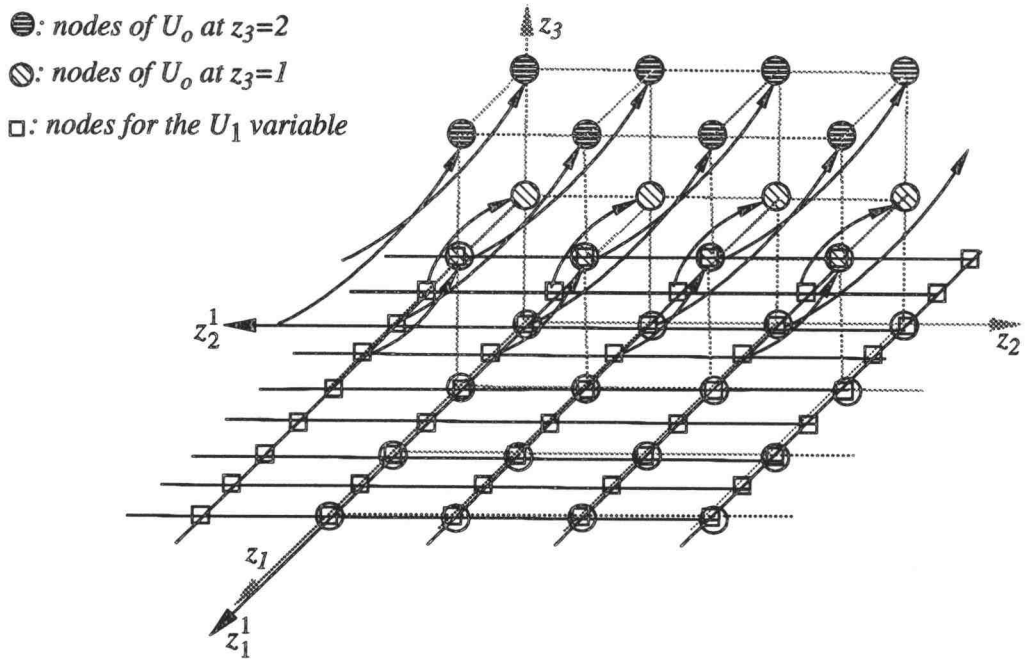



**Figure 3.22:** The vicinity domain of $\Omega_1$ is defined as $\Omega_1^{\varepsilon} = \{0 \leq z_3 \leq 2\}$. When $p \in \Omega_1^{\varepsilon}$, then $U_o(p) \leftarrow U_1(D_1 p)$ should be used.

The uniqueness of MRA architecture is that the temporal locality constraint does not exist and each data link may have its own clock frequency. This feature makes it a suitable target array structure for the implementation of RMCS. When the different scale sizes in MCS system are used properly, they can be transformed into the definitions of different clock rates in MRA architectures. The following example briefly shows the synthesis procedure based on the MCS technique.

**Example[3.15]:** The ADIO defined in Eq. (3.9) for the decimation filter is studied here to show the synthesis procedure. To simplify the analysis of this algorithm, let $M = 3$ and $N = 10$, thus the ADIO is specified as:

$$y(i) = \sum_{j=0}^{9} h(j)\, x(3i-j), 0 \le i \tag{3.44}$$

where $h(j)$'s are the filter's coefficients and $x(i)$'s are the filter's inputs. The computational domain of this ADIO is defined as $\Omega = \{0 \le i, 0 \le j \le 9\}$. Let us use the MCS technique discussed in previous sections to synthesize this algorithm.

*Step 1.* This algorithm is apparently convertible to regular specifications, since each variable has only associated with one linear part.

*Step 2.* Construct an MCS system for the ADIO. This algorithm has a 2-dimensional computational domain, which is to be defined in accordance to a 2-dimensional standard coordinate system $C_s^2$. The index mapping matrices are specified as:

$$D_y = [1\ 0], D_h = [0\ 1], D_x = [3\ -1] \tag{3.45}$$

For the variables $y$ and $h$, each of them has only one submatrix which is non-singular, thus their corresponding $\beta$ must be selected as $\beta_y = \{1\}$ and $\beta_h = \{2\}$. Therefore,

variables $y$ and $h$ each may have one set of coordinate system only where the element of the set is coordinate systems with different origin only but the others are the same. However, $D_x$ has two non-singular submatrices $D_x(\{1\}) = [3]$ and $D_x(\{2\}) = [-1]$, therefore, variable $x$ has two sets of coordinate systems. Let the coordinate systems be defined by the following transformations:

Figure 3.23a corresponding to the selection of $C_x^1 = \{[1/3\ 0]\}$ with the same origin with $C_s^2$. Figure 3.23b shows the same coordinate system except that its origin is defined at $[N/3\ (N-1)]$ for an $N$th order filter. Figure 3.23c shows the coordinate system defined as $C_x^1 = \{[0\ -1]\}$ with the same origin with $C_s^2$. It should be point out here that the number of selections for embedding a lower dimensional array into a higher dimensional space is infinite if no restrictions are imposed. It is impossible to find all of them. Based on the fact that the coordinate systems are constructed from the selection of $\beta$, we thus have limited ourselves to the selection of an $m$-standard space to embed an $m$-array. But we should remember that we have the freedom to select other embedding strategies and whenever we need this kind of freedom, we may still use it to optimize designs.

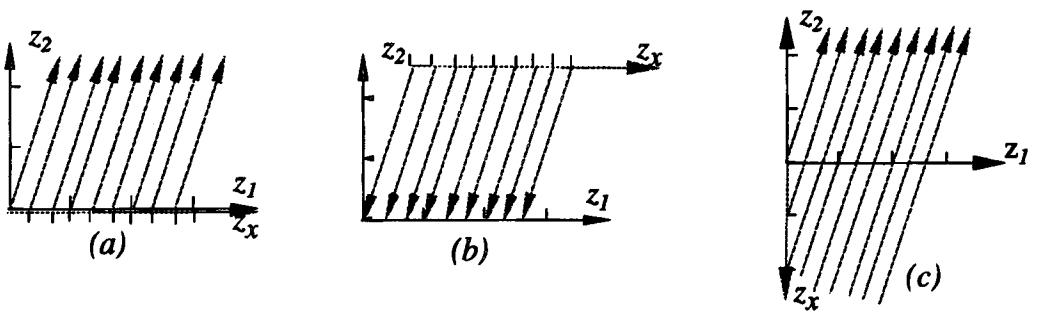



**Figure 3.23:** Several different possible coordinate systems for the indexing of the variable array $x$.

Let us now consider the implementation of this algorithm. Let the transformations be defined as $\mathbf{T}_y = [1]$, $\mathbf{T}_x = [1/3]$, $\mathbf{T}_h = [1]$. The corresponding coordinate systems are then defined as:

$$C_y^1 = \{[1\ 0]\}, C_x^1 = \{[1/3\ 0]\}, C_h^1 = \{[0\ 1]\}$$

These coordinate systems are as shown in Figure 3.24, where the axis of each coordinate system are labelled as $z_y, z_x, z_h$ respectively. The gray lines shows the standard coordinate system $C_s^2$. Firstly, let the origins of $C_x^1, C_h^1$ be located at the same location as the standard coordinate system $C_s^2$, and let the origin of $C_y^1$ be located at the location $(z_1, z_2) = (0, N-1)$ for an $N$th order filter. This yields in the MCS system as given in Figure 3.24. Such selection will result in one MRA implementation.

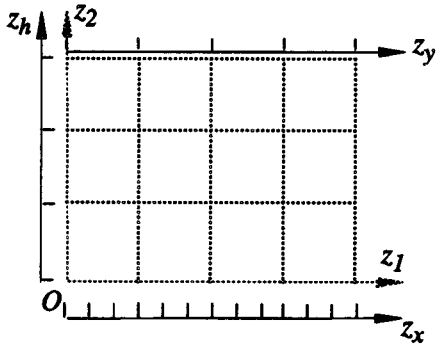



**Figure 3.24:** The MCS system for in $R^2$ consists of $C_s^2$ and three other coordinate systems $C_y^1, C_x^1, C_h^1$.

---

*Step 3.* Once the MCS system has been constructed for the algorithm, the dependence structure of the ADIO is also defined and Eq. (3.39) can be represented as the following structured equation:

$$y(i)_{u_y} = \sum_{j=0}^{9} h(j)_{u_h} x(3i-j)_{u_x}, 0 \le i \tag{3.46}$$

All variable arrays can now be embedded into the computational domain $\Omega$ according to the following embedding functions:

$$\begin{aligned} \sigma_y &: Z \to Q^2, \sigma_y(i) = (z_1, N-1) \\ \sigma_h &: Z \to Q^2, \sigma_h(i) = (0, z_2) \\ \sigma_x &: Z \to Q^2, \sigma_x(i) = (\frac{z_1}{3}, 0) \end{aligned} \tag{3.47}$$

After such embeddings, the dependence structure is as shown in Figure 3.25 where the dependencies are not yet transformed into regular specifications. Broadcasting dependencies need to be localized. Note that the dependence vectors associated with those input variables ($h$ and $x$) are directed into the index points $p$ while those associated with the output variable $y$ are directed toward the nodes of $y$. This shows how information flows.

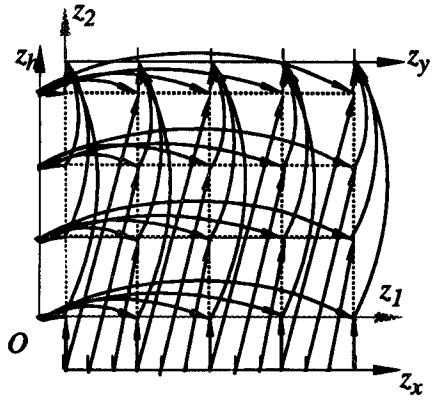



**Figure 3.25:** The dependence structure under an MCS system for in $R^2$ consists of $C_s^2$ and three other coordinate systems $C_y^1$, $C_x^1$, $C_h^1$.

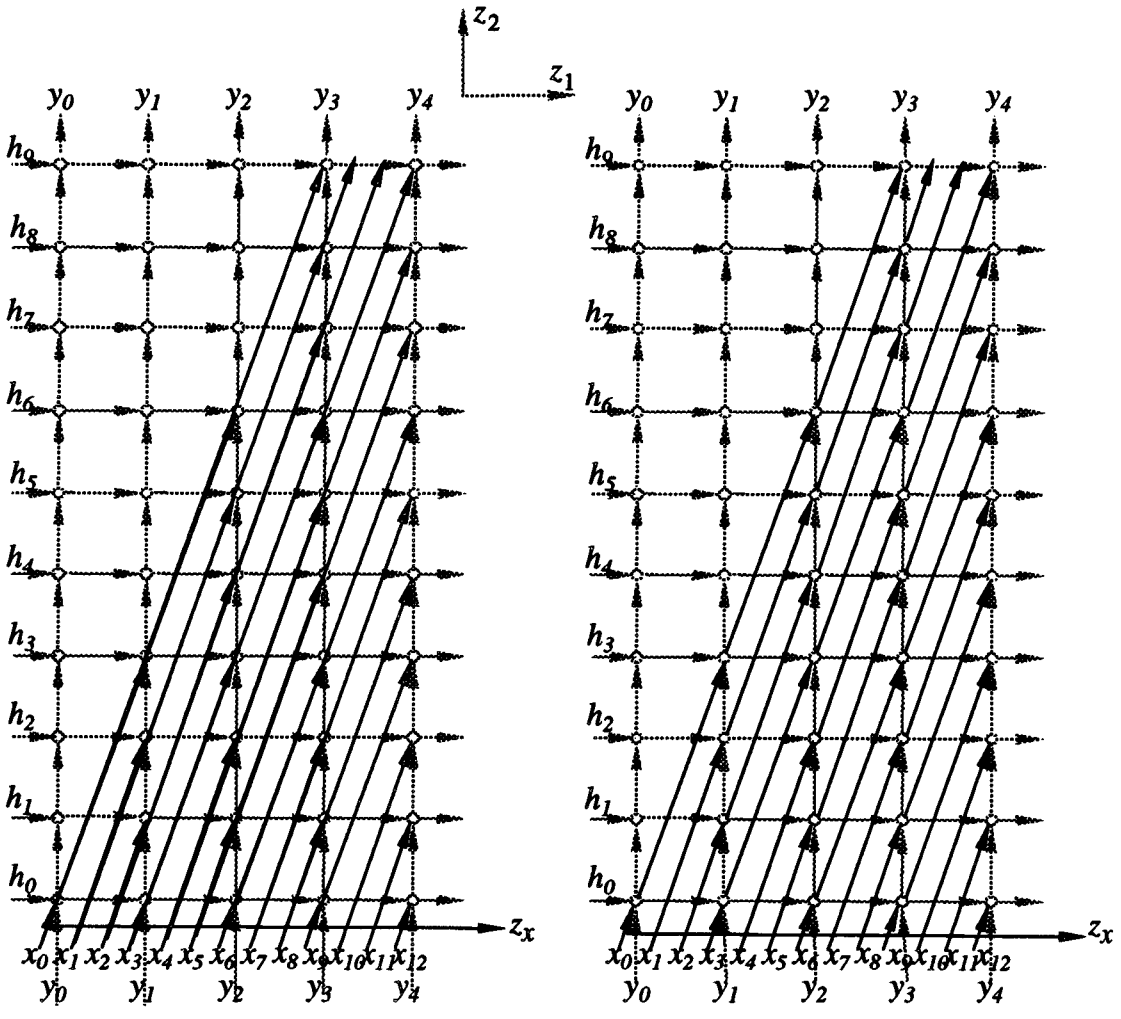
Step 4. *Index space expansion and dependence localization*: The iterative property of Eq. (3.41) is not explicitly expressed in the equation and the dependence structure is still not uniform. The index space expansion aims at extracting the iterative property and the regular property of an equation. Let us first perform the index space expansion for variables $y$ and $h$ (Note, it is possible to expand the index spaces of all variables at the same time). These two arrays are expanded from 1-dimensional arrays where the variables are defined only in their 1-dimensional index spaces to 2-dimensional arrays where these variables are defined over the entire domain. After this expansion, Eq. (3.41) can be represented as a system of ARE as follows:

$$\begin{cases} For \quad 0 \le i, 0 \le j \le 9 \\ \quad y(i,j) = y(i,j-1) + h(i,j)\, x(3i-j)_{\mathbf{u}_x} \\ \quad h(i,j) = h(i-1,j) \\ \quad h(0,j) = h(j)_{\mathbf{u}_h} \\ \quad y(i,-1) = 0 \end{cases} \tag{3.48}$$

Please refer to chapter 2 for the dependence structure of Eq.(3.43) under a single coordinate system. We can further localize those global dependencies of $x$-variable by pipelining, and Eq.(3.43) can be expressed as a system of regular structures specifications as follows:

$$
\begin{cases}
For\,(0 \le i, 2 < j \le 9) \\
\quad y(i,j) = y(i,j-1) + h(i,j)\,x(i-1,j-3) \\
\quad h(i,j) = h(i-1,j) \\
\quad x(i,j) = x(i-1,j-3) \\
For\,(0 \le i, 0 \le j \le 2) \\
\quad x(i,j) = x(3i-j)_{u_x} \\
\quad y(i,-1) = 0 \\
\quad h(-1,j) = h(j)_{u_h}
\end{cases}
\tag{3.49}
$$

Under the MCS system defined above, Eq.(3.43) has a data dependence structure as given in Figure 3.26a with broadcasting global dependencies. The corresponding DAG for Eq.(3.44) is given in Figure 3.26b, whose dependencies are localized by pipelining. From this uniform DAG, it is straight forward to implement it onto regular array architectures.




*(a) DAG with global dependencies*

*(b) DAG with uniform dependencies*

**Figure 3.26:** The DAG under MCS system for the decimation filter algorithm studied in Example[3.12]. The dependencies of $h$ and $y$ variables have been localized by index space expansion.

---

Step 5. Mapping the uniform DAG in Figure 3.26b to regular array architecture. Since it is regularly structured, so it is easy to derive the MRA architecture from Figure 3.26b. Let the allocation function be defined as $a(p) = [0\ 1]p = z_2$, and the timing functions be

given as $t_1(p) = [1\ 1]p = z_1 + z_2$, $p \in Z^2$, for the variables $y$ and $h$ (where $z_1$ and $z_2$ are the coordinates in relative to the standard basis), and $t_2(q) = [3\ 3]q = 3(z_1 + z_2)$, $q \in Q^2 = \{\frac{p}{3} \mid p \in Z^2\}$ for variable $x$ respectively, then resultant MRA architecture is the same as the one given in Figure 2.6.

As for comparison, we provide an single rate array architecture derived from the DAG in Figure 3.26b for this algorithm. Since the DAG is uniform, it is possible to implement this algorithm onto SRA structures. One possible implementation is as shown in Figure 3.27 by projecting the DAG along the $i$-axis onto the $j$-axis, where 3 links for the propagation of $x$-variables are required. Each black strip in each link represents a delay while each processing element contributes another delay to each channel. This SRA array is not very suitable for VLSI implementations.

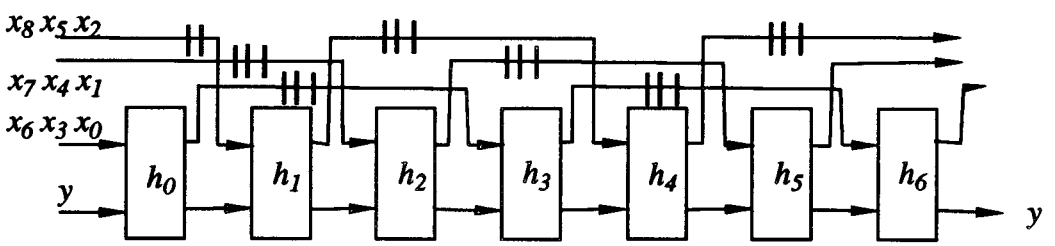



**Figure 3.27:** Another single rate VLSI array architecture for the decimation filter derived from the DAG in Figure 3.26b. Even though it has constant interconnections, the length increases with the decimation factor $M$.

---

MRA solution 2. Now let us consider another embedding strategy for this algorithm. Let the coordinate systems be defined the same way as before except that the origins are relocated. For practical applications, it is always desirable to input data elements at the

boundary processors of the array. Therefore, the coordinate systems for variable $x$ is chosen to be at the boundary of the computation domain as show in Figure 3.23a and b. Elements of $x$ are propagated along its null space into the domain. Let be origins of $C_h^1$, $C_y^1$ be located at the same origin as the standard basis. Let the origin of $C_x^1$ be located at $(\frac{N-1}{3}, N-1)$. The resultant MCS system is given in Figure 3.28.

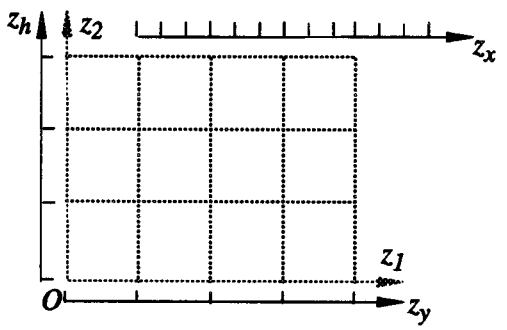



**Figure 3.28:** Another MCS system for in $R^2$ consists of $C_s^2$ and three other coordinate systems $C_y^1$, $C_x^1$, $C_h^1$ with different origins.

---

Now, for $N = 10$, the embedding functions are defined as follows:

$$\begin{aligned} &\sigma_y : Z \to Q^2, \sigma_y(i) = (z_1, 0) \\ &\sigma_h : Z \to Q^2, \sigma_h(i) = (0, z_2) \\ &\sigma_x : Z \to Q^2, \sigma_x(i) = (\frac{z_1 + 9}{3}, 9) \end{aligned} \tag{3.50}$$

The dependence structure is illustrated in Figure 3.29, where only a part of the dependence structure in the domain $\Omega$ is given.

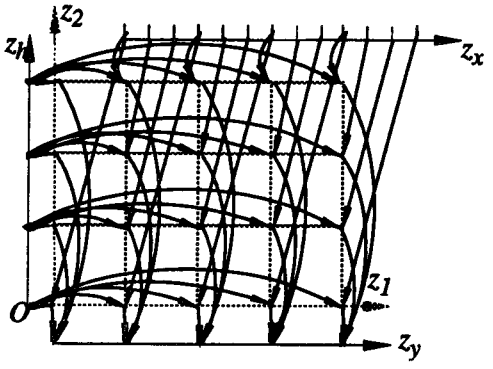



**Figure 3.29:** The dependence structure another MCS system for in $R^2$ consists of $C_s^2$ and $C_y^1$, $C_x^1$, $C_h^1$ with different origins.

---

By employing null space propagation to pipeline the data transmissions, the algorithms can be represented as the following specification with regular dependence structure. Figure 3.30 shows the DAG with localized dependencies. This new RMCS is presented as follows:

$$\begin{cases} For\,(0 \le i, 2 < j \le 9) \\ \quad y(i,j) = y(i,j+1) + h(i,j)\,x(i,j) \\ \quad h(i,j) = h(i-1,j) \\ \quad x(i,j) = x(i+1,j+3) \\ For\,(0 \le i, 0 \le j \le 2) \\ \quad x(i,j) = x(3i-j)_{u_x} \\ \quad h(0,j) = h(j)_{u_h} \\ \quad y(i,N) = 0 \end{cases} \tag{3.51}$$

The DAG corresponding to Eq.(3.51) is given in Figure 3.30.

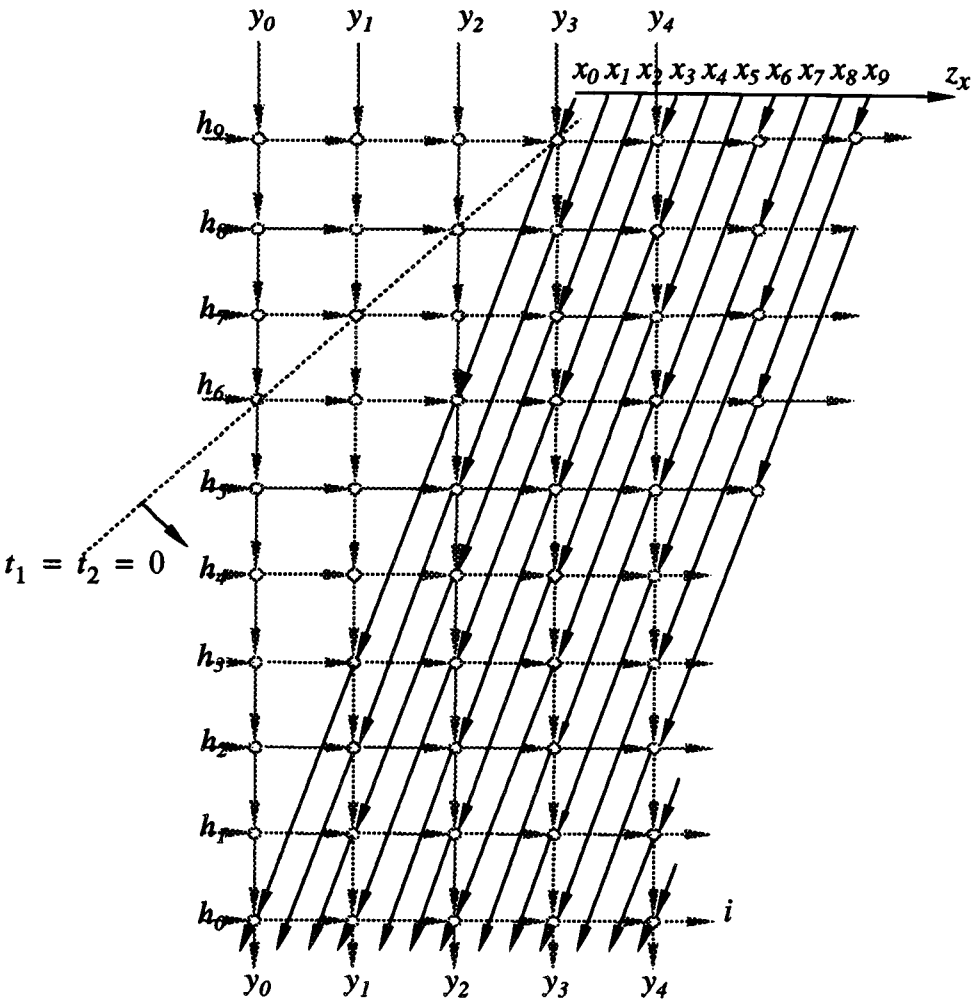
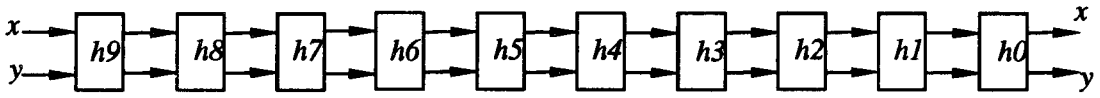



**Figure 3.30:** Another localized DAG for Eq.(3.39). From this DAG, a more efficient MRA architecture can be derived.

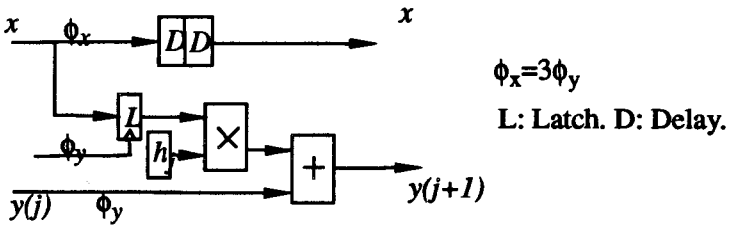
---

Let the schedule for variable $y$ and $h$ be given as $s(p) = z_1 - z_2 + c$. Then, from the relationship given in the definition, that is $s_x = r_x s$, we can easily derive the multi-rate schedule for variable $x$ to be that $s_x(q) = 3z_1 - 3z_2 + c_x$. where $c = 6$ and $c_x = 18$.

With the allocation function $a(p) = z_2$ and the timing functions as defined above, Figure 3.31 shows the new MRA array. The total computation time of this new array is less than the one shown in Figure 2.6. Moreover, this new MRA array uses less registers (two) than the previous one. So, this new array is a better design than the first one both in computational time and area. ■




*(a) Multi-rate array for the decimation filter*




*(b) Internal block diagram of the processing element*

**Figure 3.31:** Another MRA architecture for the decimation filter. This array has less delay elements in each PE, and its total computation time is less that the one given in Figure 2.6.

### 3.6.3 Dependence localization for other ADIOs

The conditions given in theorems 3.2 and 3.3 provide the sufficient and necessary conditions for localizing the dependencies of a given algorithm by the use of null space propagation technique solely. However, algorithms which do not satisfy these conditions do not necessarily mean that they can not be transformed into regular specifications. This

simply means that the null space propagation technique is not sufficient to localize the dependencies of such algorithms, and therefore multiple directional propagation would be required. Even for this class of algorithms, the MCS technique developed in this chapter is still very useful in helping a designer to determine the spaces to embed each variable array into the computational domain. The following example shows how this can be achieved.

**Example[3.16]:** A matrix Lyapunov equation is given by:

$$AX + XB = C$$

where $A$ is an $n \times n$ non-singular lower triangular matrix, $B$ is an $n \times n$ non-singular upper triangular matrix. The objective is to solve the $n \times n$ matrix $X$. the above equation can be represented in detail as:

$$for \quad 1 \le i,j \le n$$

$$c(i,j) = \sum_{k=1}^{i} a(i,k)\,x(k,j) + \sum_{k=1}^{j} b(k,j)\,x(i,k)$$

This equation can be easily represented as an ADIO defined as follows:

$$for \quad 1 \le i,j \le n$$

$$x(i,j) = \left( c(i,j) - \sum_{k=1}^{i-1} a(i,k)\,x(k,j) + \sum_{k=1}^{j-1} b(k,j)\,x(i,k) \right) / \left( a(i,i) + b(j,j) \right)$$

The computational domain of this ADIO is $\Omega = \{p \mid 1 \le i,j \le n, 1 \le k < i,j\}$. To localize the dependencies of this ADIO, the first step is to extract all index mapping matrices from it. They are given as:

$$D_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, D_c = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, D_{a1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, D_{x1} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

$$D_{b1} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, D_{x2} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, D_{a2} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}, D_{b2} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

where $D_x$ is the index mapping matrix for the output variable and the others are the index mapping matrices of the input variables ($D_{x1}, D_{x2}$ denote the different index mapping

matrices for the input variable $x$). Even though the dependence properties of this ADIO are not exactly the same as described by the convertibility condition, the dependence analysis method developed in this chapter still can be of great help in localize the dependencies of this algorithm. Let the domain be defined in relative to the standard basis $C_s^3$ whose axes are labelled as $z_1, z_2, z_3$ respectively. Since the dependence relations are so simple, we are not going to show the procedures for the derivation of each coordinate systems. The output variable array $x(i,j)$ must be embedded into the $(z_1, z_2)$ subspace since it is computed in this subspace. The $c(i,j)$ array should be similarly embedded as for th e variable $x$. The array $a(i,k)$ is embedded into the $(z_1, z_3)$ subspace while $b(k,j)$ is embedded into the $(z_3, z_2)$ subspace. These embeddings are shown in Figure 3.32. In such an embedding scheme, the only real conflict is the embedding of the $x$ variable. The embeddings for variables $a$ and $b$ suit well for both index mapping matrices of each variable.

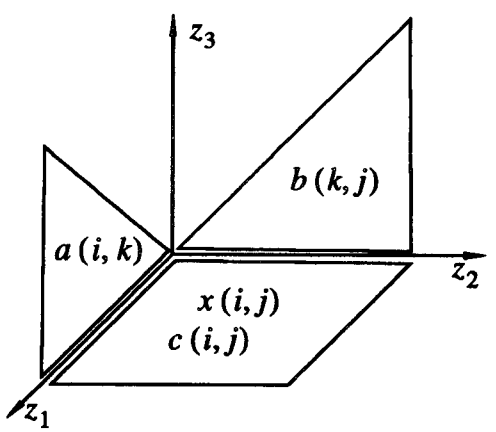



**Figure 3.32:** The subspaces for embedding the input and output arrays.

For the embedding of the $x$ variable, the above embedding scheme can not satisfy the requirements of the other two index mapping matrices $D_{x1}, D_{x2}$ which require that the variable array be embedded into the $(z_3, z_2)$ and $(z_1, z_3)$ subspaces respectively. But such requirements can not be satisfied because they conflict with the requirement of the first index mapping matrix $D_x$, which is more important since it is the output variable and so it should be considered with the highest priority. Therefore, other directional propagations might be employed to transfer the $x$ array from the $(z_1, z_2)$ subspace to the $(z_3, z_2)$ and $(z_1, z_3)$ subspaces. However, such directional propagations may not be necessary if we study the dependence structure among the elements of $x$ variable in more detail, which is given as $x(i,j) \leftarrow x(k,j), x(i,l), 1 \le k < i, 1 \le l < j$. Since $x(i,j)$ is not needed for the computations of $x(k,j), k \le i$ and $x(i,l), l \le j$, it needs to be sent to those nodes at either $z_1 > i$ or $z_2 > j$ only. Therefore, it can be first sent to the point $(i,j,k)$ and then sent to those nodes which require it along their corresponding iteration spaces. If the DAG is to be projected onto the $(z_1, z_2)$ subspace, then this other directional propagation is not needed.

After such embeddings, to localize the dependencies, the iteration spaces are identified as $IS_x = z_3$, this is the iteration space for the output variable $x$; $IS_{x1} = z_1$ and $IS_{x2} = z_2$ these are for the input variable $x$ respectively. The array $c(i,j)$ has the iteration space $IS_c = z_3$. However, since in the computation, the elements of $c$ are not used until the iterations are at the boundary of the computational domain, the elements of c do not need to be distributed over the domain. $IS_{a1} = z_2$, $IS_{b1} = z_1$ are the iteration spaces for variables $a$ and $b$ due to $D_{a1}$ and $D_{b1}$ respectively. $IS_{a2} = (z_2, z_3)$ and $IS_{b2} = (z_1, z_3)$ are for $D_{a2}$ and $D_{b2}$ (these iteration spaces are for the diagonal elements of variables $a$ and $b$ only).

Instead of propagating the input variable $x$ from the nodes $(i,j,0)$ to the node $(i,j,k)$ for performing the computations $a(i,k)x(k,j)$ and $b(k,j)x(i,k)$, we can propagate the variables $a(i,k)$ and $b(k,j)$ from $z_3 = k$ to $z_3 = 0$ and then perform the computations. These do not change the input output relationships. However, the data dependence relations are greatly simplified. The localized algorithm is given as:

$$
\begin{aligned}
&for \quad 1 \le k \le i \le n, 1 \le l \le j \le n \\
&a(i,j,0) = a(i,j-1,0) \\
&a(i,0,k) = a(i,0,k+1) \\
&b(i,j,0) = b(i-1,j,0) \\
&b(0,j,l) = b(0,j,l+1) \\
&c(i,j,0) = c(i,j) \qquad\qquad (3.52)\\
&x(i,j,0) = \begin{cases} x(i,j,0) + a(i,j)x(i-1,j) + b(i,j,0)x(i,j-1), k < i, l < j \\ x(i,j,0) + b(i,j,0)x(i,j-1), k = i, 1 \le l < j \\ x(i,j,0) + a(i,j)x(i-1,j), 1 \le k < i, l = j \\ (c(i,j) - x(i,j)) / (a(i,j,0) + b(i,j,0)), k = i, l = j \end{cases}
\end{aligned}
$$

The localized algorithm can be implemented on array architecture by projecting the dependence graph onto the $(z_1, z_2)$ space, which results in the architecture as shown in Figure 3.33.

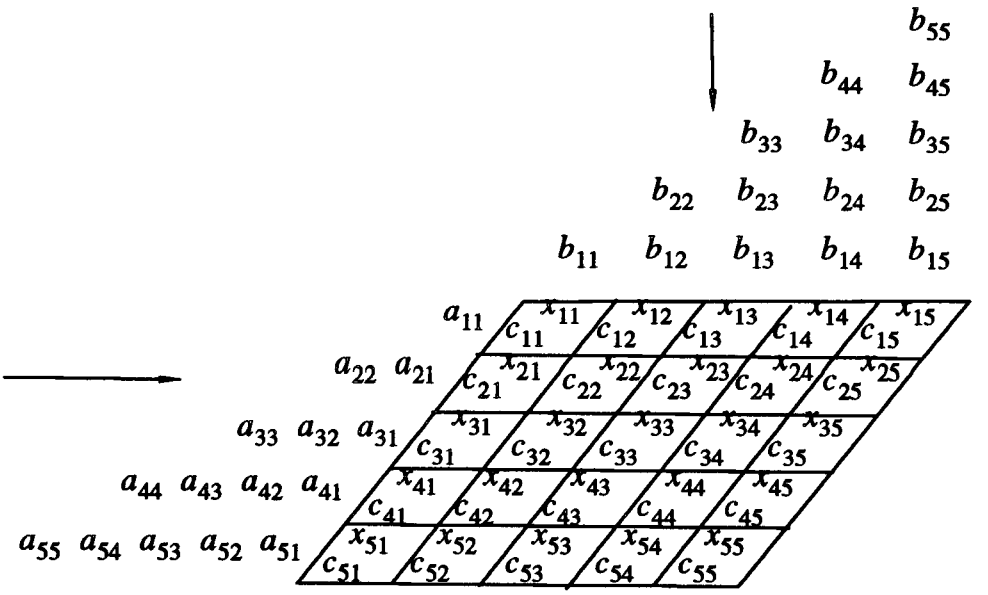



**Figure 3.33:** The array for the matrix Lyapunov equation.

## 3.7 Conclusions

In this chapter, a new initial specification ADIO was proposed which resembles most original algorithmic specifications for many applications. For example, most DSP and matrix operation algorithms are initially specified as ADIO. The roles played by coordinate systems were then studied and the limitations of single coordinate system were identified. Multi-coordinate systems was introduced and defined, together with RMCS. Then the dependence structures of ADIOs were investigated and the iteration spaces for index space expanding were also identified. A procedure was developed for the transformation of ADIOs into RMCS specifications. The introduction of MCS technique makes it possible to localize global dependencies by the use of unidirectional data propagation solely, in contrast to the multiple directional propagation technique which requires multi-

ple communication channels for a variable. This unidirectional propagation technique can thus reduce the communication cost and the complexity of control mechanism of VLSI implementations. Moreover, MCS technique allows us to synthesize algorithms whose index mapping matrices are not totally unimodular, which was not possible for conventional methodologies. Furthermore, the use of ADIO as the initial specification for synthesis allows us to derive VLSI architectures for many applications systematically and also allows us to explore the full solution space of an algorithm.

This chapter also provided a procedure for the construction of a system of MCS for a given algorithm and an algorithm to determine if a given algorithm can be transformed into a corresponding RMCS specification. A procedure was also given for the construction of an MCS system for a variable which requires more than one coordinate systems to define its index space. A procedure for computing a partitioning of the computational domain of a variable has also been provided. Examples have been provided to show the application of these procedures.

It is very desirable to reduce the number of directional propagations for each variable because each directional propagation may later require an interconnection in a VLSI implementation. The conditions given in this chapter for the determination of convertibility of regular specifications are necessary only for the use of null space propagation technique for localizing dependencies under MCS systems. However, for a given algorithm, it does not mean that a given algorithm can not be transformed into regular specifications if it does not satisfy these conditions. It does mean that the algorithm can not be localized by the use of null space propagation only if it does not satisfy these conditions, and other directional propagation may be required. However, even under such circumstances, the MCS technique developed in this chapter can still be very helpful for a designer on deciding how to embed each variable array into the computational domain.

# Chapter 4 The Synthesis of Multi-Rate VLSI Systems

Chapter 2 discussed the mapping of DAREs onto multi-rate array architectures by the multi-rate transformation. Chapter 3 introduced the multi-coordinate systems, investigated the dependence properties of ADIOs and developed procedures based on the MCS technique for data dependence localizations for ADIOs. This chapter first will investigate the scheduling issue for MRA architectures. It will introduce and define the multi-rate affine schedules and multi-rate timing functions. Sufficient and necessary conditions will be provided for the determination of multi-rate affine schedules. Then the synthesis of MRA systems based on the MCS technique will be investigated. A procedure will be developed for synthesizing MRAs from ADIOs without the phase components. Another synthesis procedure will be provided for ADIO with phase components. Examples will be studied to illustrated the applications of these procedures.

## 4.1 MULTI-RATE AFFINE SCHEDULES

Many works have been done by numerous researchers on the computation of schedules for given uniform algorithms [Qui83, Qui84, Rao85, DI86, YC88, Roy88, SF89, SF90, WD92]. However, all these techniques are concerned with the computation of single rate schedules and their optimizations. As presented in previous chapters, multi-rate arrays are advantageous over single rate arrays in many applications. The computation of multi-rate schedules is essential for designing MRA architectures. Synthesizing multi-rate arrays from regular specifications consists of two affine transformations, the multi-rate timing functions $t(p)$ and the allocation function $a(p)$. This section provides definitions of multi-rate schedules, multi-rate timing functions and their characterizations. In the design

of single rate arrays, the computational partial ordering can be determined completely by counting the number of clock cycles. Therefore, in single-rate arrays, a schedule or a timing function can be used interchangeably for specifying the computational partial ordering of a given algorithm. However, in multi-rate arrays, in order to determine this partial ordering, not only the number of clock cycles, but also the period of each clock signal must be considered. In this chapter, the term "a schedule" will be used to define a function which computes the number of clock cycles; and the term "a timing function" will be employed define the function for computing the amount of time in relation to the common clock signal, the fundamental clock rate.

In order to derive a set of affine schedules for a given ADIO, the dependence structure of the ADIO must be specified.

**Definition 4.1:** *Structurally Well Defined ADIO:* The following specification:

$$\begin{aligned} & U_i(D_i p + d_i)_{\mathbf{u}^i} = \\ & f \quad [U_i(D_{ii} p + d_{ii})_{\mathbf{u}^i}, U_j(D_{ji,1} p + d_{ji,1})_{\mathbf{u}^j}, U_j(D_{ji,2} p + d_{ji,2})_{\mathbf{u}^j}, \ldots] \end{aligned} \tag{4.1}$$

is said to be a structurally well defined specification of the ADIO. The subscripts $\mathbf{u}^i$ and $\mathbf{u}^j$ denote that the index spaces of variables $U_i$ and $U_j$ are defined relative to the coordinate systems $C_i^{m_i}$ and $C_j^{m_j}$ respectively. The origin of $C_i^m$ will be the same as $C_s^n$ if it is not defined otherwise. ■

In a regular specification, there exists a partial ordering $\propto$ for evaluating the nodes of all variables. $\propto$ is pronounced as "precedes". If $p_1, p_2 \in \Omega$, then $p_1 \propto p_2$ ($p_1$ precedes $p_2$) if and only if the evaluation of a variable at node $p_2$ uses the value of that variable at node $p_1$. In other words, $p_1 \propto p_2$ if and only if there exists a direct path from node $p_1$ to $p_2$ in the DAG. A similar definition of the partial ordering can be found in [SF89]. Computations in an ADIO are carried out only at the integer index points $p \in \Omega \subset Z^n$ of the

standard coordinate system. Operations at those fractional index points $p_q \in \Omega_q \subset Q^n$ are data transferring operations. Before introducing the multi-rate affine schedule and timing function, let us define the fundamental schedule and timing function. A fundamental schedule and a fundamental timing function for a given algorithm are affine mappings which map the index set $p \in Z^n$ into a one-dimensional integer number sequence and the real time space respectively.

**Definition 4.2:** Fundamental affine schedule and timing function: A fundamental affine schedule $s_f(p) = \pi_f p + c_f$ for a given ADIO is a mapping $s_f : Z^n \rightarrow Z$ which is strictly monotone increasing with respect to the ordering induced from the regular specification, where $\pi_f \in Q^{1 \times n}$ and $c_f \in Q$ are called the scheduling vector and the translation part respectively. The fundamental timing function $t_f(p) = s_f(p)\,\tau_f = (\pi_f p + c_f)\,\tau_f$ is a mapping function $t_f : Z^n \rightarrow R$ such that if $\exists p_1, p_2 \in \Omega \wedge (p_1 \propto p_2)$, then $t_f(p_1) < t_f(p_2)$.

For convenience, let the period of the fundamental clock signal be defined as $\tau_f = 1$. Therefore, the fundamental schedule $s_f(p)$ and the fundamental timing function $t_f(p)$ will have the same format. Under this assumption, the timing function can be considered as a mapping $t_f : Z^n \rightarrow Z$. However, the amount of time should not be confused with the number of clock cycles. The clock periods $\tau_j$ of other clock signals will be defined relative to $\tau_f = 1$.

Without loss of generality, $\pi_f$ is normalized, that is, the gcd (greatest common divider) of the elements of $\pi_f$ is 1. $t_f(p) = \pi_f p + c_f$ defines the slowest clock rate in an MRA architecture which in general involves with the most complicated operations. A multi-rate schedule $s_j(p) = \pi_j p + c_j$ assigns an integer number to each index point in its domain and a multi-rate affine timing $t_j(p) = (\pi_j p + c_j)\,\tau_j$ assigns each index point a time value for evaluating variable $U_j$ such that the computation partial ordering is preserved.

**Definition 4.3:** *Multi-rate affine schedule and multi-rate timing function:* A multi-rate affine schedule $s_j(p) = \pi_j p + c_j$ for $U_j(p)$ (it has been expanded from an $m$-dimensional variable $U_j(D_j p + d_j)$ to an $n$-dimensional array) in a regularly structured specification is a mapping $s_j : Z^n \rightarrow Z$. $s_j$ is monotone increasing with respect to the partial ordering from the corresponding regular specification. The multi-rate timing function $t_j(p) = (\pi_j p + c_j)\tau_j$ is a mapping $t_j : Z^n \rightarrow R$. For $1 \leq j \leq V$, there exists a set of integers $r_j$ so that $\pi_j = r_j \pi_f$. $r_j$ is called the multi-rate clock-rate ratio for variable $U_j$. If $\exists p, q \in \Omega$ and $q \propto p$, then $t_j(q) < t_j(p)$. The period of the clock frequency for $t_j : Q^n \rightarrow Q$ will be defined as $\tau_j = 1/r_j$.

The fundamental schedule and other multi-rate schedules for each variable form a set of schedules for a given algorithm which assigns proper time instants to all variable instants in the computational domain.

**Definition 4.4:** *Sets of multi-rate affine schedules* and *timing functions*: For a given ADIO with $V$ variables, the set of multi-rate affine schedules $\Lambda_s = \{s_f, s_1, \ldots, s_V\}$ is defined by a set of scheduling vectors $\Pi = \{\pi_f, \pi_1, \ldots, \pi_V\}$, a set of clock rate ratios $r = \{1, r_1, r_2, \ldots, r_V\}$, and a set of translation parts $c = \{c_f, c_1, \ldots, c_V\}$. If the set of scheduling vector $\Pi$ has only one component, i.e., $\pi_f$, then the schedule becomes a single rate schedule. The set of multi rate timing functions $\Lambda_t = \{t_f, t_1, \ldots, t_V\}$ is defined as $t_j(p) = s_j(p)\tau_j, \forall j \in \{f, 1, 2, \ldots, V\}$.

For example, in the decimation filter design discussed in Example[3.15], the fundamental affine schedule and timing function are defined as $s_f(p) = t_f(p) = i + j$. The multi-rate schedule for the variable $x$ is defined as $s_x(p_q) = 3i + 3j;\ \forall i, j \in \Omega_q \subseteq Q^2$, that is $\pi_f = [1\ 1]$ and $\pi_x = 3[1\ 1]$, where the clock-rate ratio is $r_x = 3$. However, the multi-rate timing function is defined as $t_x(p) = (3i + 3j)\tau_x;\ \forall i, j \in \Omega_q \subseteq Q^2$, where

the cycle period for $t_x(p_q)$ is $\tau_x = 1/3$. In other words, three $t_x$ cycles have the same period as one $t_f$ period.

**Lemma 4.1:** Given a system of regular specifications:

$$\begin{cases} U_i(p) = f_i[\ldots, U_i(q_i), U_j(q_j), \ldots] \\ \quad \ldots \end{cases}$$

$\Lambda_t = \{t_f, t_1, \ldots, t_V\}$ defines a valid set of multi-rate timing functions for the algorithm if and only if it satisfies the following conditions:

$$(1)\ \forall i, 1 \le i \le V, U_i(p) \leftarrow U_i(q_i) \Rightarrow \pi_i(p - q_i) > 0 \Rightarrow \pi_i d_i > 0 \tag{4.2}$$

$$(2)\ \forall i, j, 1 \le i, j \le V, U_i(p) \leftarrow U_j(q_j) \Rightarrow (\pi_i p + c_i)\tau_i > (\pi_j q_j + c_j)\tau_j \tag{4.3}$$

where $p, q \in \Omega \subseteq Q^n$.

**Proof.** (1) Since $U_i(p)$ depends on $U_i(q_i)$, i.e., $\forall i, 1 \le i \le V, U_i(p) \leftarrow U_i(q_i)$, therefore, the evaluation of variable $U_i(q_i)$ must be performed before the computation of $U_i(p)$. In other words, $t_i(p) > t_i(q_i)$ must be respected, which can be represented as $\pi_i p + c_i > \pi_i q_i + c_i \Rightarrow \pi_i(p - q_i) > 0 \Rightarrow \pi_i d_i > 0$. Moreover, since the dependence relation happens among the different elements of the same variable, the clock rate is the same. Therefore, the clock period does not need to be considered.

(2) This is similar to the proof of (1). However, since the dependence relation occurs between different variables and each variable may have its own clock rate, therefore the clock periods must be taken into account when considering which computation is performed first. Counting the number of clock cycles is not sufficient in multi-rate schedules. ■

Mapping an $n$-dimensional algorithm into an $(n-m)$-dimensional array architecture, the allocation function $a(p)$ can be represented by an $(n-m)\times n$ matrix $\Lambda_a$ or $m$ normalized projection vectors $\mu_1, \ldots, \mu_m$, where $\mu_i$, $1 \le i \le m$, are the right null spaces of $\Lambda_a$, that is, $\Lambda_a \mu_i = 0$, $\forall i$, $1 \le i \le m$. In other words, we restrict ourselves to consider the case of perpendicular projection only (a space is projected into a subspace normal to the projection subspace). The other condition which a multi-rate timing function must satisfy is the conflict free condition, that is, if nodes $p_1, p_2 \in \Omega$ are assigned to the same processing element, then $p_1, p_2$ should not be evaluated at the same time instant. This condition is expressed as:

$$\forall p_1, p_2 \in \Omega, a(p_1) = a(p_2) \Rightarrow t_i(p_1) \neq t_i(p_2) \text{ or}$$

$$\forall p_1, p_2 \in \Omega, t_i(p_1) = t_i(p_2) \Rightarrow a(p_1) \neq a(p_2).$$

This condition is called the conflict-free condition. This condition is necessary because each PE (or functional unit) can perform only one computation at any given time instant. If each PE contains more than one functional units which can operate independently, than the assignment of these function units and the timing function must satisfy the conflict-free condition.

## 4.2 A SUFFICIENT CONDITION FOR THE EXISTENCE OF MULTI-RATE AFFINE SCHEDULES

Assume the evaluation of $U_j(q)$ take one clock period of the clock signal $t_j$, which has a period of $\tau_j = 1/r_j$. The derivation of a feasible schedule for a variable in general depends on how its elements are embedded into the computational domain. Let the embedding function $\sigma_j : Z^{m_j} \rightarrow Q^n$ for variable $U_j$ be $\sigma_j(q) = p_j$ where $q \in \Omega_j \subseteq Z^{m_j}$ and $p \in Q^n$. The following theorem relates each individual multi-rate schedule with the fundamental schedule.

**Theorem 4.1:** Let a structurally well defined ADIO be represented as follows with the use of the intermediate variable $W(p)$:

$$U_i(D_i p + d_i)_{ui} = g[W(p)] = f[\ldots, U_j(D_{ji} p + d_{ji})_{uj}, \ldots]$$

Let $\sigma_j(q) = p_j, \forall j \in \{1, 2, \ldots, V\}$ be the embedding functions for $U_j$. Then the ADIO admits a set of multi-rate timing functions $\Lambda_t = \{t_f, t_1, \ldots, t_V\}$ if and only if:

(1) $\forall p \in N_i(q), q = D_i p + d_i \Rightarrow (\pi_i p_i + c_i)\tau_i \geq \pi_f p + c_f$

(2) $\forall j, 1 \leq j \leq V, \forall p \in N_{ji}(q), q = D_{ji} p + d_{ji} \Rightarrow (\pi_j p_j + c_j)\tau_j \leq \pi_f p + c_f$

(3). If $\gamma$ is a ray of $\Omega$, then $\pi_j \gamma > 0$;

(4). For every vertices in the domain, $\forall v \in \Omega$, $\pi_j v + c_j \geq 0$.

$U_i$ denotes the output variable; $U_j$ denotes the input variables in the ADIO; $N_i(q)$ denotes the node set where the output variable $U_i(q)$ is computed from the values at these index points; and $N_{ji}(q)$ is the node set where the computations at these index points require the value of $U_j(q)$.

**Proof.** (1) Since $\forall p \in N_i(q), U_i(q) \leftarrow W(p)$, therefore, the evaluation of $U_i(q)$ cannot be finished until all the computations at nodes $p$, $\forall p \in N_i(q)$, are finished. In other words, $\forall p \in N_i(q), t_i(p_i) > t_f(p)$ must be enforced under any circumstances. Since the clock period of $t_i(p)$ is $\tau_i$ relative to the period of $t_f(p)$ whose period $\tau_f = 1$, therefore, the result is achieved.

(2) Since $\forall p \in N_{ji}(q), W(p) \leftarrow U_j(q)$, therefore, the evaluation of $U_j(q)$ must be completed before the computations at $\forall p \in N_{ji}(q)$ can be started. In other words, the values of $U_j(q)$ must be available before starting the computations at nodes $p$, $\forall p \in N_{ji}(q)$. Therefore, the schedule for input variable $U_j(q)$ should satisfy the condition $t_f(p) > t_j(p_j)$.

(3). By the definition of a ray of a domain $\Omega$, we have:

$$\forall a > 0, a \in R, \forall p \in \Omega \Rightarrow p + a\gamma \in \Omega$$

Then there must exist a point $p_1 = p + a\gamma \in \Omega$ with a time value given as $t_j(p_1) = (\pi_j(p + a\gamma) + c_j)\tau_j$. Assume that this condition is not true, i.e., $\pi_j\gamma \leq 0$, then for a sufficiently large $a$, $t_j(p_1) = (\pi_j(p + a\gamma) + c_j)\tau_j < 0$ is expected. This is contradict to the definition of a (schedule) timing function, which cannot have negative values. On the other hand, if $\pi_j\gamma = 0$, this means that all the points lying along the ray of $\Omega$ are assigned the same timing value. Thus the number of points computed simultaneously is not bounded, which would require an infinite number of processor and is not feasible.

(4). This condition is apparent since all point belong to the domain must have a non-negative time value. ■

**Example[4.1]:** Consider the scheduling problem for the dependence relation in the 3-dimensional ADIO studied in example [3.7]:

$$U_o(i,j,k) = f[U_1(2i-k, k-j), \ldots], 0 \leq i,j,k \leq N$$

The domain is defined as $\Omega = \{p | 0 \leq i,j,k \leq N\}$ where $p = [i\ j\ k]^T$ is the index point. The index mapping matrices are:

$$D_o = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, D_1 = \begin{bmatrix} 2 & 0 & -1 \\ 0 & -1 & 1 \end{bmatrix}$$

The index mapping matrix $D_1$ is not a square matrix so it is not full rank, therefore, the scheduling technique given in [YC88] cannot be employ.

To use the multi-rate schedule for this problem, the embedding functions must be determined for each variable. As shown before, the embedding function $\sigma_o : Z^3 \rightarrow Z^3$ for $U_o(p)$ is given as $\sigma_o(i,j,k) = (z_1, z_2, z_3)$, where $i,j,k$ are not relative to any specific coordinate system, however, $z_1, z_2, z_3$ are relative to the standard basis. Again, let $\mathbf{u}_1^1 = [1/2\ 0\ 0]^T, \mathbf{u}_2^1 = [0\ -1\ 0]^T$ be the basis of the coordinate for $U_1$, i.e., $C_1^2 = \{\mathbf{u}_1^1, \mathbf{u}_2^1\}$. With the origin of $C_1^2$ been located at the same location of the origin of $C_s^3$, the embedding function for $U_1$ is defined as:

$$\sigma_1(i,j) = (\frac{z_1}{2}, -z_2, 0) \tag{4.4}$$

Thus the data dependency is defined as:

$$U_o(z_1, z_2, z_3) \leftarrow U_1(\frac{2z_1 - z_3}{2}, z_2 - z_3, 0)$$

For any point $p = [z_1, z_2, z_3]^T$, its time value $t_o(p)$ must be greater than the time value assigned to the point $(\frac{2z_1 - z_3}{2}, z_2 - z_3, 0)$ for variable $U_1$. That is

$$\pi_o \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} + c_o \geq \left( \pi_1 \begin{bmatrix} \frac{2z_1 - z_3}{2} \\ z_2 - z_3 \\ 0 \end{bmatrix} + c_1 \right) \tau_1 \tag{4.5}$$

The clock signal for $U_o$ is the fundamental clock signal, whose period is defined as $\tau_o = 1$. At point $p = [k\ k\ k]^T \in \Omega$, Eq.(4.5) becomes

$$\pi_o \begin{bmatrix} k \\ k \\ k \end{bmatrix} + c_o \geq \left( \pi_1 \begin{bmatrix} k/2 \\ 0 \\ 0 \end{bmatrix} + c_1 \right) \tau_1 \tag{4.6}$$

Since the value of the schedule $s_1(p)$ must be integer numbers, therefore, the first element of $\pi_1$, $\pi_{1,1}$ must divide two, i.e., $\pi_{1,1} = \frac{k}{2} = integer$. Let it be $\pi_{1,1} = 2\pi_{o,1}$. At point $p = [k\ 0\ 0]^T \in \Omega$, we have:

$$\pi_o \begin{bmatrix} k \\ 0 \\ 0 \end{bmatrix} + c_o \geq \left( \pi_1 \begin{bmatrix} k \\ 0 \\ 0 \end{bmatrix} + c_1 \right) \tau_1 \Rightarrow \pi_{o,1} k + c_o \geq (2\pi_{o,1} k + c_1)\, \tau_1 \tag{4.7}$$

To satisfy this inequality relation, $\tau_1 \leq 1/2$ is required. Let it be $\tau_1 = 1/2$. This procedure can be carried on to determine the other elements of the schedule vectors. As the relationship between the fundamental schedule vector and the multi-rate vector is defined as $\pi_j = \pi_f / \tau_j$, since $\tau_1 = 1/2$, therefore, $\pi_1 = 2\pi_o$. Depending on other dependence relations, $\pi_o$ can be determined, thus $\pi_1$ can also be determined.

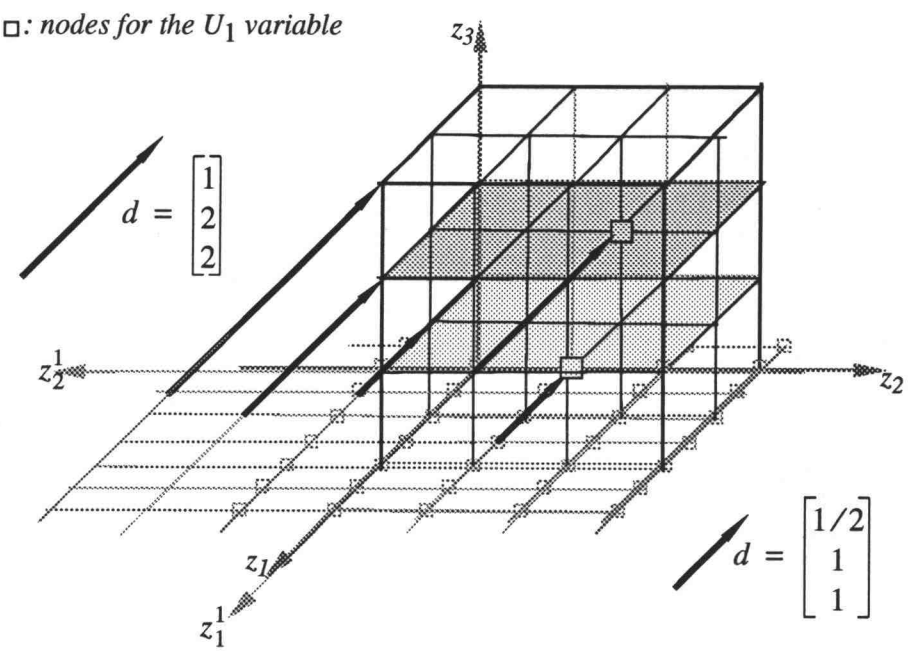



**Figure 4.1:** The dependencies of $U_o(p) \leftarrow U_1(D_1 p)$. The time values assigned to the nodes $p_q$ of $U_1$ must be less than the time values assigned those nodes $p$ of $U_o$ which depend on them.

After every index point $p_q \in \Omega_1 \subseteq Q^2$ (which is defined relative to the standard basis) has been assigned a time value, it is straight forward to further assign time values for each element of $U_1$ relative to its own basis. For example, if $p_q = (z_1, z_2, 0)$ has been assigned the time value $t_1$, then the element $U_1(2z_1, -z_2)$ is also assigned the value $t_1$.

Theorem 4.1 can be employed to compute a set of schedules for a structurally well defined ADIO, however, it is not a constructive way because the node sets are not explicitly defined in the algorithm. Recall that for every convertible ADIO, there exist corresponding specifications with regular dependence structures defined as follows:

$$
\begin{aligned}
U_i(p) &= f[U_1(p-d_{1i}), \ldots, U_j(p-d_{ji}), \ldots, U_V(p-d_{Vi})], \forall p \in \Omega_i \\
U_j(p) &= \begin{cases} U_j(p-d_j), \forall p \notin \Omega_j^{\varepsilon} \\ U_j(D_j p + d'_j)_{\mathbf{u}_j}, \forall p \in \Omega_j^{\varepsilon} \end{cases}
\end{aligned}
\tag{4.8}
$$

In this specification, all variables have been expanded so that they are defined over the entire computational domain and the dependencies are constant. As it was shown in example[3.15], since this specification is regular, therefore, it is possible to derive single rate schedules for this algorithm. However, as it was shown in that example, the use of single rate schedule may have many undesirable properties for VLSI implementations (e.g., multiple interconnections among processors and non-nearest neighboring interconnections). Therefore, multi-rate array and multi-rate schedule suit better for VLSI implementations. To derive a set of multi-rate schedules for the specification given in Eq.(4.8), we have the following theorem:

**Theorem 4.2:** For a given system of regular recurrence equations defined as:

$$U_i(p) = f[U_1(p-d_{1i}), \ldots, U_j(p-d_{ji}), \ldots, U_V(p-d_{Vi})], \forall p \in \Omega$$

$$U_j(p) = \begin{cases} U_j(p-d_{ji}), \forall p \notin \Omega_j^{\varepsilon} \wedge p \in \Omega \\ U_j(D_j p + d_j)_{\mathbf{u}_j}, \forall p \in \Omega_j^{\varepsilon} \end{cases} \tag{4.9}$$

where $d_j \in Z^m$ is the translation part in the original index mapping function. Then $\Lambda_s = \{s_f, s_1, \ldots, s_V\}$ defines a set of multi-rate schedules if and only if it satisfies the following conditions $\forall j \in \{f, 1, 2, \ldots, V\}$ :

(1). $\forall p_q \in Q^n, \sigma_j(q) = p_q, s_j(p_q) = integral$;

(2). $\pi_j d_{ji} \geq 1$;

(3). If $\gamma$ is a ray of $\Omega$, then $\pi_j \gamma > 0$;

(4). For every vertices in the domain, $\forall v \in \Omega$, $\pi_j v + c_j \geq 0$.

**Proof.** (1). Since all operations of $U_j$ are synchronized by the clock signal $\phi_j$, non-integral number cannot be used to define a clock signal. Moreover, $p_q$ corresponds to an integer point $q \in Z^m$ if it is defined in relative to the coordinate system $C_j^m$ for variable $U_j$, any action happening to a variable can only happen at integer number of clock signals. Therefore, $s_j(p_q)$ must be an integer number to be a valid timing function.

(2). Recall that $s_j(p) = \pi_j p + c_j$ computes the clock cycles for index point $p$. Since $U_j(p) \leftarrow U_j(p-d_{ji})$, the time value assigns to the point $(p-d_{ji})$ must be at least one clock cycle earlier, that is, the condition $s_j(p) - s_j(p-d_{ji}) \geq 1$ must be respected. $s_j(p) - s_j(p-d_{ji}) = \pi_j d_{ji} \Rightarrow \pi_j d_{ji} \geq 1$. The clock period is $\tau_j$, therefore, at least $\tau_j$ time difference should be assigned to points $p$ and $(p-d_{ji})$.

(3). The same as the proof for theorem 4.1 (3).

(4). The same as the proof for theorem 4.1 (4). ■

For a given localized algorithm, condition (1) of theorem 4.2 can be used to determine the clock rate ratio associated with each variable. Condition (2) can be used to formulate a linear programming problem for solving a set of multi-rate timing functions for the algorithm. Moreover, if we consider the dependence vectors in the vicinity domain as different dependence vectors, then Condition (2) can also be used to decide the clock rate ratio. Condition (3) is used to ensure that the clock increases in the direction of the ray thus that the time value will not become negative. And Condition (4) is used to determine the translation parts $c_j$ of the schedules.

**Remarks:** Conditions (2), (3) and (4) in theorem 4.2 are similar to the Quasi-Affine-Time-Function (QATF) given in [Qui84]. However, the index point in [Qui84] is strictly integer vector and it is concerned about the single rate schedule only. Here we are dealing with fractional index vectors and multi-rate schedules. Therefore, Condition (1) is necessary. The number of clock cycles must be integer numbers, however, the time value relative to the fundamental time function can be fractional numbers.

Geometrically, the clock rate ratios can be determined from the ratios of the scale sizes of different coordinate systems along the projection direction. This can be interpreted as follows: if the direction of projection is considered as the time axis, then the scale sizes of each coordinate along this time axis mark the clock periods for different clock signals.

**Example[4.2]:** Consider the localized decimation filter algorithm represented as follows:

$$\begin{cases} For\,(0 \le i, 2 < j \le 9) \\ \quad y(i,j) = y(i,j+1) + h(i,j)\,x(i,j) \\ \quad h(i,j) = h(i-1,j) \\ \quad x(i,j) = x(i+1,j+3) \\ For\,(0 \le i, 0 \le j \le 2) \\ \quad x(i,j) = x(3i-j)_{\mathbf{u}_x} \\ \quad h(0,j) = h(j)_{\mathbf{u}_h} \\ \quad y(i,N) = 0 \end{cases} \tag{4.10}$$

The dependencies in the above specification are given as:

$$d_y = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, d_h = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, d_x = \begin{bmatrix} -1 \\ -3 \end{bmatrix}, d_{x1} = \begin{bmatrix} -1/3 \\ -1 \end{bmatrix}, d_{x2} = \begin{bmatrix} -2/3 \\ -2 \end{bmatrix}$$

where $d_{x1}$, $d_{x2}$ are the dependence vectors of variable $x$ in the vicinity domain. They have the same orientation as $d_x$ but their norms are different from $d_x$'s. The index spaces for variables $y$ and $h$ are defined in the integer lattice points $p \in Z^2$. However, the index space of $x$ is defined as:

$$p_q \in \{(3,9), (3\tfrac{1}{3},9), (3\tfrac{2}{3},9), \ldots\}$$

Let the schedule for $x$ be $s_x(p_q) = \pi_x p_q + c_x$ and the schedule for variables $y$ and $h$ be defined as $s(p) = \pi p + c$. According to condition (1), if the elements of $\pi$ and $c$ are integers, then the schedule $s(p) = \pi p + c$ will be integral. However, for $s_x(p_q) = \pi_x p_q + c_x$, the first element of $\pi_x = \begin{bmatrix} \pi_{x1} & \pi_{x2} \end{bmatrix}$ must be dividable by 3. This determines the clock rate ration $r_x = 3$. Higher ratios are possible, e.g., $r_x = 6$. However, if $r_x = 6$ is selected for this algorithm, then dummy variables must be introduced to ensure the correct timing relationships. Introducing redundancy is not what we want.

This clock rate ratio can also be derived from $d_{x1}, d_{x2}$ by the application of Condition (2). However, the integer condition is still necessary here. Otherwise, the number 4 will also satisfy Condition (2), but the number 4 cannot be used as the clock rate ratio for this problem. From condition (2), we have the following inequalities:

$$\begin{cases} \pi d_y = \begin{bmatrix} \pi_1 & \pi_2 \end{bmatrix} \begin{bmatrix} 0 \\ -1 \end{bmatrix} = -\pi_2 \geq 1 \Rightarrow \pi_2 \leq -1 \\ \pi d_h = \begin{bmatrix} \pi_1 & \pi_2 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \pi_1 \geq 1 \\ \pi_x d_x = \begin{bmatrix} \pi_{x1} & \pi_{x2} \end{bmatrix} \begin{bmatrix} -1 \\ -3 \end{bmatrix} = -\pi_{x1} - 3\pi_{x2} \geq 1 \Rightarrow \begin{cases} \pi_{x1} \leq -1 \\ \pi_{x2} \leq -1/3 \end{cases} \end{cases} \tag{4.11}$$

These inequalities define a convex hull on which the parameters for the schedule can be selected. This convex hull is represented by the shaded area in Figure 4.2.

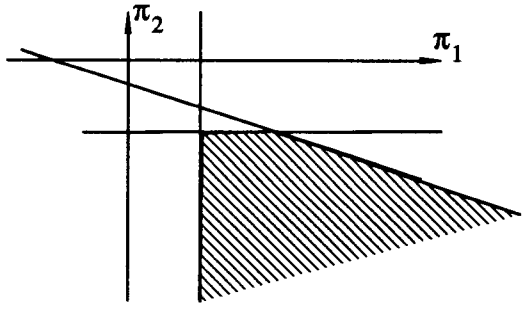



**Figure 4.2:** The convex hull defining the parameters of the schedules.

---

If we select the vertex point $\begin{bmatrix} 1 & -1 \end{bmatrix}$ of the convex hull as the schedule vector, that is $\pi = \begin{bmatrix} 1 & -1 \end{bmatrix}$, then we have the fundamental schedule given as $s(p) = z_1 - z_2 + c$. Then, from the relationship given in the definition, that is $s_x = r_x s$, we can easily derive the multi-rate schedule for variable $x$ to be $s_x(q) = 3z_1 - 3z_2 + c_x$. This selection satisfies

Condition (3) since the ray of the domain is $\gamma = [1\ 0]^T$. From Condition (4) and a vertex point of $\Omega$, $v = [3\ 9]^T$, we can determine that $c = 6$ and $c_x = 18$. ■

## 4.3 THE SYNTHESIS PROCEDURES

Chapter 3 provided procedures for dependence localization for ADIOs. Last section developed theorems for computing scheduling and timing functions for ADIOs and their corresponding regular specifications. This section provides systematic procedures for mapping ADIOs onto multi-rate VLSI arrays.

### 4.3.1 The synthesis of ADIOs without the phase component

This section outlines the synthesis procedure for the synthesis of MRAs from ADIOs without polyphase components. The synthesis of ADIOs with polyphase components is studied in next section. The synthesis procedure consists of the following steps:

1. Extract the sets of index mapping matrices from the given ADIO specification, using theorem 3.3 and 3.4 to determine if the given ADIO is regular specification convertible. If yes, go on to step 2; otherwise, use multiple directional propagation.
2. Determine the iteration spaces for each variable.
3. If $|\Delta_j| > 1$, select an index mapping matrix $D_{j,l}$, $1 \le l \le |\Delta_j|$ as the matrix for the construction of a coordinate system for $U_j$, $1 \le j \le V$. Invoke Procedure 3.1 to construct an MCS system for the ADIO.
4. If the index mapping matrices of variable $U_j$, $1 \le j \le V$ do not satisfy the sufficient condition Eq. (3.26), invoke Procedure 3.3 to compute a partition of its domain. Then invoke procedure 3.2 to construct a system of coordinate systems for each region of the domain.
5. For each variable $U_j$, $1 \le j \le V$, select an iteration orientation for it and decide the origins for each coordinate system. Find the embedding functions for each variable and embed each variable array into the computational domain $\Omega$.

6. Transform the ADIO into a system of regular specification by removing broadcasting variables using pipelining technique along their iteration spaces respectively.
7. Determine an allocation function $a(p) = \Lambda_a p + c_a$. Compute a set of multi-rate timing functions $\Lambda_t = \{t_f, t_1, \ldots, t_V\}$. Apply these functions to the regular specification, derive a regular array architecture and design the internal PE structure.
8. If the resultant array satisfies the specification, stop; otherwise, select another $D_{j,l}$ and go back to Step 3 and continue to derive another implementation.

**Remarks:** In the above procedure, step 1 examines if the dependencies of a given ADIO can be localized using null-space propagation. If it can be localized by null-space propagation, the procedure finds a proper MCS system for the localization. However, if it does not satisfy the conditions, then multiple directional propagation must be employed. If the index mapping matrices associated with the output variable do not satisfy the conditions, the priority should be given to finding the embedding functions of the output variables. If the variable is an input variable, then the priority should be given to finding the embedding functions which can embed the entire $m$-dimensional array into the computational domain.

For algorithms with phase components in their index quantizations, such as the interpolation filter example, the synthesis procedure presented above needs to be modified. This will be discussed in next section.

The following example illustrate how to use the synthesis procedure for mapping algorithms onto VLSI arrays.

**Example[4.3]:** *Two-dimensional decimation filter.* A 2-dimensional decimation filter is defined as:

$$y(i,j) = \sum_{k=0}^{K} \sum_{l=0}^{L} h(k,l)\, x(M_1 i - k, M_2 j - l) \tag{4.12}$$

where $x(i,j)$ is the input signal array (let its size be $N \times N$), $h(k,l)$ is a $(K+1) \times (L+1)$ rectangular weight window, and $y(i,j)$ is the output signal array, $M_1$ and $M_2$ are the decimation factors. For the sake of simplicity, in this example, we are going to consider the following specific filter with $M_1$=$M_2$=2 and $K = L = 3$:

$$y(i,j) = \sum_{k=0}^{K} \sum_{l=0}^{L} h(k,l)\, x(2i-k, 2j-l), \quad 0 \le i,j \le \left\lfloor \frac{N+1}{2} \right\rfloor \tag{4.13}$$

This algorithm has a four-dimensional computational domain which is defined as: $\Omega = \{0 \le i,j \le \left\lfloor \frac{N+1}{2} \right\rfloor, 0 \le k \le K, 0 \le l \le L\}$ under a four-dimensional standard coordinate system $C_f^4$. Let us define a point in $C_f^4$ as $(i, j, k, l)$. The first step is to extract the index mapping functions for each variable in this ADIO. They are specified as follows:

$$D_y = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \quad D_h = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad D_x = \begin{bmatrix} 2 & 0 & -1 & 0 \\ 0 & 2 & 0 & -1 \end{bmatrix} \tag{4.14}$$

Since every variable in this ADIO has only one index mapping function, according to Theorem 3.3, Eq.(4.12) is RMCS convertible. The iteration spaces of each variable are determined as:

$$\begin{aligned} IS_y &= \aleph(D_y) = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ IS_h &= \aleph(D_h) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \\ IS_x &= \aleph(D_x) = \begin{bmatrix} 1 & 0 & 2 & 0 \\ 0 & 1 & 0 & 2 \end{bmatrix} \end{aligned} \tag{4.15}$$

From the index mapping matrices specified in Eq.(4.14) and following the Procedure 3.1, we can easily determine that there is only one coordinate system for $y$ and one for $h$, since there is only one possible submatrix extracted from either $D_y$ or $D_h$ which is not singular. These submatrices are defined as:

$$D_y(\{1,2\}) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, D_h(\{3,4\}) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{4.16}$$

The coordinate systems for variables $y$ and $h$ are defined as:

$$\begin{aligned} C_y^2 &= \{\mathbf{u}_y^1, \mathbf{u}_y^2\}, \mathbf{u}_y^1 = [1\ 0\ 0\ 0], \mathbf{u}_y^2 = [0\ 1\ 0\ 0] \\ C_h^2 &= \{\mathbf{u}_h^1, \mathbf{u}_h^2\}, \mathbf{u}_h^1 = [0\ 0\ 1\ 0], \mathbf{u}_h^2 = [0\ 0\ 0\ 1] \end{aligned} \tag{4.17}$$

No other coordinate system for either $y$ or $h$ arrays is possible. (Here we are concerned with different base vectors only, different origin has not been taken into account for different coordinate systems). In contrast, we have up to four different coordinate systems for the indexing of variable $x$. they can be derived from the following four nonsingular submatrices of $D_x$:

$$\begin{aligned} D_x(\{1,2\}) &= \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}, D_x(\{1,4\}) = \begin{bmatrix} 2 & 0 \\ 0 & -1 \end{bmatrix} \\ D_x(\{2,3\}) &= \begin{bmatrix} 0 & -1 \\ 2 & 0 \end{bmatrix}, D_x(\{3,4\}) = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \end{aligned} \tag{4.18}$$

Note that the total number of $2 \times 2$ submatrices of $D_x$ is six but two of them are singular matrices (rank 1). These rank deficit matrices cannot be used to determine coordinate systems for the 2-dimensional variable. Each valid submatrix will define a coordinate system for $x$. Following Procedure 3.1, we have the following corresponding coordinate systems:

$$\begin{aligned} C_{x1}^2 &= \{\mathbf{u}_{x1}^1, \mathbf{u}_{x1}^2\}, \mathbf{u}_{x1}^1 = [1/2\ 0\ 0\ 0], \mathbf{u}_{x1}^2 = [0\ 1/2\ 0\ 0] \\ C_{x2}^2 &= \{\mathbf{u}_{x2}^1, \mathbf{u}_{x2}^2\}, \mathbf{u}_{x2}^1 = [1/2\ 0\ 0\ 0], \mathbf{u}_{x2}^2 = [0\ 0\ 0\ -1] \\ C_{x3}^2 &= \{\mathbf{u}_{x3}^1, \mathbf{u}_{x3}^2\}, \mathbf{u}_{x3}^1 = [0\ 1/2\ 0\ 0], \mathbf{u}_{x3}^2 = [0\ 0\ -1\ 0] \\ C_{x4}^2 &= \{\mathbf{u}_{x4}^1, \mathbf{u}_{x4}^2\}, \mathbf{u}_{x4}^1 = [0\ 0\ -1\ 0], \mathbf{u}_{x4}^2 = [0\ 0\ 0\ -1] \end{aligned} \tag{4.19}$$

Among these coordinate systems, only $C_{x1}^2$ derived from $D_x(\{1,2\})$ lies at the boundary of the computational domain $\Omega$. It is used to imbed the input variable $x$ array at the boundary of $\Omega$, which is shown in Figure 4.3. Because we can present at most three dimensional subjects graphically, we must decompose the four-dimensional computational domain down to two subspaces, that is, the $(z_1, z_2)$ and the $(z_3, z_4)$ subspaces. The input variable $x$ and the output variable $y$ are imbedded into the $(z_1, z_2)$ subspace. The $h$ array is embedded into the $(z_3, z_4)$ subspace.

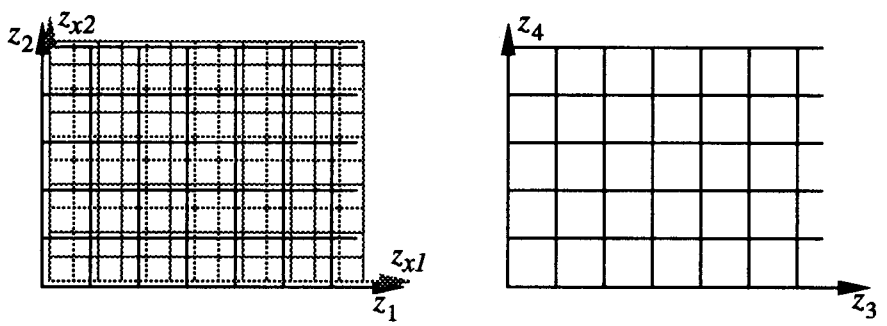



**Figure 4.3:** The MCS system for the 2-dimensional decimation filter. The 4-dimensional computational domain is decomposed down into two 2-dimensional subspaces.

---

The next step in the synthesis procedure is to find an embedding function for each variable. Iteration direction is selected for each variable for index space expansion Let the coordinate systems for $x$ and $h$ have the same origin as $C_f^4$, i.e., at $(0, 0, 0, 0)^T$. Let the origin of the coordinate for $y$ be at $(0, 0, K, L)^T$. Let the iteration orientation for $y$ be $[0\ 0\ 1\ 0]$ and $[0\ 0\ 0\ 1]$ (along the positive directions $z_3$ and $z_4$ axes respectively), let the iteration orientation for $h$ be $[0\ 0\ 1\ 0]$ and $[0\ 0\ 0\ 1]$ (along positive directions of $z_1$ and

$z_2$ respectively), and x along $[1\ 0\ 2\ 0]$ and $[0\ 1\ 0\ 2]$ directions, then the embedding functions are given as:

$$\begin{aligned}
&\sigma_y : Z^2 \to Z^4 \Rightarrow \sigma_y([i\ j]) = [z_1\ z_2\ 0\ 0] \\
&\sigma_h : Z^2 \to Z^4 \Rightarrow \sigma_h([i\ j]) = [0\ 0\ z_3\ z_4] \\
&\sigma_x : Z^2 \to Q^4 \Rightarrow \sigma_x([i\ j]) = \left[\frac{z_1}{2}\ \frac{z_2}{2}\ 0\ 0\right]
\end{aligned} \tag{4.20}$$

After embedding all variables into the computational domain $\Omega$, we can perform index expansion for all variables to remove those global dependencies and transform the ADIO into a system of regular specifications. However, if we perform the index space expansion operation for the output variable $y$ alone, then the ADIO in Eq.(4.13) is transformed into a system of ARE as given below:

$$\begin{aligned}
&For\quad 0 \le i,j \le \left\lfloor \frac{N+1}{2} \right\rfloor \\
&\quad For\,(0 \le k \le K) \\
&\quad\ For\,(0 \le l \le L-1) \\
&\qquad y(i,j,k,l) = y(i,j,k,l-1) + h(k,l)\,x(2i-k, 2j-l) \\
&\quad\ For\,(l = L) \\
&\qquad y(i,j,k,l) = y(i,j,k,l-1) + y(i,j,k-1,l) + h(k,l)\,x(2i-k, 2j-l) \\
&\qquad\ y(i,j,k,-1) = 0 \\
&\qquad\ y(i,j,-1,L) = 0
\end{aligned} \tag{4.21}$$

Eq.(4.21) is just one of many possible SARE specifications of the original algorithm. By selecting different iteration orientations, we may arrive at different SAREs. Performing index space expansion for $h$ array is similar to that of the index space expansion for $y$ except with different iteration space and orientation.

The index space expansion and dependence localization for $x$ dependencies deserve our special attention. Even though in multi-coordinate systems, they are treated by the same way as the other variables. After the embedding, all dependencies are bounded in each null space only. Then, we can localize those broadcast global data dependencies of $x$ by propagating the data samples of $x(i, j)$ along the null space of $D_x$, that is:

$$\aleph(D_x) = \begin{bmatrix} 1 & 0 & 2 & 0 \\ 0 & 1 & 0 & 2 \end{bmatrix}^T$$

This null space is 2-dimensional. The null space propagation can be carried out by propagating the $x$ elements first along the $[1\ 0\ 2\ 0]^T$ direction, then along $[0\ 1\ 0\ 2]^T$, or vice versa. This process is illustrated in Figure 4.4, where we have decomposed the 4-dimensional space down into two 3-dimensional subspaces.

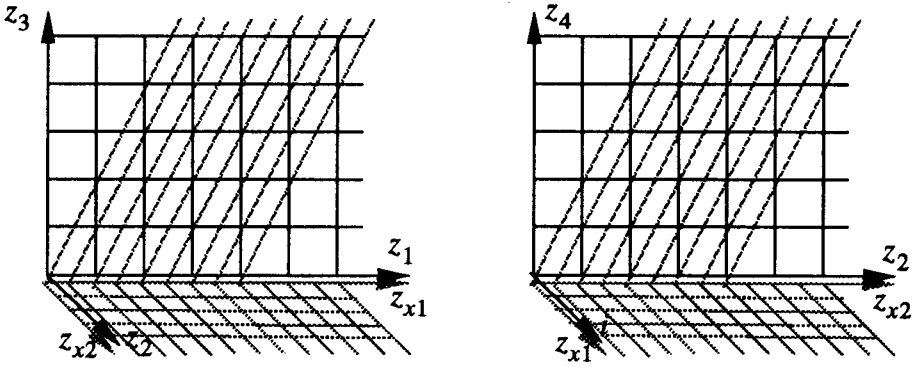



**Figure 4.4:** The null spaces for propagating the elements of $x$ variable array to localize the dependencies.

---

After the null-space propagation for dependence localization, the algorithm in Eq.(4.13) can be represented as a system of regular specifications in an MCS system as in Eq.(4.22). The window size parameters are given as $(K+1) \times (L+1)$. For simplicity, we

will consider the special case with $K = L = 3$, i.e., a 2-dimensional decimation filter of order $4 \times 4$.

$$
\begin{aligned}
&For\,(0 \le i,j \le \left\lfloor \frac{N+1}{2} \right\rfloor) \\
&\quad For\,(0 \le k \le K) \\
&\qquad For\,(0 \le l \le L-1) \\
&\qquad\quad y(i,j,k,l) = y(i,j,k,l-1) + h(i,j,k,l)\,x(i,j,k,l) \\
&\qquad For\,(l = L) \\
&\qquad\quad y(i,j,k,l) = y(i,j,k,l) + y(i,j,k-1,l) + h(i,j,k,l)\,x(i,j,k,l) \\
&\qquad y(i,j,k,-1) = 0 \\
&\qquad y(i,j,-1,3) = 0 \\
&\quad For\,(2 \le k \le K, l = 0) \\
&\qquad\quad x(i,j,k,l) = x(i-1,j,k-2,l) \\
&\quad For\,(1 \le l \le L) \\
&\qquad\quad x(i,j,k,l) = x(i,j-1,k,l-2) \\
&\quad For\,(0 \le k \le 1, l = 0) \\
&\qquad\quad x(i,j,k,l) = x(2i-k, 2j-l)_{ux} \\
&\quad For\,(i = 0, j = 0) \\
&\qquad\quad h(i,j,k,l) = h(k,l)_{uh} \\
&\quad For\,(j = 0, 0 < i \le \left\lfloor \frac{N+1}{2} \right\rfloor) \\
&\qquad\quad h(i,j,k,l) = h(i-1,j,k,l) \\
&\quad For\,(0 < j \le \left\lfloor \frac{N+1}{2} \right\rfloor) \\
&\qquad\quad h(i,j,k,l) = h(i,j-1,k,l)
\end{aligned}
\tag{4.22}
$$

To see a more complete dependence structure of this algorithm, let us project the localized DAG along the $z_2$ direction. Figure 4.5 shows this projected DAG partially. A small part of the dependencies is shown. After the projection, the entire subspace along the $z_2$-axis is projected to those points where $z_2 = 0$ and all the elements of each row *x(i)* of the input data array *x(i, j)* is concentrated to a single point in the projected DAG, which we present as *x(i, *)*. It is a very involved process to express the complete projected DAG in a three dimensional space, and if we do so it may do more harm than good. Here we only show the dependencies for $z_3 = 0$ because all other dependencies for $z_3 \neq 0$ are similar to those at $z_3 = 0$.

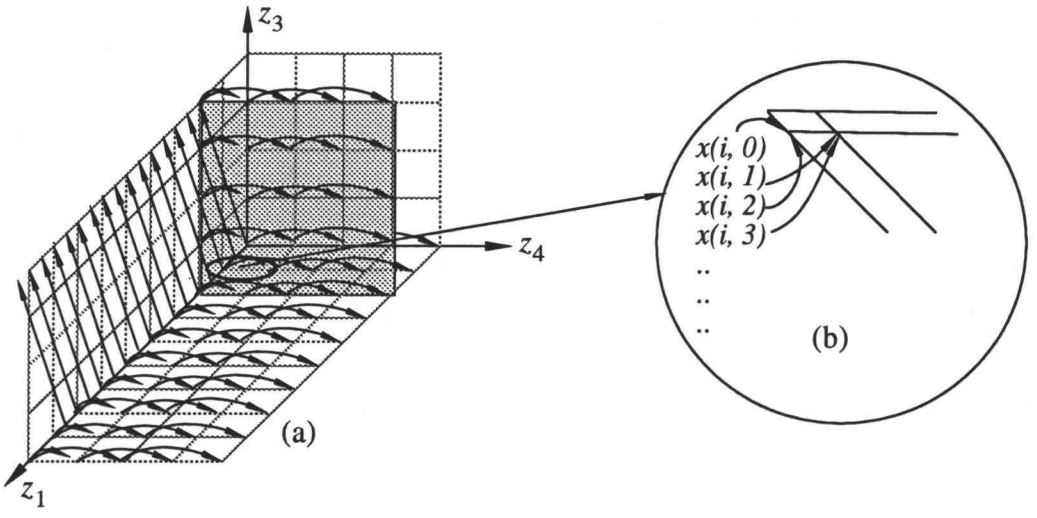



**Figure 4.5:** The projected DAG under an MCS system. (a) shows the dependence structure at $z_3 = 0$ and one layer; and (b) shows the detailed dependencies at $z_4 = 0$ and $z_4 = 1$.

With the understanding of the VLSI implementations of the 1-dimensional decimation filter MRA arrays, it is not difficult to derive an MRA architecture from this P_DAG.

If we have the luxury to afford a 3-dimensional processor array, then Figure 4.6 shows a 3-dimensional MRA array processor architecture. The time required for processing an $N \times N$ image frame is $O(N)$. The array has two clock frequencies $2\phi$ and $\phi$. $2\phi$ is the clock signal controlling the operations of variables $x$ and $\phi$ is the clock signal for $y$. $x(i, *)$ and $y(i, *)$ represent *row-i* of $x$ and $y$ respectively in Figure 4.6.

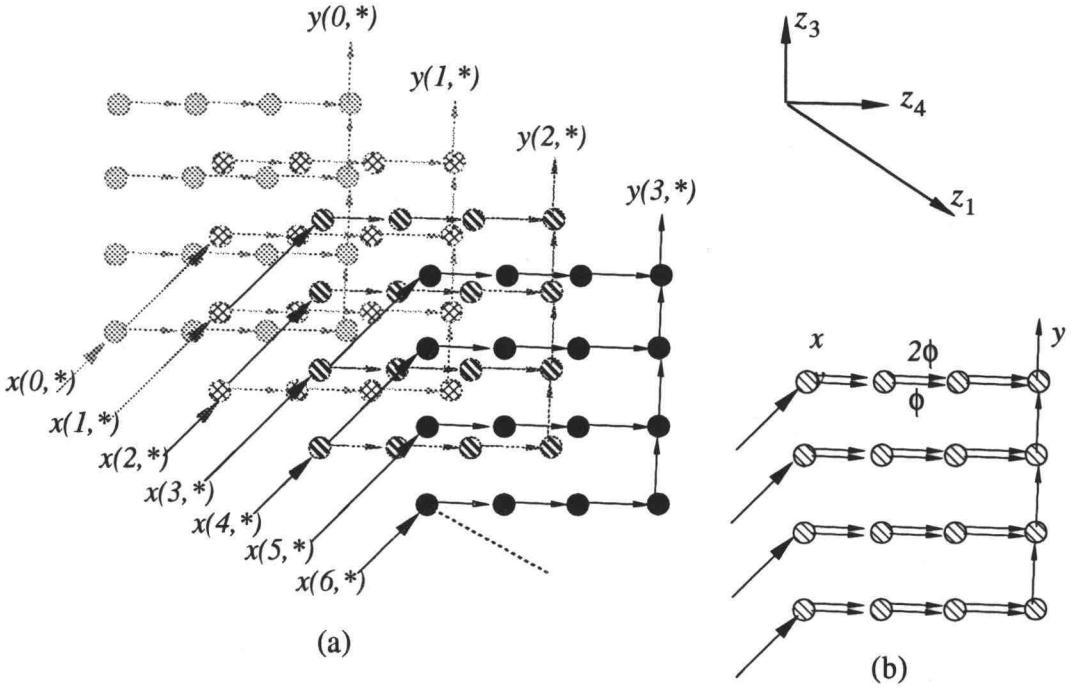



**Figure 4.6:** A three dimensional MRA implementation for the ADIO Eq.(4.13) with window size $4 \times 4$ and the input sample array $x(i, j)$ with size $N \times N$. The MRA array size is then $4 \times 4 \times (N+1)$. (a) shows part of the 3-D array; (b) shows a 2-dimensional array in more detail at each layer along $z_1$.

---

In reality, for VLSI implementation, we can hardly afford a three-dimensional implementation. The number of processing elements required by this 3-dimensional array

is tremendous even for a moderate image size of $512 \times 512$. Therefore, instead of a 3-dimensional implementation, a 2-dimensional MRA implementation would be much more practical for many applications. To reduce the dimension of the array, we need to project the projected DAG once more along another direction. In order to obtain an array whose size is almost independent of the problem size (it cannot be completely independent of the problem size because the number of internal storage elements required by the array will be a function of the problem size, but the others are independent of the problem size), we would project the projected DAG along $z_1$ direction.

Let the allocation function $a(p) = \Lambda_a p$ be defined by $\Lambda_a = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$. As a result, each node $p = [i\ j\ k\ l]^T \in Z^4$ in the DAG is mapped onto a 2-dimensional processor array as $a(p) = [k\ l]^T$

The resultant multi-rate array is given in Figure 4.7, which is the same as the one given in chapter 2 (Figure 2.8).

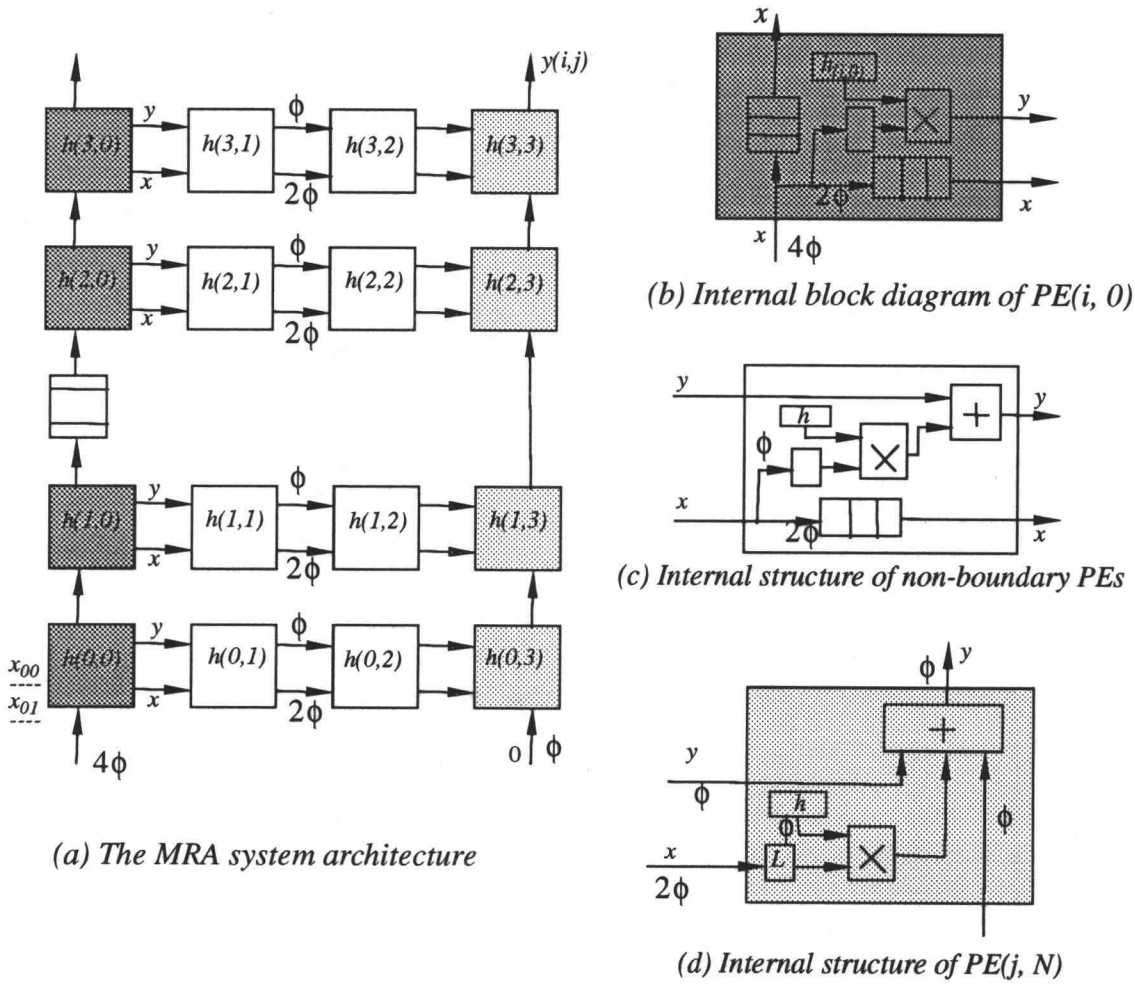



**Figure 4.7:** An MRA architecture for the 2-dimensional decimation filter with decimation factors $M_1 = M_2 = 2$ (a); Diagrams (b), (c) and (d) illustrate the internal block diagram of the processing elements.

### 4.3.2 The synthesis of ADIOs with phase components

Last section presented the synthesis procedure for ADIOs without considering the phase components. This section will generalize the synthesis procedure to allow the synthesis of ADIOs with polyphase phase components.

Assume that a given ADIO with a phase component $0 \le \varphi \le L-1$, the synthesis procedure starts at setting the polyphase component to zero, i.e., let $\varphi = 0$ and obtaining its dependence graph $DAG_{\varphi=0}$ using Procedure 4.1. After obtaining the MCS system and the dependence graph $DAG_{\varphi=0}$, the complete dependence graph can be obtained by shifting $DAG_{\varphi=0}$ in corresponding directions as $\varphi$ increases. This procedure is described as follows:

**Procedure 4.2:** Synthesizing ADIOs with Phase Component $0 \le \varphi \le L-1$

1. Let $\varphi = 0$, invoke Procedure 4.1 to construct the dependence graph $DAG_{\varphi=0}$ with a node called the *starting node* which is located at the origin of the standard coordinate systems. A corresponding regular specification of the ADIO with $\varphi = 0$ also can be obtained. The above operations should have finished the construction of the MCS system, the selection of embedding functions, and the index space expansion operation.
2. Let $\varphi = \varphi + 1$, if $\varphi < L$ Do
   2.1. *for* $(1 \le j \le V)$, for $U_j(\Gamma_j(p) + \varphi_j)$, if $\varphi_j = \varphi$ (i.e., $\varphi_j \neq 0$ in the ADIO), let the *starting node* of $DAG_{\varphi=0}$ be located at $(\varphi)_{\mathbf{u}j}$, change the corresponding labels in $DAG_{\varphi=0}$ from $U_j(\Gamma_j(p))$ to $U_j(\Gamma_j(p) + \varphi)$. This is the dependence graph for the phase component at $\varphi$, denote it as $DAG_{\varphi}$.
   2.2. In $DAG_{\varphi}$, if the label $U_j(\ )$ of a dependence vector pointing into a node $p$ is the same as those labels in $DAG_{\varphi=0}$ to $DAG_{\varphi-1}$, in other words, variable $U_j$ does not have a polyphase component, pipeline the signal by drawing a dependence vector from the corresponding node in $DAG_{\varphi-1}$ to node $p$ and remove the original vector and label.
3. In the DAG constructed in Step 2, index nodes with dependence vectors going into and leaving them (fractional index points relative to the standard basis are allowed) are the nodes where computations need to be performed. On the other hand, those nodes with dependence vectors cross over each other are the dummy nodes where no computations will occur, they may be removed. If the dummy nodes are removed, re-indexing and scaling may be required. This step is optional.
4. Find an allocation function $a(p)$ and a set of multi-rate timing functions.

Recall the subband coding system presented in Chapter 2, where the decimation and interpolation filters are the key construction blocks.

**Example[4.4]:** A three-bands subband coding system is depicted by Figure 4.8.

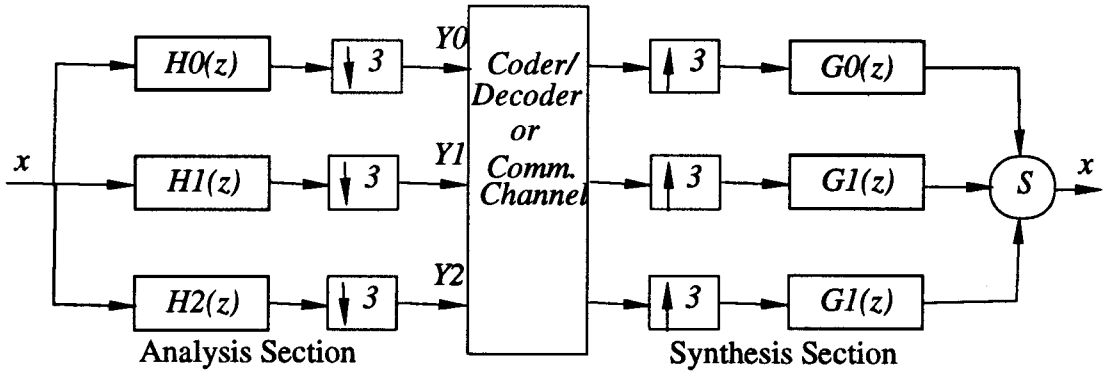
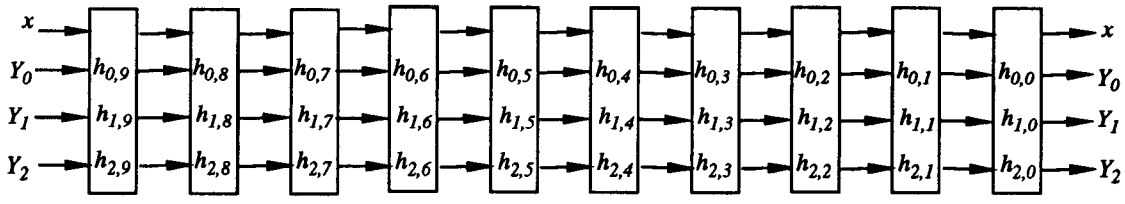
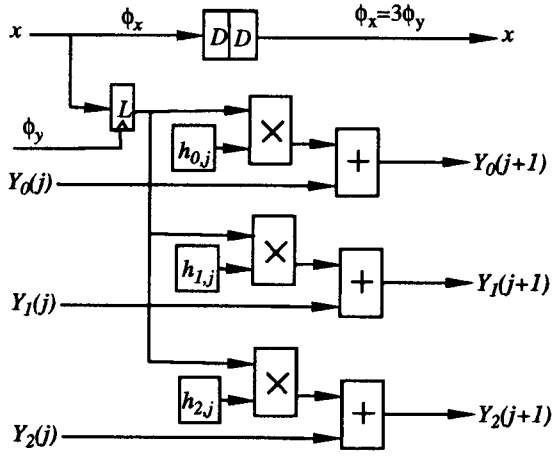



**Figure 4.8:** The three-band critically sampled subband coding system.

In the analysis section, the decimation filter bank can be implemented using the decimation filters derived in Chapter 3. Because the data dependencies of each decimation filter are identical except the values of the filter coefficients are different, the data path for propagating the $x$ variable can be shared by all decimation filters. If the MRA architecture in Figure 3.31 is employed to implement this filter bank, the analysis section has a structure as shown below in Figure 4.9:




*(a) The three-band decimation filter bank*




L: Latch. D: Delay.

*(b) Internal block diagram of the processing element*

**Figure 4.9:** The analysis filter bank of the three-band subband coding system.

---

The synthesis section of the subband coding system consists of interpolation filter banks. An interpolation filter with up-sampling factor $L$ can be represented as:

$$\begin{cases} 0 \le \varphi \le L-1 \\ \bar{x}(iL+\varphi) = \sum_{l=0}^{\left\lceil \frac{N-1}{L} \right\rceil} g(jL+\varphi)\, y(i-j) \end{cases} \tag{4.23}$$

The computational domain is defined as $\Omega = \{[i\ j]^T \mid 0 \le i, 0 \le j \le N-1\}$. For a three band subband coding system, we have $L = 3$. For the sake of simplicity, let $N = 10$. The synthesis procedure of this ADIO is as follows:

**Step 1:** Set the phase component $\varphi = 0$. Eq.(4.23) is represented as:

$$\bar{x}(3i) = \sum_{l=0}^{3} g(3j)\, y(i-j) \tag{4.24}$$

The null space of each index mapping matrix and the corresponding coordinate systems are defined as follows:

$$\aleph(D_{\bar{x}}) = [0\ 1],\ \aleph(D_g) = [1\ 0],\ \aleph(D_y) = [1\ 1]$$

$$C_{\bar{x}}^1 = \{\mathbf{u}_{\bar{x}}\},\ \mathbf{u}_{\bar{x}} = [1/3\ 0],\ C_g^1 = \{[0\ 1/3]\},\ \mathbf{u}_g = [0\ 1/3]$$
$$C_y^1 = \{\mathbf{u}_{g1}\},\ \mathbf{u}_{g1} = [1\ 0],\ C_y^1 = \{\mathbf{u}_{g2}\},\ \mathbf{u}_{g2} = [0\ -1]$$

If we select the iteration orientations for variables $x$, $g$, and $y$ to be along the directions $[0\ 1]$, $[1\ 0]$, $[1\ 1]$ respectively, then Eq.(4.24) can be transformed into the following system of regular specifications where $\Omega = \{p \mid (p \in Z^2) \wedge (0 \le i, 0 \le j \le 3)\}$:

$$\begin{aligned}
&\forall (i,j) \in \Omega \\
&\quad \bar{x}(i,j) = \bar{x}(i,j-1) + g(i,j)\, y(i-1,j-1) \\
&\quad g(i,j) = g(i-1,j) \\
&\quad y(i,j) = y(i-1,j-1) \\
&\quad \bar{x}(i,0) = \bar{x}(3i)_{\mathbf{u}\bar{x}} \\
&\quad g(0,j) = g(3j)_{\mathbf{u}g} \\
&\quad y(i,0) = y(i)_{\mathbf{u}y}
\end{aligned} \tag{4.25}$$

**Step 2:** Construct the complete regular specification from Eq.(4.25). First introduce the phase component back into the index quantization, then increase the phase component $\varphi$

from 0 to 1 and to 2 respectively. Graphically, relocate the starting node of the dependence graph $DAG_{\varphi=0}$ from $p = [0\ 0]$ to $p = [1/3\ 1/3]$ and to $p = [2/3\ 2/3]$ respectively. These operations generate the following complete system of regular specifications for Eq. (4.23):

$$
\begin{aligned}
\Omega_q = \{ [i_q\ j_q] \mid\ & (i_q, j_q \in Q_3) \wedge (0 \le i_q, 0 \le j_q \le 3) \wedge (i_q - j_q = integer) \} \\
& \forall (i,j) \in \Omega_q \\
& \bar{x}(i,j) = \bar{x}(i,j-1) + g(i-1,j)\, y(i-\tfrac{1}{3}, j-\tfrac{1}{3}) \\
& g(i,j) = g(i-1,j) \\
& y(i,j) = y(i-\tfrac{1}{3}, j-\tfrac{1}{3}) \\
& \bar{x}(i,0) = \bar{x}(3i)_{u\bar{x}} \\
& g(0,j) = g(3j)_{ug} \\
& y(i,0) = y(i)_{uy}
\end{aligned}
\tag{4.26}
$$

Where $Q_3$ denotes the set of fractional numbers with the denominator equals to 3, that is $Q_3 = \{p \mid p \in (Z^2/3)\}$. The complete dependence graph is given in Figure 4.10. The deep dark nodes and dependence vectors represent those of $DAG_{\varphi=0}$; the gray ones represent those of $DAG_{\varphi=1}$ and $DAG_{\varphi=2}$ respectively. $DAG_{\varphi=1}$ and $DAG_{\varphi=2}$ are obtained by relocating the starting node of $DAG_{\varphi=0}$.

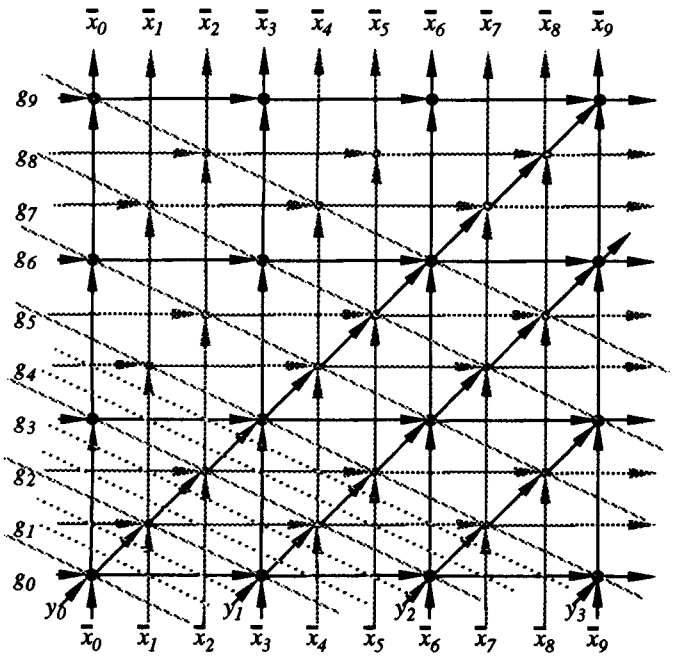



**Figure 4.10:** The DAG for the interpolation filter. The dark nodes and vectors represent $DAG_{\varphi=0}$; the gray ones for $DAG_{\varphi=1}$ and $DAG_{\varphi=2}$.

**Step 3:** This is an optional step. It is possible to derive the set of multi-rate schedules from the DAG constructed in step 2 as shown in Figure 4.10. Computation is to be performed at a node $(i_q - j_q = integer)$. There are dummy nodes in this DAG. Without changing the input-output relationships, we can eliminate these dummy nodes. This can be done by re-indexing the variables and making the scale of the coordinate system for variable $g$ be the same as the standard basis. Changing the scale size for the coordinate for $g$ is because the index space of variable $g$ is used as the processor space. The processor space is preferred to have the same scale size as the standard basis. By eliminating the dummy nodes in Figure 4.10, the new DAG is as given in Figure 4.11.

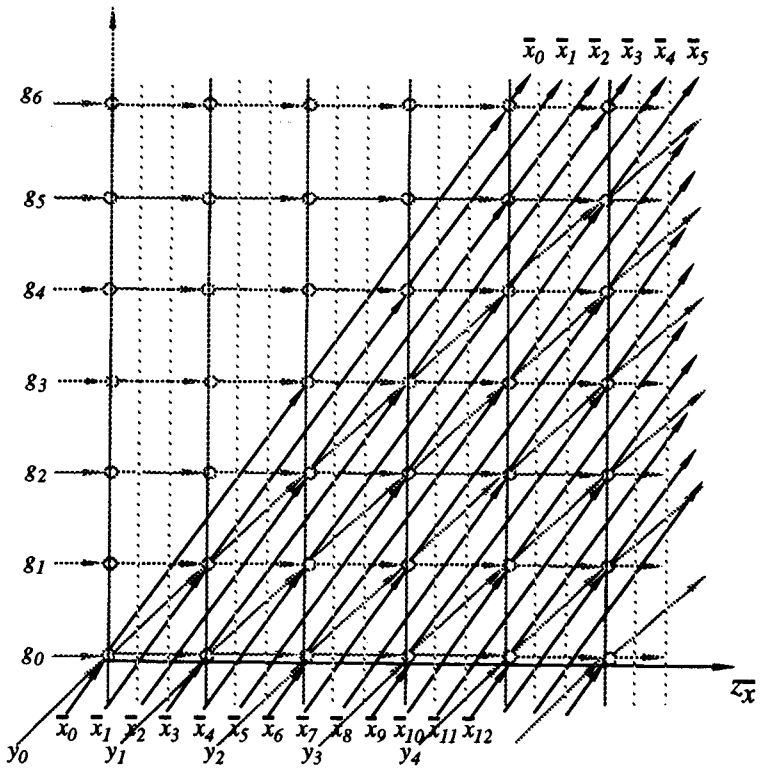



**Figure 4.11:** A modified DAG (partial) of the DAG in Figure 4.10.

---

The multi-rate timing functions can be determined as follows:

$$\begin{cases} \pi d_g = \begin{bmatrix} \pi_1 & \pi_2 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \pi_1 \geq 1 \\ \pi d_y = \begin{bmatrix} \pi_1 & \pi_2 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \pi_1 + \pi_2 \geq 1 \Rightarrow \pi_2 \geq 0 \\ \pi_{\bar{x}} d_{\bar{x}} = \begin{bmatrix} \pi_{\bar{x}1} & \pi_{\bar{x}2} \end{bmatrix} \begin{bmatrix} 2 \\ 3 \end{bmatrix} = 2\pi_{\bar{x}1} + 3\pi_{\bar{x}2} \geq 1 \Rightarrow \begin{matrix} \pi_{\bar{x}1} \geq 1/2 \\ \pi_{\bar{x}2} \geq 1/3 \end{matrix} \end{cases} \tag{4.27}$$

These inequalities define a convex hull on which the parameters for the schedule can be selected. This convex hull is represented by the shaded area in Figure 4.12.

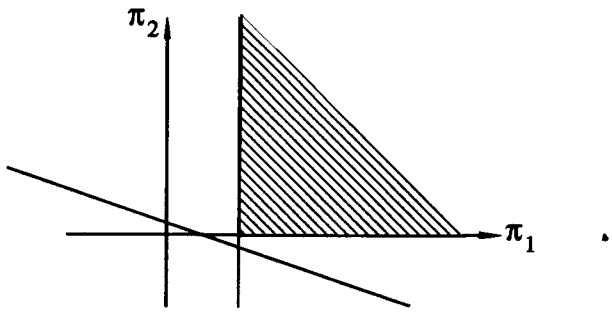
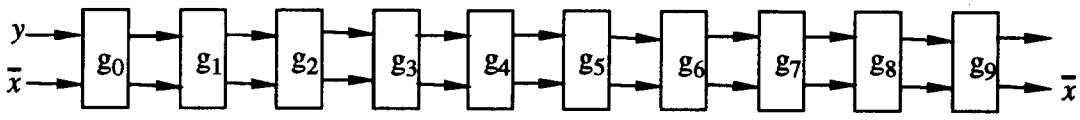



**Figure 4.12:** The convex hull defining where the parameters of the schedules can be selected from.

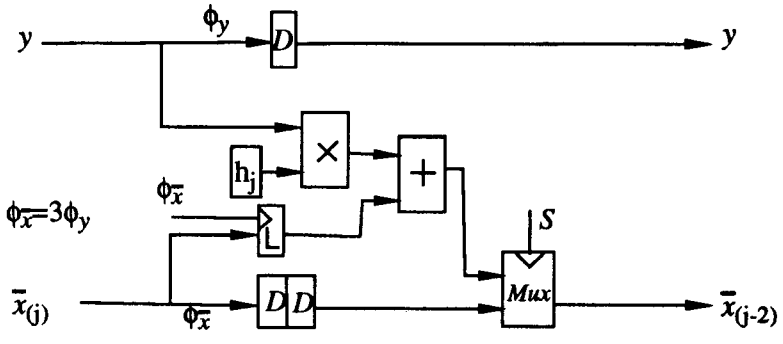
---

Projecting the DAG in Figure 4.11 along the $z_1$-direction onto the $z_2$-axis results in an MRA architecture as shown in Figure 4.13a, where each PE performs the computations involved with a single coefficient $g_j$, $0 \le j \le 9$. The system level architecture of this MRA interpolation filter is the same as the decimation filter with the clock rate ratio $r = 3$. However, the data elements of the computed variable $\bar{x}$ travel at the higher clock rate $\phi_{\bar{x}} = 3\phi_y$ instead of at a slower rate. Notice that despite that the computed variables propagate at the higher clock rate, the multiplication-accumulation unit is still work at the slower clock rate. This is possible because we have eliminated the operations which involve with zero valued operands. However, depending on the actual timing function, the internal PE architecture may vary with different selections of timing functions. If the schedule functions are selected as $t_y(p) = i$ (which is equivalent to $t_y(p) = i + 2j$ if it is determined from the DAG in Figure 4.10) for variables $y$ and $g$; and $t_{\bar{x}}(p_q) = 3i_q$ (which is equivalent to $t_{\bar{x}}(p_q) = 3(i_q + 2j_q)$ if it is determined from the DAG in Figure 4.10) where the subscript $q$ denotes that these coordinates can be fractional numbers). These schedules are shown in Figure 4.10 and Figure 11 respectively, where the gray hyperplanes show the time instants for variables $y$, $\bar{x}$ and $g$, while the dashed hyperplanes

are the time instants for $\bar{x}$ only. The resultant internal PE architecture is given in Figure 4.13b. In this MRA interpolation filter, the amount of time allocated to performing an multiply-accumulate operation is $(L-1)\,\tau_{\bar{x}}$ instead of $L\tau_{\bar{x}}$ (where $\tau_{\bar{x}}$ is the clock period of $\phi_{\bar{x}}$), this is because the computed partial result is due to arrive at the next PE by $(L-1)\,\phi_{\bar{x}}$ clock cycles.




*(a) A multi-rate array architecture for the interpolation filter*




*(b) Internal block diagram of the processing element*

**Figure 4.13:** An MRA implementation of the interpolation filter with $L = 3$.

---

For some applications, if the amount of time $(L-1)\,\tau_{\bar{x}}$ is not enough for performing a multiply--accumulate operation, then different schedule must be determined to allocate more time for the multiply-accumulate operation. If we select the new schedules as $t_y(p) = i+j$ for variables $y$ and $g$ for the DAG in Figure 4.11 (or $t_y(p) = i+5j$ for the

DAG in Figure 4.10), and $t_{\bar{x}}(p_q) = 3(i_q + j_q)$ for $\bar{x}$ in Figure 4.11 (or $t_{\bar{x}}(p_q) = 3(i_q + 5j_q)$, from the DAG in Figure 4.10), we can have a two level pipeline MRA implementation for this interpolation filter with the PE architecture as shown in Figure 4.14. Now the multiplication operation has been allocated with $L\tau_{\bar{x}}$ amount of time and the accumulation operation has been allocated with $(L-1)\tau_{\bar{x}}$ amount of time.

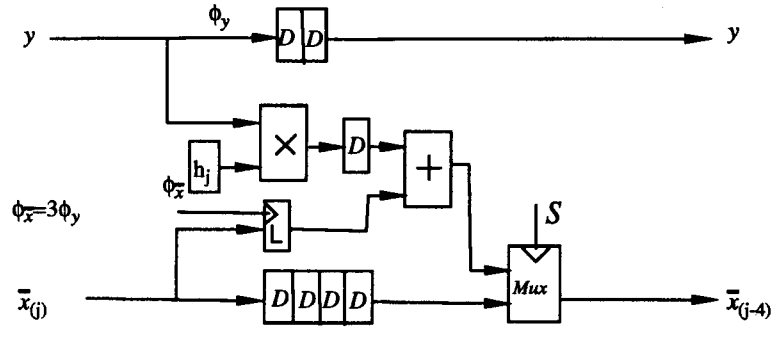



**Figure 4.14:** A two-level pipeline MRA implementation for the interpolation filter for applications require higher filtering speed.

---

The selections of different schedule functions can affect the speed requirement of a processing element, such as those two different designs shown above. While the selection of different iteration orientations for dependence localization may affect the total computational time and the area of each PE. For this interpolation algorithm, a more efficient array architecture can be derived from another DAG by selecting different iteration orientations for $\bar{x}$, $g$, and $y$ as $[0\ -1]$, $[1\ 0]$, $[-1\ -1]$ respectively. The resultant localized DAG is as shown in Figure 4.15. Projecting along the direction $z_1$, the resultant MRA architec-

ture is given in Figure 4.16. The schedule functions can be derived following the same procedure. They are given as $t_y(p) = i-4j+9$ and $t_{\bar{x}}(p_q) = 3(i_q-4j_q)+27$ respectively.

As compared to the implementation in Figure 4.13, this MRA architecture in Figure 4.16 allocates $L\tau_{\bar{x}}$ amount of time for the multiply-accumulate operation instead of $(L-1)\tau_{\bar{x}}$ for this operation.

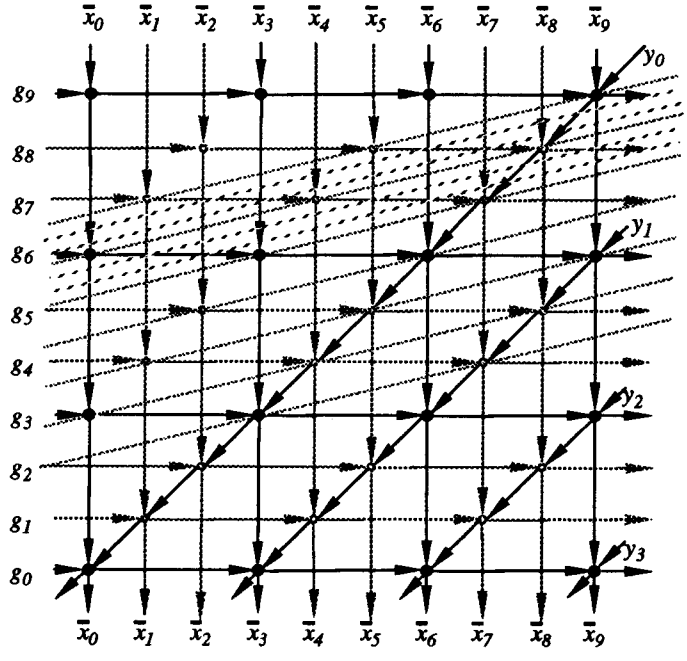
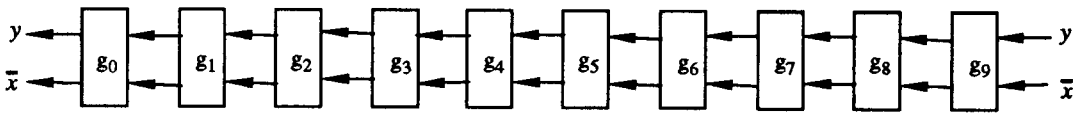
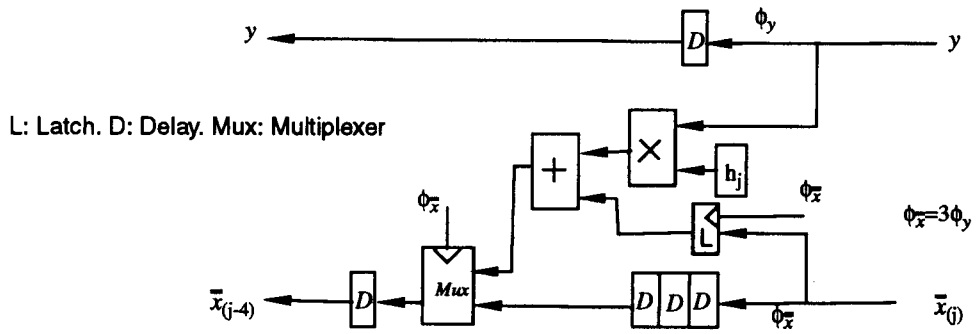



**Figure 4.15:** Another localized DAG which may yield a more efficient MRA implementation of the interpolation filter.




*(a) Another MRA architecture for the interpolation filter*




*(b) Internal block diagram of the processing element*

**Figure 4.16:** A corresponding MRA implementation for the interpolation filter from the DAG shown in Figure 4.15.

---

The MRA architecture is Figure 4.16 can be used in a similar way to construct an interpolation filter banks as the decimation filter banks. Figure 4.17 shows this interpolation filter bank. Figure 4.18 shows the complete subband coding system using these MRA filter banks.

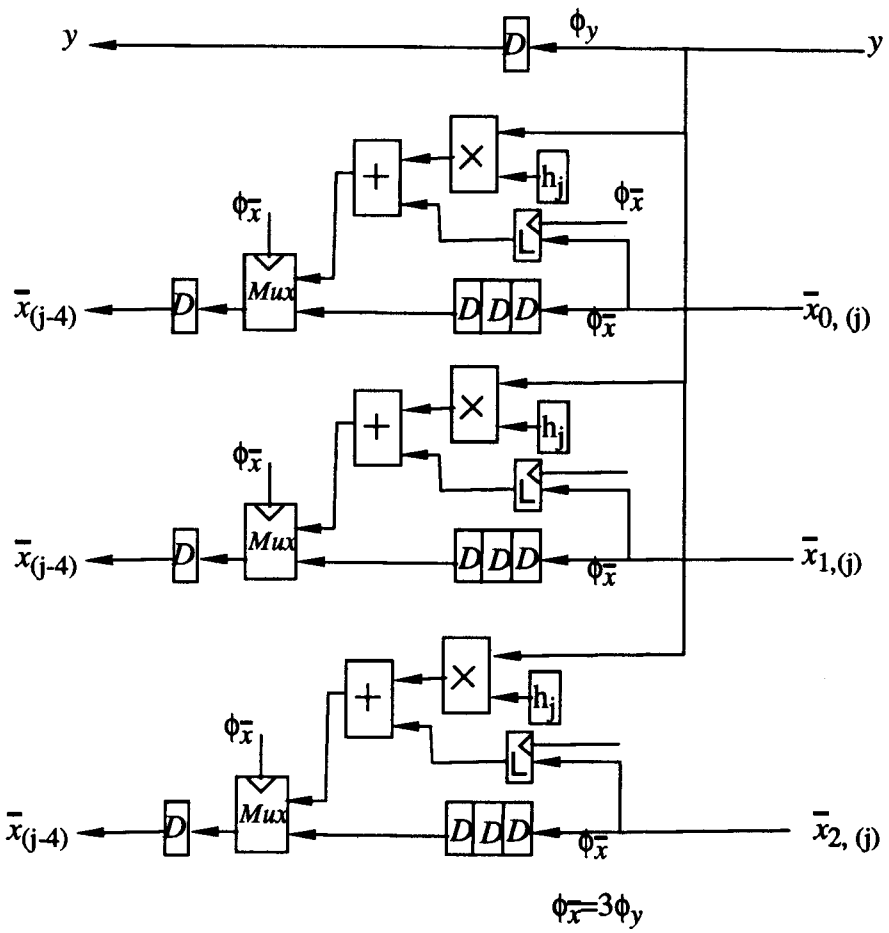



**Figure 4.17:** An MRA implementation for the interpolation filter.

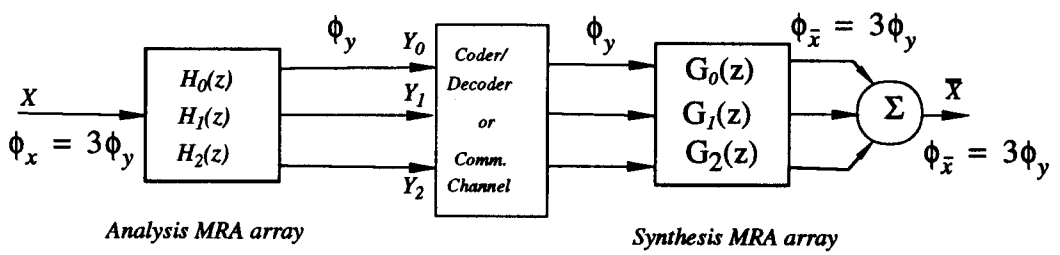



**Figure 4.18:** The three-band critically sampled subband coding system.

**Example[4.5]:** In digital audio, different sampling rates have been used for different media. Nonintegral sampling rate converters are used for copying materials from one medium to another. This is performed through the use of decimation filters with fractional decimation factors. Decimation filter with fractional decimation factor $M/L$ can be described by the block diagram shown in Figure 4.18. The implementation of such filters has drawn a great attention in recent DSP applications.

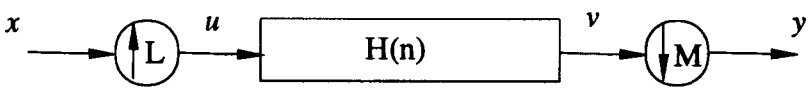



**Figure 4.19:** A block diagram for a decimation filter with fractional decimation factor $M/L$

---

An $N$th order decimation filter with fractional factor decimation $M/L$ can be expressed as the following set of equations:

$$\begin{cases} u(i) = \begin{cases} x(\frac{i}{L}), \frac{i}{L} = \text{integer} \\ 0, \text{otherwise} \end{cases} \\ v(i) = \sum_{j=0}^{N-1} h(j)\, u(i-j) \\ y(i) = v(Mi) \end{cases} \tag{4.28}$$

Direct implementation of the block diagram in Figure 4.18 or Eq.(4.28) results in arrays of extremely low efficiency, only one out of $(L \times M)$ computed result is truly needed to be computed. This efficiency is derived as follows: first, there are $(L-1)$ out of $L$ computations involved with the multiplication of zero valued samples; second, only one

out of $M$ computed result is used as the output. Polyphase implementations [Vai90] present an efficient approach to improve efficiencies of these realizations, but regular interconnection pattern has been greatly hampered by the continuous redrawing of the block diagrams. Furthermore, these polyphase implementations cannot be programmed to suit for different applications.

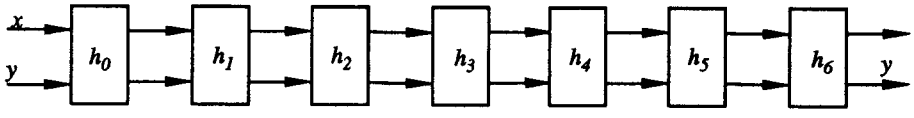
Eq.(4.28) can be transformed into an ADIO specification as follows:

$$\begin{cases} for \quad (0 \le \phi \le L-1) \\ y(Li+\phi) = \sum_{j=0}^{\left\lceil \frac{N-1}{L} \right\rceil} h(Lj+\phi)\, x(Mi-j) \end{cases} \tag{4.29}$$

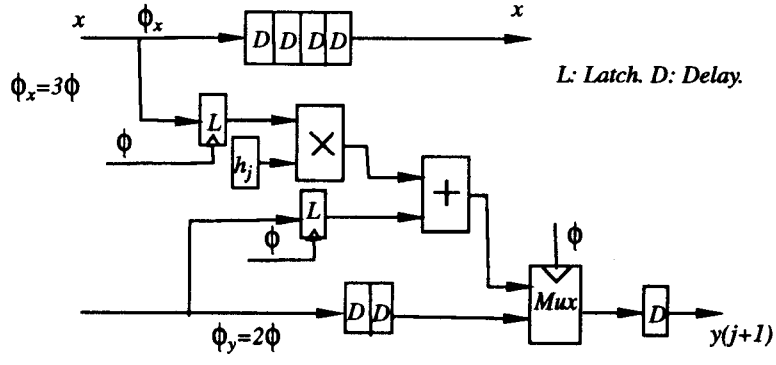
For the ease of illustration, let us derive the MRA architecture for the decimation filter with $L = 2, M = 3$. This ADIO can be synthesized following the synthesis procedure to find out the MCS system, carry out the embeddings of variables and perform dependence localization, and obtain a system of regular specifications. Then the multi-rate timing functions can be derived from these regular specifications. Similar to the designs of decimation and interpolation filters with integral factors, different schedules and iteration orientations will yield different processing element structures.

Here we are not going through these steps again for deriving an MRA array for Eq.(4.29). After we have derived the MRA structures for decimation and interpolation filters with integral factors, it is not difficult to derive a structure for the filter with fractional factors. It can be constructed by combining the interpolation and decimation filters. There three clock rates are used for the propagation of different data. The multiply-accumulate unit operates at the slowest clock rate $\phi$, the filter output propagates at the clock rate $2\phi$ and the input sequence propagates at the highest clock rate $3\phi$. The system MRA architecture and the internal PE structure are illustrated in Figure 4.19. These structures share the

basic structure of the decimation and interpolation filters. The operation of this MRA array is similar to the operation of the MRA decimation filter and the interpolation filter with integral decimation/interpolation factors.




*(a) Multi-rate array for the decimation filter*




*(b) Internal block diagram of the processing element*

**Figure 4.20:** An MRA array of the decimation filter with $M = 2/3$.

## 4.4 Conclusions

In this chapter we first introduced the multi-rate schedules and multi-rate timing functions. Necessary and sufficient conditions have been provided for the determination of the existence of multi-rate schedules and multi-rate timing functions. Such conditions can be formulated into a linear programming problem to derive these schedules and timing functions. Then the synthesis procedure for the automatic mapping of ADIOs without

phase component are presented. Using this procedure, a two-dimensional MRA decimation filter structure has been derived.

The synthesis procedure has been extended to allow the synthesis of ADIOs with phase components. Different applications have been synthesized from their algorithmic specifications onto MRA architectures. By controlling the clock rate ratios and by making the number of delays in each interconnection programmable, these MRA architectures will become programmable and therefore can be used for a larger class of applications.

# Chapter 5

# Conclusions

In this dissertation, we have proposed and developed the multi-rate array as the target architecture for a larger class of algorithms. We have also developed the synthesis technique multi-rate transformations for mapping DAREs onto MRAs. However, beside the MRA architecture, the major contributions of this dissertation are the introduction of a new indexing mechanism, multi-coordinate systems and the development of a unified synthesis theory based on the MCS technique. This theory enables a designer to start the synthesis procedure from the initial algorithmic specifications, instead of other low level specifications.

Single rate array architecture was discussed and the limitations of SRA architecture investigated. The multi-rate array architecture was proposed and developed as a more general target model for the synthesis of DSP and other algorithms onto VLSI systems. Algorithms rarely appear as uniform recurrence equations or regular iterative algorithms. These specifications must be derived from original algorithmic specifications. Most existing synthesis theories start from these low level specifications and leave to the designer the hard task of deriving these low level specifications from their original algorithmic specifications. Moreover, most existing synthesis methodologies require that the index mapping matrices of algorithms be totally unimodular. This requirement has excluded a large class of algorithms to be synthesized onto VLSI systems.

In this dissertation, the concept of multi-rate (systolic) array architecture has been revived and materialized by the designs of MRA decimation and interpolation filters. The

introduction and application of the multi-coordinate systems have made it possible to synthesize algorithms from their original algorithmic specifications, including those algorithms whose index mapping matrices are not totally unimodular. Moreover, the synthesis procedures developed in this dissertation can be used to map a class of general algorithms onto VLSI systems with minimal interconnections.

In Chapter 2, notations and some basic definitions used in this dissertation are provided. Then we give a definition for the new target array architecture, i.e., the multi-rate array (MRA) architecture. The MRA architecture was not really a new concept; many researchers had already mentioned the concept of multi-rate systolic array and multi-rate array. It has been believed that such MRA architectures are more realistic and would suit VLSI implementations better. They would have higher performances than their single rate counterparts [Lev84, Rao85, Kun88, Roy88, LK90]. These advantages of MRA architectures over SRA architectures have been recognized. However, there was no concrete model of MRA architectures, nor any executable MRA architecture showing how such architectures operate.

The real challenges did not come from trying to understand the advantages of MRA architectures, but on the nature of a concrete model for MRA architectures, how they operate, and how to design them. The more challenging task is the development of a synthesis methodology for automatically deriving MRA architectures from a large class of algorithms. A class of algorithms termed directional affine recurrence equations (DARE) was defined and their properties investigated. The most distinguishing feature of DAREs is that their dependence structures can be decomposed into uniform and non-uniform subspaces. When these dependence structures are projected along the non-uniform subspaces ($\mu$'s) into their corresponding uniform subspaces, the resultant array architectures have constant interconnections. Regular structure is an important property for effective VLSI implemen-

tations. Other properties of DAREs include families of parallel hyperplanes with constant dependence vectors. These hyperplanes are useful for scheduling the multi-rate variables. The MRA concept was developed in this chapter by the VLSI implementation of the multi-rate decimation filter architectures. MRA architecture is indeed superior to its single rate counterpart. It is a suitable target architecture for implementing algorithms, including those with non-totally unimodular index mapping matrices. The multi-rate transformation technique has been developed and can be used to map DAREs onto regular VLSI multi-rate architectures.

The synthesis procedure based on the DARE specifications shares some of the problems and limitations with existing methodologies. 1) DAREs are low level specifications like ARE and RIA, which must be transformed from their original algorithmic specifications by the heuristic method. 2) If in a given algorithm, there exist more than one DARE dependence relationship, but they have different uniform subspaces, or if $\mu_1 \neq \mu_2$, then there does not exist any projection vector $\mu$ which can produce a regular array architecture. 3) If the computational domain has a ray $\gamma$ which is different from the uniform projection vector, then the resultant array architecture either has an irregular interconnection pattern or an infinite number of processing elements. Both cases are not suitable for VLSI implementations. Moreover, this synthesis procedure cannot be of much help in the design of internal processor structures of MRA arrays.

In the development of methodologies presented in the last chapter and in those reported by other researchers, there has been an implicit assumption that the index spaces of an algorithm are defined as relative to a single coordinate system, the standard coordinate system. This assumption has a delicate impact on the analysis of the true dependence structures of algorithms. It is this assumption which makes the development of a unified synthesis methodology for all linearly (affine) indexed algorithms so frustrating.

In Chapter 3, we first defined the affine direct input output (ADIO) specifications. ADIOs are the original algorithmic specifications for most applications. The ADIO specifications aim at reducing redundant computations in algorithms introduced by the use of intermediate variables. Intermediate variables are usually employed to transform non-uniform algorithms into regular specifications. It was shown that most DSP algorithms, matrix manipulation algorithms and many others can be conveniently presented as ADIOs. The use of ADIOs as initial specifications allows a designer to explore the full solution spaces of given algorithms, which is important for design optimizations. However, the synthesis of VLSI systems from ADIOs is not a trivial task, because of the non-uniform dependence structures, the use of global operators such as summation and product operators, and the different dimensional variable arrays in ADIO specifications. However, the most challenging difficulty comes from the fact that the index mapping matrices in a given ADIO may not be idempotent nor totally unimodular, which are restrictions imposed by conventional synthesis methodologies.

To overcome all of the difficulties in synthesizing MRA arrays from ADIOs, we introduced the multi-coordinate systems (MCS) for the data dependence analysis of algorithms and dependence localization. Under a system of multi-coordinate systems, the index spaces of different variables in a given algorithm may be defined relative to different coordinates systems. Each variable may have its own coordinate systems. In an MCS system, there is a standard coordinate system used to define the computational domain and as the reference system for other coordinate systems. Each coordinate system may have different dimensions and a different basis. The applications of the MCS technique for dependence structure analysis were illustrated by examples.

We then investigated the most important dependence properties of ADIOs. A conclusion was derived: the right null space data dependence property does not depend on

whether the index mapping matrices are idempotent or not. Those index points lying in the same null space always depend on the same data element. This dependence relationship is not affected by the idempotent property of the index mapping matrices. However, if all index spaces in an algorithm are defined in relation to a single coordinate system, then the idempotent condition is necessary for using the null space propagation technique for dependence localization. This was derived in [RTRK88] where an implicit assumption was made of the use of a single coordinate system for defining the index spaces of an algorithm.

But, under the MCS system the right null spaces $\aleph(D)$ of index mapping matrices can always be used effectively for dependence localization. Such observation is important because then unidirectional propagation, instead of multiple directional propagation, can be used to localize most data dependencies. This has reduced the requirement for processor interconnections and simplified the control mechanisms in VLSI implementations. Procedures have been provided in this chapter for the construction of multi-coordinate systems from given algorithms. Constructing the proper MCS systems for given algorithms is important because only then data dependence localization can be done through the use of null space data propagation solely, and no other directional propagation is required. In contrast, those conventional techniques employ multiple directional propagation for data dependence localization when idempotent condition is not satisfied. Multiple directional propagation requires multiple interconnections for a variable in VLSI implementations.

The ultimate objective is to map algorithms onto regular VLSI systems. Only algorithms with regular dependence structures can be mapped onto regular array architectures. Given an algorithm, it is important to decide if it can be implemented onto regular array architectures. Sufficient and necessary conditions are provided in Chapter 3 to detect if an algorithm can be transformed into regular specifications. This is followed by the proce-

dures for transforming algorithm specifications with global dependencies into regular specifications. These transformations require the operation called index space expansion. Index space expansion retrieves the iterative and regular properties from an algorithm and expands an $m$-array to an $n$-array. Examples have been provided for the applications of MCS systems for dependence structure analysis and dependence localization.

The convertibility condition investigated in Chapter 3 was necessary under the assumption that only the right null space propagation technique is used for dependence localizations. For some applications, this assumption may be too restrictive. If these conditions cannot be satisfied, the multiple directional propagation technique still can be employed for dependence localizations. Even for such applications, the MCS technique will still be useful in helping a designer decide how to embed each variable array into the computational domain. The matrix Lyapunov algorithm was used to illustrate how this could be done.

Chapter 4 dealt with the problems of scheduling algorithms onto MRA architectures. Existing theories are concerned with single rate schedules only. Multi-rate schedules and multi-rate timing functions were introduced and defined. Sufficient and necessary conditions were developed to determine if a given algorithm has a set of multi-rate timing functions. We have also developed theorems which can be used to formulate a linear programming problem for the derivation of a set of timing functions. Procedures have been provided for synthesizing ADIOs onto VLSI multi-rate systems. First, a procedure was developed for ADIOs without the phase components in their index functions. This procedure later was expanded to synthesize ADIOs with the polyphase components. Examples for synthesizing a two dimensional decimation filter, a one-dimensional interpolation filter, and a decimation filter with fractional decimation factor were presented. These filters can be easily used to construct subband coding systems for speech and image codings.

There are several open topics remain to be investigated:

1. What architecture is best for an application?

Multi-rate array is a more general architecture suitable for a larger class of applications. However, for some applications, it is advantageous to use asynchronous implementations, and occasionally to use broadcasting or bus architectures. For example, to solve the matrix Lyapunov equation and the Algebraic Path Problem (APP), systems with broadcast ability seem to solve the problem with less time, but the possibly longer delays introduced from this broadcast function must be considered.

The asynchronous approach can also be used to improve the performance of such systems. However, the control overhead for the asynchronous operations may make the seemingly faster system too costly and may also be proved to be slower than expected.

Multi-rate array can also be employed for the design for propagating the computed results at a faster rate so that a higher performance is achieved. But the synchronization of such an MRA architecture is believed to be more complex than the single rate architecture. The single rate array architecture might have the simplest control mechanism, but it is also the slowest. So, given an algorithm, is there any method to detect which implementation would be the best under certain specification?

2. Other multi-rate schedules?

The second interesting topic is the multi-rate schedule problem. If the restriction $\pi_j = r_j\pi_f$ is not imposed, will it produce a better set of multi-rate schedules than with the restriction?

# Bibliography


[AG89] G. S. Ammar and W. B. Gragg, "Numerical experience with a superfast real Toeplitz solver," Naval Postgraduate School, NPS-53-89-008, Feb. 1989.

[AK78] J. Agnew and R. C. Knapp, LINEAR ALGEBRA WITH APPLICATIONS, Brooks/Cole Publishing Company, Monterey, CA, 1978.

[Atr58] A.J. Atrubin, A STUDY OF SEVERAL PLANAR ITERATIVE SWITCHING CIRCUITS, S.M. Thesis, MIT, Dept. of Electr. Engr., 1958.

[BA93] D.G. Baltus and J. Allen, "Efficient exploration of non-uniform space-time transformations for optimal systolic array synthesis," in *Proc. 1993 Application Specific Array Processors*, Ed. L. Dadda and B. Wah, pp. 428-441, IEEE Computer Society Press, 1993.

[Ban90] Thomas F. Banchoff, BEYOND THE THIRD DIMENSION, Geometry, Computer Graphics, and Higher Dimensions. Scientific American Library, New York, 1990.

[Bat80] K.E. Batcher, "Design of a massively parallel processor," *IEEE Trans. Comput.* C-29, pp.836-840, 1980.

[Bet03] Enrico Betti, Opree matematiche, Vol. 1-2, Milano, 1903-1913.

[BK81] R. P. brent and H.T. Kung, "The area-time complexity of binary multiplication," *J. ACM*, 28(3), July 1981.

[BR90] A. Bennaini and Y. Robert, "Spacetime minimal systolic arrays for Gaussian elimination and the algebraic path problem," In Proc. Int. Conf. on Application Specific Array Processors, pp. 746-757, Princeton, IEEE Computer Society, Sept. 1990.

[BSG91] M.K. Birbas, D. J. Soudris,and C.E. Goutis, "Design methodology for direct mapping of iterative algorithms on array architectures," Intern. Symposium on Circuits and Systems, vol.5, pp.3058-3061, 1991.

[Bu90] J. Bu, *Systematic Design of Regular VLSI Processor Arrays*, Ph.D thesis, Delft University of Technology, Delft, The netherlands, May, 1990.

[Bun85] J. R. Bunch, "Stability of methods for solving Toeplitz systems of equations," *SIAM J. Sci. Statist. Comput.*, pp. 349-364, Vol. 6, 1985.

[Bur70] A.W. Burks, Editor, ESSAYS ON CELLULAR AUTOMATA, University

of Illinois Press, Urbana, IL, 1970.

[Cap82] P.R. Cappello, *VLSI Architectures for Digital Signal Processing*, Ph.D thesis, Princeton University, Princeton, NJ, Oct. 1982.

[Cap89] Peter R. Cappello, "A spacetime-minimal systolic array for matrix product," In J. V. McCanny, J. McWhirter, and E. E. Swartzlander Jr., editors, *Systolic Array Processors*, pp. 347-356, Prentice-Hall, Killarney, Ireland, May 1989.

[Cap92] P. Cappello, "A processor-time-minimal systolic array for cubical mesh algorithms," *IEEE Trans. on Parallel and Distributed Systems*, Vol. 3, no. 1, Jan. 1992.

[Che86] M. C. Chen, "A design methodology for synthesizing parallel algorithms and architectures," *J. of Parallel and Distributed Computing*, pp.461-491, Dec. 1986.

[Che88] M. C. Chen, "The generation of a class multipliers: synthesis highly parallel algorithms in VLSI," *IEEE Trans. on Computers*, Vol. 37, no. 3, pp.329-338, March 1988.

[CL88] P. R. Cappello and A. J. Laub, "Systolic computation of multivariable frequency response," *IEEE Trans. Autom. Control.* 33(6), pp.550-558, June, 1988.

[CM85] M. C. Chen and C. Mead, "Concurrent algorithms as space-time recursion equations," In S.Y. Kung, H. J. Whitehouse, and T. Kailath, editors, *VLSI & Modern Signal Processing*, Prentice-Hall, Englewood Cliffs, 1985.

[CMP90] Ph. Clauss, C. Mongenet, and G.R. Perrin, "Calculus of space-optimal mapping of systolic algorithms onto processor arrays," *1990 Int. Conf. on Application Specific Array Processors*, Sun-Yuan Kung, Earl E. Swartzlander,Jr; Jose A.B. Fortes and K. Wojtek Przytula, Ed.. IEEE Computer Society Press.

[Col69] S. N. Cole, "Real-time computation by $n$-dimensional iterative arrays of finite state machines," *IEEE Trans. Computers*, 18, pp.349-365, 1969.

[Cox91] H. S. M. Coxeter, REGULAR COMPLEX POLYTOPES, Second edition, Cambridge University Press, 1991. (QA691. C66 1991).

[CR81] C. W. Curtis and I. Reiner, METHODS OF REPRESENTATION THEORY, With Applications To Finite Groups and Orders, vol.1, John Wiley & Sons, New York, 1981.

[CR83] R.E. Crochiere and L.R. Rabiner, MULTIRATE DIGITAL SIGNAL PROCESSING, Prentice Hall, Inc., Englewood, 1983.

[CS83] P. R. Cappello and K. Steiglitz, "Unifying VLSI array design with geometric transformations," in H.J. Siegel and L.Siegel, Ed. *Proc. Int. Conf. on Parallel Processing*, pp.448-457, Belaire, MI, Aug. 1983.

[CS84] P. R. Cappello and K. Steiglitz, "Unifying VLSI array design with linear transformations of space-time," in F.P. Preparata, Ed. *Advances in Computing Research*, pp. 23-65, JAI Press, Inc. 1984.

[CV93] T. C Chen and P.P Vaidyanathan, "The role of integer matrices in multidimensional multirate systems," *IEEE Trans. on Signal Processing*, Vol.41, No.3, March 1993.

[CVV88] J. P. Charlier, M. Vanbegin and P. Van Dooren, "Systolic algorithms for digital signal processing," *Philips Journal of Research*, Vol. 43, pp. 268-290, Nos 3/4, 1988.

[DD90] Alain Darte and Jean Marc Delosme, "Partitioning for array processors, TR 90-23. Lah de Informatigue

[DI85] Jean Marc Delosme and Ilse C. F. Ipsen, "Efficient systolic arrays for the solution of Toeplitz systems: An illustration of a methodology for the construction of systolic architectures in VLSI," Research Report YALEU/DCS/RR-370, June 1985.

[DI86a] Jean-Marc Delosme and I.C.F Ipsen, "An illustration of a methodology for the construction of efficient systolic architectures in VLSI," In Proc. 2nd Int. Symp. on VLSI Technology, Systems and Applications, pp. 268-273, Taipei, 1985.

[DI86b] J.M. Delosme and I.C.F Ipsen, "Systolic array synthesis: computability and time cones," in M. Cosnard, P. Quinton, Y. Robert and M. Tchuente, Ed. *Parallel Algorithms & Architectures*, Elsevier Science Publishers B.V. (North-Holland), pp. 295-312, 1986.

[Don89] V. V. Dongen, "Quasi-regular array: definition and design methodology," Proc. of Intern. Conf. on Systolic Arrays, 1989.

[Duf78] M.J.B. Duff, "Review of VLIP image processing system," Proc. Nat. Comput. Conf., pp.1055-1060, 1978.

[FFW85] J.A.B. Fortes, K.S. Fu, and B.W. Wah, "Systematic approaches to the design of algroithmically specifies systolic arrays," Proc. 1985 Int'l Conf. Acoustics, Speech, and Signal Processing, IEEE , pp. 8.9.1-8.9.5, Piscat-

away, NJ, 1985.

[FM84] J.A.B. Fortes and D.I. Moldovan, "Data broadcasting in linearly scheduled array processors," *Proc. 11th Ann. Symp. on Computer Architecture*, pp. 224-231, June, 1984.

[FM85] Jose A. B. Fortes and Dan I. Moldovan, "Parallel detection and algorithm transformation techniques useful for VLSI algorithms," Journal of Parallel and Distributed Computing, Vol.2, pp.277-301, 1985.

[FP84] Jose A. B. Fortes and F. Parisi-Presicce, "Optimal linear shedules for the parallel execution of algorithms," J. Parallel and Distributed Computing, pp.227-301, Aug. 1984.

[GRÜ67] Branko GRÜNBAUM, CONVEX POLYTOPE, Interscience Publishers, London, 1967

[Gol65] M. J. E. Goley, "Apparatus for counting bi-nucleate lymphocytes in blood," US Patent 3,214,574, 1965.

[Hen61] F. C. Hennie, ITERATIVE ARRAYS AND LOGICAL CIRCUITS, The M.I.T. Press and John Wiley & Sons, Inc., 1961.

[Hen68] F. C. Hennie, FINITE STATE MODELS FOR LOGICAL MACHINES, John Wiley & Sons, Inc., New York, 1968.

[HJ85] R. A. Horn and C. R. Johnson, MATRIX ANALYSIS, Cambridge University Press, Cambridge, 1985

[HJ90] J. N. Hwang and J. M. Jong, "Systolic architecture for 2-D rank order filtering," pp.90-99, 1990 International Conference On Application Specific Array Processors", Edited by Sun-Yuan Kung; Earl E. Swartzlander,Jr; Jose A.B. Fortes and K. Wojtek Przytula. IEEE Computer Society Press.

[Hoo87] F. de Hoog, "A new algorithm for solving Toeplitz systems of equations," Lin. Algebra Appl., pp. 122-138, 1987.

[JEH91] J. A. K. S. Jayasinghe, F. M. El-Hadidy, and O.E. Herrmann, "An array processor design methodology for hard real-time systems," ISCAS, vol.5, pp3062-3065, 1991.

[HR93] A. Halanay and V. Rasvan, APPPICATIONS OF LIAPUNOV METHODS IN STABILITY, Kluwer Academic Publishers, Dordrent, 1993.

[HV930] C. Herley and M. Vetterli, "Wavelets and recursive filter banks," IEEE Trans. on Signal Processing, Vol.41, No.8, pp.2536-2556, Aug. 1993.

[KA89] S. Kiaei and L. Aihua, "Analysis of Multirate/Bounded Broadcast," Technical Report, Elec. & Comp. Engr., Oregon State Univ., 1989.

[Kat90] Y. Kato, "Application-oriented high speed processors: experiences and perspectives," 1990 International Conference On Application Specific Array Processors", Edited by Sun-Yuan Kung; Earl E. Swartzlander,Jr; Jose A.B. Fortes and K. Wojtek Przytula. IEEE Computer Society Press.

[KDLS87] D. J. Kuck, E. S. Davison, D.H. Lawrie and A.H. Sameh, "Parallel supercomputing today and the cedar approach," pp.1-23, Experimental Parallel Computing Architectures, Edited by J.J Dongarra, 1987, Elsevier Science Publishing Co., North Holland.

[KH83] Sun-Yuan Kung and Yu Hen Hu, "A highly concurrent algorithm and pipelined architecture for solving Toeplitz systems," IEEE Trans. on ASSP, Vol. 31, No. 1, pp.66-76, Feb. 1983.

[KL78] H. T. Kung and C. E. Leiserson, "Systolic arrays for VLSI," *Sparse Matrix Proceedings*, SIAM, pp.245-282, 1978.

[KL84] H.T. Kung and M.S. Lam, "Wafer-scale integration and two level pipeline implementations of systolic arrays," *J. Parallel Distributed Comput.*, vol.1, pp.32-63, 1984.

[KMW67] R. M. Karp, R. E. Miller, and S. Winogard, "The organization of computations for uniform recurrence equations," Journal of ACM, vol.14, pp. 563-590, July 1967.

[Kun80] H.T. Kung, "The structure of parallel algorithms," in *Advances in Computers*, Vol. 19. New York: Academic, 1980.

[Kun82] H.T. Kung, "Why systolic architectures," *IEEE Computer*, pp. 37-45, Jan. 1982.

[Kun84] S. Y. Kung, "On supercomputing with systolic/wavefront array processors," Proc. IEEE, vol. 72, pp. 867-884, July, 1984.

[Kun85] H. T. Kung, "Systolic alagorithms for the CMU warp processor," in *Systolic Signal Processing*, E.Swartzlander, Ed. pp. 73-96, New York: marcel Dekker, 1987.

[Kun88] S.Y. Kung, VLSI Array Processors, Prentice Hall, Englewood Cliffs, 1988.

[Lev84] H. Lev-Ari, Nonstationary lattice filter modeling, Ph.D Dissertation, Information Systems Laboratory, Dept. of Elect. Engr., Stanford University, Stanford, CA, Sept., 1984.

[LK88] P. Lee and Z. M. Kedem, "Synthesizing linear array algorithms from nested for loop algorithms," *IEEE Trans. on Computers*, 37(12), pp.1578-1598, Dec. 1988.

[LK90] A. Li and S. Kiaei, "VLSI design of multi-rate arrays for DSP algorithms," *1990 Int'l Conf. on Acoustics, Speech and Signal Processing*, New Mexico.

[LK93] P. Lenders and S. Kiaei, "Synthesis of multi-dimensional ARE's", Submitted to IEEE Trans. on Parallel and Dist. Computing, Feb. 1993.

[LRS83] C.E. Leiserson, F.M. Rose, and J.B. Saxe, "Optimizing synchrounous circuitry by retiming," in Third Caltec Conference on VLSI, R. Bryant Ed. pp. 87-116, Rockville, MD: Comnputer Science Press, 1983.

[LW84] G. J. Li and B. W. Wah, "The design of optimal systolic arrays," IEEE Trans. on Computers, Vol. C-33, no. 10, Oct. 1984.

[LW85] G. J. Li and B. W. Wah, "The design of optimal systolic algorithms," IEEE Trans. on Computers, Vol. C-34, no. 1, Jan. 1985.

[Mal89] S. E. Mallat, "A theory for multiresolution signal decomposition: the wavelet representation," IEEE Trans. on Pattern Analysis and Machine intelligence, Vol.11, No.7, pp.674-693, July, 1989.

[McC58] E.J. McCluskey, "Iterative combinational switching networks-general design considerations," IRE Trans. on Electronic Computers, Vol. EC-7, pp.285-291, 1958.

[MF86] D. I. Moldovan and J. A. B. Fortes, "Partitioning and mapping algorithms into fixed-size systolic arrays," IEEE Trans. Computers, Vol. C-35, No. 1, pp. 1-12, Jan. 1986.

[Mol82] D.I. Moldovan, "On the analysis and synthesis of VLSI algorithms," IEEE Trans. Comput., C-31:1121-1126, November 1982.

[Mol93] Moldovan, Dan I., PARALLEL PROCESSING: From Applications to Systems, Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1993.

[MQRS90] C. Mauras, P. Quinton, S. Rajopadhye, and Y. Saouter, "Scheduling affine parameterized recurrences by means of variable dependent timing functions," Int. Conf. on Application Specific Array Processors, IEEE Computer Society, pp.100-110, 1990.

[OF86] M.T. O'Keefe and J.A.B. Fortes, "A comparative study of two systematic design methodologies for systolic arrays," in M. Cosnard, P. Quinton, Y. Robert and M. Tchuente, Ed. *Parallel Algorithms & Architectures*, Elsevier

Science Publishers B.V. (North-Holland), pp. 313-324, 1986.

[Opp78] A. V. Oppenheim, Ed., Applications of Digital Signal Processing, Englewood cliffs, NJ: Prentice-Hall, 1978.

[Orf88] S. J. Orfanidis, OPTIMUM SIGNAL PROCESSING: An Introduction (2nd edition), McGraw Publishing Company, pp.239-246, 1988.

[PD84] K. Preston and M.J.B. Duff, MODERN CELLULAR AUTOMATA, Plemum Press, New york, 1984.

[PRLN92] J.G. Proakis, C.M. Rader, F. Ling and C. L. Nikias, ADVANCED DIGITAL SIGNAL PROCESSING, McMillan Publishing Company, New York, 1992.

[Qui83] Patrice Quinton, "The systematic design of systolic arrays," Tech. Rep. 216, Institute National de Recherche en Informatique et en Automatique INRIA, July, 1983.

[Qui84] P. Quinton, "Automatic synthesis of systolic arrays from uniform recurrence equations," Proc., 11th Ann. Symp. on Computer Architecture, pp.208-214, 1984.

[Raj86] S.V. Rajopadhye, "Synthesis, optimization and verification of systolic architectures," Ph.D. thesis, University of Utah, Salt Lake City, Utah 84112, Dec. 1986.

[RF86] S.V. Rajopadhye and R.M. Fujimoto, "Systolic array synthesis by static analysis of program dependencies," in Parallel Architecture and Languages Europe (Eds J.W. deBakker, A.J. Nijman and P.C. Treleaven), Springer Verlag, pp.295-310, 1987.

[Raj89] S. V. Rajopadhye, "Synthesizing systolic arrays with control signals from recurrence equations," Distributed Computing, pp.88-105, May 1989.

[Rao85] Sailesh Rao, "Regular iterative algorithms and their implementations on processor arrays," Ph.D. dissertation, Standard University, Information System Lab., Standard, CA, Oct. 1985.

[RCM92] J. Rosseel, F. Catthoor, and H. D. Man, "The exploitation of global operations in affine space-time mapping," VLSI Signal Processing, V. pp. 309-318, 1992

[RF88] S. V. Rajopadhye and R. M. Fujimoto, "Synthesizing systolic arrays from recurrence equations," Parallel Computing, 1988.

[Ros88] B.A. Rosenfeld, A HISTORY OF NON-EUCLIDEAN GEOMETRY, Evolution of the Concept of a Geometric Space, Springer-Verlag, New York, 1988.

[RTRK88] P. Roychowdhury, L. Thiele, S.K. Rao, and T. Kailath, "On the localization of algorithms for VLSI Processor Arrays," in R.W. Brodersen and H.S. Moscovitz (Eds) VLSI Signal Processing III, IEEE, pp.459-470, New York, 1988.

[SBM62] D. L. Slotnick, W.C. Borck, and R.C. Mcreynolds, "The SOLOMON computer," Proc. of the AFIPS Fall Joint Computer Conference, pp. 97-107, Washington D. C., 1962.

[Sch86] A. Schrijver, THEORY OF LINEAR AND INTEGER PROGRAMMING, John Wiley & Sons Ltd.Tiptree, Essex, 1986.

[SC92] Chris J. Scheiman and P. Cappello, "A processor-time minimal systolic array for transitive closure," IEEE Trans. on Parallel and Distributed Systems, Vol.3, No.3, pp. 257-269, May 1992.

[SC93] C. J. Scheiman and P. Cappello, "A period-processor-time-minimal schedule for cubical mesh algorithms," In Proc. Int. Conf. on Application Specific Array Processors, pp. 261-272, IEEE Computer Society, Oct. 1993.

[SF88] W. Shang and J.A.B. Fortes, "Time optimal linear shcedules for algorithms with uniform dependencies," in *Int'l Conf. on Systolic Arrays*, pp.393-402. San Diego, May 1988.

[SF89] W. Shang and J.A.B. Fortes, "On the optimality of linear shcedules," *J. of VLSI Signal Processing*, Vol.1, 1989, pp.209-220, Kluwer Academic Publishers, Boston.

[Sha73] R. S. Shankland, "Conversations with Albert Einstein. II," American Journal of Physics, Vol. 41, pp.895-901. 1973.

[Sha82] R. Shaw, LINEAR ALGEBRA AND GROUP REPRESENTATIONS, Vol.1, "Linear Algebra and Inroduction to Group Representations," Academic Press, London, 1982.

[Sha83] R. Shaw, LINEAR ALGEBRA AND GROUP REPRESENTATIONS, Vol.2, "Multilinear Algebra and Group Representations," Academic Press, London, 1983.

[Tau79] gerald E. Tauber, Editor, ALBERT EINSTEIN'S THEORY OF GENERAL RELATIVITY, Crown Publishers, Inc. New York, 1979.

[Thi89] L. Thiele, "On the design of piecewise regular processor arrays," IEEE symp. on Circuits and Systems, Portland, pp.2239-2242, 1989.

[Ung58] H. Unger, "A computer oriented toward special problems,", Proc. IRE, 46, pp.1744-1750, 1958.

[Vai90] P. P. Vaidyanathan, "Multirate digital filters, filter banks, polyphase networks, and applications: A tutorial", Proc. of The IEEE, Vol.78, No.1, pp.56-93, 1990.

[Wai67] W.M. Waite, "Path detectionin multidimensional iterative arrays," J. ACM, 14(2), pp.300-310, Apr. 1967.

[WD85] Yiman Wong and J. M. Delosme, "Optimal systolic implementation of N-dimensional recurrences," IEEE Proc. ICCD, pp. 618-621, 1985.

[WD89] Yiman Wong and J. M. Delosme, "Optimization of processor count for systolic arrays," Dept. of Computer Sci. RR-697, Yale Univ., May 1989.

[WD92] Yiman Wong and J. M. Delosme, "Optimization of computation time for systolic arrays," IEEE Trans. on Computers, Vol. 41, no.2, pp. 159-177, Feb. 1992.

[YC88] Yoav Yaacoby and Peter R. Cappello, "Scheduling a system of affine recurrence equations onto a systolic array," Dept. of Electrical & Computer Engineering, and Dept. of Computer Science, U. of California, Santa Barbara, 1988.

[YC88a] 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.

[YC88b] Y. Yaacoby and P.R. Cappello, "Bounded Broadcast in Systolic Arrays," Technical Report, Dept. of Computer Science, UCSB, Santa Barbara, April 1988.

[ZK93] Y.P. Zheng and S. Kiaei, "Multi-rate transformation of directional affine recurrence equations," The Inter. Conf. on Application Specific Array Processors, Venice, Italy, pp.392-403, 1993.

[ZK93] Y.P. Zheng and S. Kiaei, "Multi-rate transformation of directional affine recurrence equations," Application-Specific-Array-Processors, IEEE Computer Society, pp.392-403, Oct. 1993.

[ZK93] Y. Zheng and S. Kiaei, "Automatic derivation of multi-rate VLSI arrays for directional affine recurrence equations," IEEE Trans. on Signal Processing,

(under review).

[ZK94] Y. Zheng and S. Kiaei, "Overlapping transformation for time-area optimal VLSI arrays for Toeplitz system," J. of Computer and Software Engr., 1994.

[ZR92] X. Zhong and S. V. Rajopadhye, "Quasi-linear allocation functions for efficient array design," J. of VLSI Signal Processing, 4(97-110), 1992.

The following papers are all cited from "1990 International Conference On Application Specific Array Processors", Edited by Sun-Yuan Kung; Earl E. Swartzlander,Jr; Jose A.B. Fortes and K. Wojtek Przytula. IEEE Computer Society Press.

[Kat90] Y. Kato, "Application-oriented high speed processors: experiences and perspectives," 1990 International Conference On Application Specific Array Processors", Edited by Sun-Yuan Kung; Earl E. Swartzlander,Jr; Jose A.B. Fortes and K. Wojtek Przytula. IEEE Computer Society Press.

- Application-oriented high speed processor development should last forever by featuring one order higher speed processing capabilities than that of conventional microprocessors. special architecture tuned for a specific application has advantages over conventional microprocessors.
- The future application trend is not a issue here, because every personal workstation will enjoy today's super computer processing capability in the near future by employing application-oriented processors in its peripheral. Such workstations will be able to perform more complex and higher speed applications. Real time foreign language interpreters and three dimensional video/image semantic recognitions are possible examples. Humanoid robots which can communicate with us will also become a reality.
- The key issue of the application-oriented high speed processor is the harmony among cost, size and processing capability for specific applications which creates harmony between human beings and machines.

[Cla90] Ph. Clauss, C. Mongenet, and G.R. Perrin, "Calculus of space-optimal mapping of systolic algorithms onto processor arrays," 1990 International Conference On Application Specific Array Processors," Edited by Sun-Yuan Kung; Earl E. Swartzlander,Jr; Jose A.B. Fortes and K. Wojtek Przytula. IEEE Computer Society Press.

- A method based on geometrical interpretations on the convex polyhedra in $Z^n$ for the mapping of systolic algorithms onto minimum number of processors is presented. Two space-optimal mappings of the Gaussian elimination algorithm are derived, one is called pipelining, and the other is grouping mapping.

- There is nothing new but they try to use different geometrical tools for the expressions of algorithms. I. polyhedron expression, expressed by the edges and vertices of the polyhedron of the algorithms. II. timing and allocation expression, decomposing the polyhedron into temporal planes and allocation directions; III. potential parallelism expression, considering the maximum parallelism of the algorithm and then determine the space-optimal mapping relative to a given timing.
- Overall, there is nothing new but try to use some geometrical terminology for the expression of algorithms and systolic arrays.

[Sch90] C.J. Scheiman and P. Cappello, "A processor-time minimal systolic array for transitive closure," Edited by Sun-Yuan Kung; Earl E. Swartzlander,Jr; Jose A.B. Fortes and K. Wojtek Przytula. IEEE Computer Society Press.

- The computation of transitive closure of a relation over a set of $n$ elements on $n^2$ PEs requires $5n$-4 steps as shown in the Kung, Lo, and Lewis (KLL) algorithm. This paper shows that to achieve this 5n-4 time bound, only $\lceil n^2/3 \rceil$ PEs are needed. A processor-time minimal systolic array using $\lceil n^2/3 \rceil$ PEs organized as a cylindrical 2D mesh when n is multiple of 3, or connected as a 2D mesh as twisted torus is presented.
- The dag of the KLL algorithm is defined as follows:
- $G_{tc(n)} = (N, A)$

$$A = \{ [(i,j,k), (i^1,j^1,k)] \,|\, i^1 = i+1. \oplus j^1 = j+1, 1 \le i,j,k < n \}$$

- $$\cup [(i,j,k), (i1,j1,k1)] \,|\, i1 = i-1, j1 = j-1, k1 = k+1, 1 < i,j \le n$$
$$1 \le k < n$$

[Hwa90] J.N. Hwang and J.M. Jong, "Systolic architecture for 2-D rank order filtering," pp.90-99, 1990 International Conference On Application Specific Array Processors", Edited by Sun-Yuan Kung; Earl E. Swartzlander,Jr; Jose A.B. Fortes and K. Wojtek Przytula. IEEE Computer Society Press.

- 2-D rank order filtering has found applications in a wide variety of applications in image processing. This design derives its architecture mainly from a systolic design for 1-D rank order filtering. The adopted design, called sample oriented rank order filter design, takes advantage of the evaluated ranks values in the current window for the evaluation of the rank values in the next window without explicitly sorting the data in the window. This fact allows the conversion of a 2-D windowed data sequence into a 1-D windowed data sequence with multiple data samples exchange at each window movement, and the 1-D systolic design can be used.By cascading many such 1-D sys-

tolic arrays into a 2-D array, and supplied with simple parallel-in serial-out logic, this architecture is able to achieve the maximally available parallelism with nearly 100% efficiency if either the pipeline interleaving or processor sharing technique is used. This design does not require the preloading of the whole 2-D data array, line scanned images are allowed to be processed with negligible time delay.

[Mau90] C. Mauras, P. Quinton, S. Rajopadhye, and Y. Saouter, "Scheduling affine parameterized recurrences by means of variable dependent timing functions," 1990 International Conference On Application Specific Array Processors", Edited by Sun-Yuan Kung; Earl E. Swartzlander,Jr; Jose A.B. Fortes and K. Wojtek Przytula. IEEE Computer Society Press.

- A new technique on affine scheduling of each variable of a system of ARE independent of the others by a affine timing function. This new technique makes it possible to analyze systems of RE with variables in different index spaces, and multi-step systolic algorithms.
- Previous synthesis techniques make implicitly the assumption that all variables are in the same index space. Thus, a preliminary heuristic transformation is necessary to ensure that this condition is met.
- This method is very similar to what I proposed but it is still a step behind, they implicitly assume that all index spaces are under the same coordinate system which makes its application to only unimodular dependence maps. But its writing will be a good example for me to follow, especially section 2. The definition for MCS specification of ARE is given in section 1.

[Jon93] Don H. Johnson and Dan E. Dudgeon, ARRAY SIGNAL PROCESSING--Concepts and Techniques, Prentice Hall, Englewood Cliffs, NJ, 1993.

- Some good points on Eigenanalysis. Eigenvectors are used to define the coordinate system tailored to the linear operator. This is very similar to what I am working on the derivation of MCS from the dependence matrices. Some terminologies may help me in presenting MCS. But in this book, only full rank matrices are considered, i.e., the linear operator has no null space. In DSP algorithms, dependence matrices are often rank deficit and therefore it is important to cope with rank deficit matrices.
-

[GRÜ67] Branko GRÜNBAUM, CONVEX POLYTOPE, Interscience Publishers, London, 1967

- This book talks about polytopes and convex hulls, hyperplanes, and all those n-dimensional geometries needed for the development of synthesis theory of VLSI array processors. The following is some notes taken from the book for the definitions of some terminologies: $(x, y)$ denotes the scalar product of vector $x$ and $y$.

  1. A *hyperplane* $H$ is a set which may be defined as $H = \{x \in R^d \mid (x, y) = \alpha\}$ for suitable $y \in R^d, y \neq 0$.

  2. An *open half-space* [*closed half-space*] is defined as $H = \{x \in R^d \mid (x, y) > \alpha\}$ [respectively $H = \{x \in R^d \mid (x, y) \geq \alpha\}$ ] for suitable $y \in R^d, y \neq 0$, and $\alpha$.

  3. A set $K \in R^d$ is a *convex* if and only if for each pair of distinct points $a, b \in R^d$ the closed segment with endpoints $a$ and $b$ is contained in $K$. equivalently, $K$ is convex if its intersection with every straight line is either empty, or a connected set.

  $K$ is a convex iff $a, b \in K$ and $0 \leq \lambda \leq 1$ imply $\lambda a + (1 - \lambda) b \in K$.

  Properties of convex:

  a). if $\{K_\nu\}$ is any family of convex sets in $R^d$, then their intersection $\bigcap_\nu K_\nu$ is also convex.

  b). if $A, B$ are convex, then $A + B$ and $A - B$ are convex, and for any real $\lambda$, the set $\lambda A$ is convex.

  c). If A is convex, $a_i \in A$ and $\lambda_i \geq 0$ for $i = 1, 2, \ldots, k$, and $\sum_{i=1}^{k} \lambda_i = 1$, then $\sum_{i=1}^{k} \lambda_i a_i \in A$.

  d). if $A \subset R^d$ is convex, the sets cl A and int A are also convex.

  e). if $A \subset R^d$ is convex, $x \in A$ $y \in \text{int}A$, then all points of the line segment between $x$ and $y$ belong to int $A$.

  f). If $T$ is affine transformation of $R^d$into itself, and if $A \subset R^d$ is convex, then $T(A)$ is convex.

  g). For a convex set $A \subset R^d$ let $H$=aff $A$ be the affine hull of $A$. The relative interior relint $A$ of $A$ as a subset of $H$ is never empty, and relint cl $A \subset A \subset$ cl relint $A$. The relative boundary relbd $A$ of $A$ with respect to $H$ is empty iff $A$ is an affine variety (i.e. $A$=$H$).

  4. *Convex hull*: A convex hull conv A of a subset A of $R^d$ is the intersection of all the convex sets in $R^d$ which contain A.

[Ban90] Thomas F. Banchoff, BEYOND THE THIRD DIMENSION, Geometry, Computer Graphics, and Higher Dimensions. Scientific American Library, New York, 1990. (QA691.B26 1990)

- The axioms of Euclidean Plane Geometry:

1. A straight line may be drawn between any two points.
2. Any terminated straight line may be extended indefinitely.
3. A circle may be drawn with any given point as center and any given radius.
4. All right angles are equal.
5. If two straight lines in a plane are met by another line, and if the sum of the internal angles on one side is less than two right angles, then the straight line will meet if extended sufficiently on the side on which the sum of the angles is less than two right angles.

5'. (5 can be stated as follows): For any given point not on a given line, there is exactly one line through the point that does not meet the given line.

- Coordinate geometry.

[Sch86] Alexander Schrijver, THEORY OF LINEAR AND INTEGER PROGRAMMING, John Wiley & Sons Ltd.Tiptree, Essex, 1986.

The emphasis of this book is on the theoretical aspects of linear and integer programming, and it aims at complementing the more practically oriented books. This book contains many basic definitions and theorems in linear algebra, matrix theory and Euclidean geometry.

- Pivoting: If $A$ is a matrix, say $A = \begin{bmatrix} a & b \\ c & D \end{bmatrix}$ where $a$ is a nonzero number, $b$ is a row vector, $c$ is a column vector, and $D$ is a matrix, then pivoting over the *pivot element* (1,1) means replacing $A$ by the matrix $\begin{bmatrix} -a^{-1} & a^{-1}b \\ a^{-1}c & D - a^{-1}cb \end{bmatrix}$. Pivoting over any other element of $A$ is defined similarly.

- The sizes of a rational number $\alpha = p/q$ (where p and q are relatively prime integers), of a rational vector $c = (\gamma_1, \ldots, \gamma_n)$ and of a rational matrix $A = (\alpha_{ij})_{i=1,j=1}^{m\ \ n}$ are:

$$\begin{aligned} size(\alpha) &= 1 + \lceil log_2(|p| + 1) \rceil + \lceil log_2(|q| + 1) \rceil \\ size(c) &= n + size(\gamma_1) + \ldots + size(\gamma_n) \\ size(A) &= mn + \sum_{i,j} size(\alpha_{ij}) \end{aligned} \tag{4.1}$$

- Polynomial algorithms:

If $f, g_1, \ldots, g_m$ are real-valued functions, then $f$ is said to be *polynomially bounded* by $g_1, \ldots, g_m$ if there is a function $\phi$ such that $\phi \geq f$ and such that $\phi$ arises by a sequence of compositions from the functions $g_1, \ldots, g_m$ and from some polynomials.

In the special case that $g_1, \ldots, g_m$ are polynomials, it follows that if $f$ is polynomially bounded by $g_1, \ldots, g_m$, then f is bounded above by a polynomial. In that case, $f$ is called a *polynomially bounded* function.

An algorithm is called *polynomial-time* or *polynomial* if its running time function is *polynomially bounded.* A problem is said to be *solvable in polynomial time* or *polynomially solvable* if the problem can be solved by a polynomial-time algorithm.

- The Classes of *P, NP* and co-*NP*

The class of decision problems solvable in polynomial time is denoted p. Another, possibly larger, "complexity class" is the class *NP* [solvable by a *N*on-deterministic Turing machine in *P*olynomial-time].

- A set $\Lambda$ of $R^n$ is called an *(additive) group* if:

$$\begin{aligned} &(i)\ 0 \in \Lambda \\ &(ii)\ \textit{if}\quad x, y \in \Lambda, \textit{then}\quad x+y \in \Lambda \quad \textit{and} -x \in \Lambda \end{aligned} \tag{4.2}$$

The group is said to be generated by $a_1, \ldots, a_m$ if

$$\Lambda = \{\lambda_1 a_1 + \ldots + \lambda_m a_m | \lambda_1, \ldots, \lambda_m \in Z\} \tag{4.3}$$

The group is called a *lattice* if it can be generated by linearly independent vectors. These vectors then are called a *basis* of the lattice.

[Mol93] Moldovan, Dan I., PARALLEL PROCESSING: From Applications to Systems, Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1993.

- SIMD Supercomputers:(pp.204-217)

(1) The Connection Machine: CM-2. 64K PEs, organized in four 16K partitions. Each partition is controlled by a sequencer. The four sequencers are connected to as many as four front-end computer systems through a $4 \times 4$ cross-point switch.

(2) The Hughes 3-D Computer: By Hughes Research Laboratory. It is the first system built with 3-D wafer-scale integration technology. The functional blocks of the individual array processing elements, such as the ALU, registers, comparators, memory, and other logic, are distributed across a number of vertically aligned wafers. Each wafer in the stack contains an entire $128 \times 128$ array constituting one particular functional block. The stack is about 1 inch high and 4 inches in diameter.

- Systolic arrays: The interconnection pattern should be simple and regular, with only local connections of PEs and without long wires that would need more area or more energy to drive them. Thus, VLSI structures are characterized by a high degree of mod-

ularity, absence of long data paths, localized connectivity for data transfer, limited capabilities of processing elements, absence of central control, and simple timing mechanisms.

The general idea in applying systolic processing to a problem is to transform the problem to meet the VLSI circuit requirements, rather than the other way around. This shifts much of the design complexity from the chip layout designer to the algorithm designer. This approach also minimizes overall design efforts, because the layout design is much more time- and labor-intensive than are the algorithm manipulations needed to "systolicize" algorithms.

The challenge is to design algorithms that are computation and I/O balanced.

Warp and *i*Warp: Warp machine has 3 components: Warp processor array, interface unit, and the host. The host executes parts of the application program that are not mapped onto systolic array, such as initialization subroutines and sequential code. the host supplies the data to, receives the results fro, the array. (Notice that this is the same concept I have for the future expert-computer-systems. Computers with different expertise are integrated together to form a supercomputer system.)

[Cox91] H. S. M. Coxeter, REGULAR COMPLEX POLYTOPES, Second edition, Cambridge University Press, 1991. (QA691.C66 1991).

[Bir91] M.K. Birbas, D. J. Soudris,and C.E. Goutis,"Design methodology for direct mapping of iterative algorithms on array architectures," Intern. Symposium on Circuits and Systems, vol.5, pp.3058-3061, 1991.

- The concept of the array architecture is combined efficiently with the existing knowledge of compiler techniques. It is considered as initial description of the iterative algorithm Fortran-like nested loops, without the requirement of transforming it into any intermediate form such as URE. The principles of Lamport's Coordinate Method are employed. Depending on the systolic or non-systolic structure of the algorithm and/or the multidimensional mapping, the resulting architectures can be either Regular Arrays or Piecewise Regular Arrays. The important subclass of algorithms known as Weak Single Assignment Codes is treated in an one time a way. Finally, it should be stressed that multiprojection can be performed easily and the hardware specification of the processing element of the array processor are described in detail.

- The clock ratios of an MRA array are determined by ratios of the scale sizes of the axes of different coordinate systems along the direction of the projection vector. Let the projection vector be $\lambda_a$. Let variable $u$ depends on variable $v$, and their dependence vector be $T$, then the clock rate ratio r can be determined as follows:

$$r = \frac{\Phi_v}{\Phi_u} = \frac{\lambda_a T_{vu}}{\lambda_a T_{uu}} \tag{4.4}$$

where $\lambda_a T_{uu}$ should not equal to zero, otherwise, the variable U is a stored variable instead of propagated variable. We will consider the propagation clock rate of a stored variable as zero.