Chapter 1

Sytematic Synthesis of Processor Arrays from Uniform recurrence equations

Kumar N. Ganapathy
Benjamin W. Wah

1.1 Introduction

The rapid growth of technology in the last decade has had a great impact on the evolution of computer architectures. High performance specialized computer systems, called Systolic arrays, are used to meet application-specific requirements. Systolic arrays have been recognized as excellent candidates for high performance parallel computing [6]. The fundamental concept behind a systolic architecture is that the *Von-Neumann* bottleneck is greatly alleviated by repeated use of fetched data item in a physically distributed array of processing elements. The flow of data through the array at clock beat is rhythmic and regular, like the pumping of a heart and hence the name systolic. The regularity of the arrays lead to cheap and dense VLSI implementations, which implies high-performance and low overhead cost. Systolic arrays fit naturally into the concept of a hardware library [3], where functional units are in relation to the host computer as subroutines from a software library are to a production code.

Initial designs of systolic arrays were ad hoc and relied heavily on designer's skill and intuition. Since every algorithm needs a specialized systolic design customized to its communication patterns, a systematic technique for generating systolic arrays from

[1]Research Supported by National Science Foundation, grant MIP 88-10584 and Joint Services Electronics Program contract N00014-90-J-1270.

the algorithm description is necessary. Hence, a great deal of effort has been devoted by numerous researchers to generate the systolic arrays in a systematic fashion. An overview of the different methods can be found in [1].

1.1.1 Algorithm representation : Recurrences

In mapping algorithms to processor arrays, the representation of the algorithm is often in the form of a recurrence equation. Recurrences have long been used to express a large body of computations [5][12]. Informally, a recurrence equation depicts the dependencies between points in the domain over which it is evaluated. Mathematically, a recurrence equation over a N-dimensional domain D is defined as an equation of the form,

$$\mathcal{Z}(p) \quad = \quad \phi\,[\mathcal{Z}(q_1), \mathcal{Z}(q_2), \ldots, \mathcal{Z}(q_m)] + \psi(p) \tag{1.1}$$

where $p, q_i \in D$, for $i = 1, \ldots, m$, ϕ is a single valued function strictly dependent on each of its arguments, ψ represents the input. Based on the nature of the dependencies, recurrences can be classified as Uniform, Linear or Non-linear.

A recurrence equation is called a uniform recurrence equation if $q_i = p + d_i$, for $i = 1 \ldots m$, where d_i's are constant N-dimensional vectors independent of p or q. Matrix multiplication of two matrices, A and B, is an example of uniform recurrence and is represented as,

$$C(i,j,k) = C(i,j,k-1) + A(i,k).B(k,j) \tag{1.2}$$

Here, $p = (i,j,k)$, $q_1 = (i,j,k-1)$, $d_1 = (0,0,-1)$.

A recurrence equation is said to be linear if $q_i = A_i p + b_i$, where A_i is a constant $N \times N$ matrix and b_i is a constant N-dimensional vector. Finding the transitive closure of a matrix C is example of a linear recurrence equation and is stated as,

$$C(i,j,k) = C(i,j,k-1) + C(i,k,k-1) \times C(k,j,k-1) \tag{1.3}$$

where $+$ is to boolean OR operation ; $\times$ is boolean AND operation. In the above recurrence,

$$q_1 \quad = \quad \begin{bmatrix} i & j & k-1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \\ k \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} \tag{1.4}$$

$$q_2 \quad = \quad \begin{bmatrix} i & k & k-1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \\ k \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} \tag{1.5}$$

$$q_3 \quad = \quad \begin{bmatrix} k & j & k-1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \\ k \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} \tag{1.6}$$

A recurrence equation is said to be non-linear if $q_i = \chi(p)$ for some non-linear function χ. The formulation of a knapsack problem [4] is an example of a non-linear recurrence equation.

$$
\begin{aligned}
f(0,0) &= 0 && (1.7)\\
f(k,w) &= \min_{j \in \{j_1\}} \{f(j, w - a_k)\} && (1.8)
\end{aligned}
$$

where $j_1 = (j : j < k$ and $(j, w - a_k)$ corresponds to an equivalence class)

1.1.2 Systematic methods

Most of the methods for synthesizing systolic structures consider only a system of uniform recurrence equations. The work done by Moldovan and Fortes [11][10] laid the foundation for a number systematic methods of array synthesis from uniform recurrences.

Dependency method

In the method due to Moldovan and Fortes, called in this chapter as dependency method, the algorithm (A) is represented as a 5-tuple (J^n, C, D, X, Y). J^n is a finite n-dimensional index set of A, C is the set of triples which represent the set of computations performed, D is the set of dependencies, X is the set of input variables and Y is the set of output variables. A feasible design is obtained by a linear transformation, represented as a $n \times n$ matrix T $(\in Z^{n \times n})$, of the index space. Thus,

$$T = [\pi S]^t \tag{1.9}$$

where π is $1 \times n$ schedule vector and S is the processor allocation matrix. For any index point $\hat{j}$, $S\hat{j}$ denotes the processor at which the index point executes and $\pi\hat{j}$ is the time of execution at that processor. Constraints are imposed on matrix T to ensure valid execution of the algorithm.

The design of a systolic array is then equivalent to determining the N^2 parameters of the transformation matrix, T. This is an integer programming problem in N^2 variables and solution methods are of exponential complexity. Hence, the allocation matrix S is chosen and the schedule π which minimizes the total execution time for that choice of S is found. The resulting designs are therefore "optimal" up to the choice of S.

In case of uniform recurrences, the dependencies are homogeneous, in the sense that the collection of dependence vectors emanating from each point are translates of one another. This means that the dependence status at one point is the same as any other in the domain (denoted by D). Therefore, once a transformation matrix T is chosen, the time between execution of points $\hat{j_1}$ and $\hat{j_2}$ where $\hat{j_1} = \hat{j_2} + d$ is equal to $\pi.d$ which is a constant independent of $\hat{j_1}$ or $\hat{j_2}$. Thus the computation would be periodic along the dependence direction d, which is a chain of index points $\hat{j_1}, \hat{j_2}, \ldots, \hat{j_k}$, where $\hat{j_{i+1}} = \hat{j_i} + d$.

Similarly, the distance in the processor space between the execution locations of points $\hat{j_1}$ and $\hat{j_2}$ is a constant equal to $S.d$. Therefore, for uniform recurrences, the coefficients of matrix T are chosen to have a periodic execution along any dependence direction. This indicates that the search space for determining T can be restricted to those that yield periodic execution profiles. This intuition is succinctly embodied in the parameter method (described in this chapter) to devise a efficient strategy that systematically yields optimal systolic designs.

In this chapter, a parameter based approach, called the parameter method, to synthesize systolic structures for uniform recurrences is presented. In the parameter method [9] the operations of the target systolic array is represented by a minimal set of parameters. These parameters capture exactly the function of the systolic array which executes the uniform recurrence. This is in contrast to the dependency method which in principle can be applied to non-uniform recurrences also. The number of parameters are often smaller than N^2 (the number of elements in the T matrix). The design objectives like the execution time or the number of processors in the array, can be easily represented in terms of the parameters. The optimal target array can then be found by a systematic enumeration over a polynomial search space.

The organization of the chapter is broadly as follows. The next section describes a special (simpler) case of the parameter method for generating 2-dimensional arrays from 3-dimensional recurrences. Section 1.3 presents a generalization of the simple parameter scheme and discusses the issues involved in searching for the optimal design. Examples are provided in both section 1.2 and 1.3 to illustrate the techniques described. The final section concludes with remarks on future work possible.

1.2 Parameter Method : Special case

In this section, a simpler case of the parameter method, proposed originally by Li and Wah [9] is described to bring out the ideas clearly.

The structure of the uniform recurrence considered, for the computation of a 2-dimensional output $\mathcal{Z}$ from 2-dimensional inputs $\mathcal{X}$ and $\mathcal{Y}$, is

$$\mathcal{Z}^{k}_{i,j} = \phi\left[\mathcal{Z}^{k-\delta}_{i,j}, x_{i,k}, y_{k,j}\right] \tag{1.10}$$

where ϕ is a single valued function (to be executed in each PE). The recurrence is termed forward if $\delta = 1$ and backward if $\delta = -1$. $x_{i,k}$ and $y_{k,j}$ are linear functions (called subscript access functions or indexing functions) which define the indices of 2-dimensional inputs $\mathcal{X}$ and $\mathcal{Y}$ in the k step or iteration of the recurrence. In the following, the coefficients of i, j, k in $x_{i,k}$ and $y_{k,j}$ are assumed to be 1 or -1. The case of the coefficients greater than one is given later. Linear indexing functions such as $x_{i,k}$ can be represented by constant coefficient matrices. For example if i_1, i_2 are the indices of $\mathcal{X}$ then $x_{i,k}$ could be given by the matrix in the following equation.

$$\begin{bmatrix} i_1 \\ i_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} i \\ j \\ k \end{bmatrix} \tag{1.11}$$

Another form of recurrence considered is a 2-dimensional recurrence computing a 1-dimensional output from 1-dimensional inputs $\mathcal{X}$ and a, given as

$$\mathcal{Z}_i^k = \phi\left[y_i^{k-1}, x_{i,k}, a_k\right] \tag{1.12}$$

Most of the description in this section will be in reference to the 3-dimensional recurrence in equation 1.10. The case of 2-dimensional recurrence is almost similar to that of the 3-dimensional case.

The goal of the parameter method is to generate 2-dimensional (resp. 1-D) systolic array to solve problems represented as 3-dimensional (resp. 2-D) recurrences. In order to see the significance of the particular forms of recurrence chosen above, a number of examples of the same general form are presented below.

1.2.1 Examples

1. *FIR Filtering* can be considered as an matrix-vector multiplication of a upper-triangular Toeplitz band matrix of bandwidth m with a vector of size n.

$$\begin{array}{rcll} y_i^0 & = & 0, & 1 \le i \le n \\ y_i^k & = & y_i^k + a_k x_{i+k-1}, & 1 \le i \le n, 1 \le k \le m, \\ & & & x_j = 0 \text{ for } j > n. \end{array} \tag{1.13}$$

2. *Discrete Fourier Transform* (DFT) is a matrix-vector multiplication in which the matrix has $\omega = \exp 2\pi\sqrt{-1}/n$, the n-th root of unity.

$$\begin{array}{rcll} y_i^0 & = & 0, & 0 \le i \le n-1 \\ y_i^k & = & y_i^{k-1}\omega^i + x_{n-k}, & 1 \le k \le n, \ \ 0 \le i \le n-1. \end{array} \tag{1.14}$$

3. *Polynomial Multiplication* of two degree-n polynomials.

$$\begin{array}{rcll} c_i^0 & = & 0, & 0 \le i \le 2n \\ c_i^k & = & c_i^{k-1} + a_{k-1} b_{i-k+1}, & 0 \le i \le 2n, \ \ 1 \le k \le n, \\ & & & b_j = 0 \ \ \mathit{for} \ \ j < 0 \ \ \mathit{or} \ \ j \ge n. \end{array} \tag{1.15}$$

4. *Deconvolution* expressed in recurrence form with a temporary variable z_i

$$\begin{array}{rcll} z_i^0 & = & y_i, & 1 \le i \le n \\ z_i^k & = & z_i^{k-1} - a_{m-k+1} x_{i+m-k}, & 1 \le k \le m-1, \ \ 1 \le i \le n, \\ & & & x_j = 0 \text{ for } j > n \\ x_i & = & \frac{z_i^{m-1}}{a_i}, & 1 \le i \le n. \end{array} \tag{1.16}$$

5. *Two-dimensional Matrix Multiplication* of two $n \times n$ matrices A and B.

$$\begin{array}{rcll} c^0_{i,j} & = & 0, & 1 \leq i,j \leq n \\ c^k_{i,j} & = & c^{k-1}_{i,j} + a_{i,k}b_{k,j}, & 1 \leq i,j,k \leq n \end{array} \tag{1.17}$$

6. *Triangular Matrix Inversion*: Matrix inversion of an upper triangular matrix U to $V = U^{-1}$ by a temporary variable $w_{i,j}$

$$\begin{array}{rcll} w^{j+1}_{i,j} & = & 0, & 1 \leq i < j \leq n \\ w^k_{i,j} & = & w^{k+1}_{i,j} - u_{i,k}v_{k,j}, & 1 \leq i < k \leq j \leq n \\ & & & \\ v_{i,j} & = & \frac{w^{i+1}_{i,j}}{u_{i,i}}, & 1 \leq i < j \leq n \\ v_{i,i} & = & \frac{1}{u_{i,i}}, & 1 \leq i \leq n \end{array} \tag{1.18}$$

7. *Two-dimensional Tuple Comparison:* Given two 2-dimensional matrices A and B, determining if the i-th row of A is identical to the j-th row of B.

$$\begin{array}{rcll} c^0_{i,j} & = & TRUE, & 1 \leq i,j \leq n \\ c^k_{i,j} & = & c^{k-1}_{i,j} \wedge (a_{i,k} = b_{j,k}), & 1 \leq i,j,k \leq n. \end{array} \tag{1.19}$$

The first 4 in the above list are 2-dimensional recurrences while others are 3-dimensional.

1.2.2 Pipelining

Consider the 3-dimensional recurrence in equation 1.10. The subscript access functions of $\mathcal{X}$ and $\mathcal{Y}$ involve only two of the three index variables i, j, k. Therefore, the same input data token $\mathcal{X}(x_{i,k})$ is used at all the index points (i, j, k), for all j. If the number of values j can take in the domain of the recurrence is n_j, the input data token must be circulated to all the n_j computation points. We assume that broadcasting of a input token is not allowed and that no input token is duplicated and sent into the array. Therefore, the input data token has to be pipelined through the n_j computation points which use the data. For our case of uniform recurrence, the input $\mathcal{X}(x_{i,k})$ flows along the increasing or decreasing j index. For example, if we assume increasing j then,

$$\mathcal{X}(x_{i,j,k}) \quad = \quad \mathcal{X}(x_{i,j-1,k}) \tag{1.20}$$

Similarly, the other input $\mathcal{Y}$ is pipelined to flow along the increasing or decreasing i index as its indexing function does not involve i.

Thus for the recurrence in equation 1.10 we introduce two extra dependencies $(0, +1, 0)$ for variable $\mathcal{X}$ and $(+1, 0, 0)$ for $\mathcal{Y}$ and the recurrence becomes,

$$\mathcal{X}(x_{i,j,k}) \quad = \quad \mathcal{X}(x_{i,j-1,k}) \tag{1.21}$$

$$\mathcal{Y}(y_{i,j,k}) \quad = \quad \mathcal{Y}(y_{i-1,jk}) \tag{1.22}$$

$$\mathcal{Z}^k_{i,j} \quad = \quad \phi\left[\mathcal{Z}^{k-1}_{i,j}, \mathcal{X}(x_{i,j,k}), \mathcal{Y}(y_{i,j,k})\right] \tag{1.23}$$

1.2.3 Parameters

The crux of the parameter method is the characterization of the behavior, correctness and performance of the systolic array by a set of parameters. To understand the parameter method we must see the rationale behind the choice of the parameters. Earlier we noted that when a uniform recurrence is executed on a systolic array the computation is periodic and equally-spaced along any dependence direction. Thus, we can associate a period with the computation along every dependence direction. In addition, since the computation is equally-spaced, a data token moves the same distance between consecutive computations. Hence the speed of the data remains constant along the dependence direction and is chosen as a parameter. Once the velocity is fixed the relative distance between two data tokens of the same variable remains unchanged throughout the entire computation. Therefore, it parameterizes the data distribution during the execution of the recurrence on the array.

Parameter 1 : Periods

For the uniform recurrence given above, there are 3 dependence directions associated with each variable, $\mathcal{Z}, \mathcal{X}, \mathcal{Y}$. So there are three periods of computation one along each direction. They are defined as follows.

Suppose the time at which a computation is performed is defined by the function τ_c, and the time at which an input is accessed for a particular computation is τ_a, the period of i and j for 2-dimensional outputs are defined as,

$$t_i = \tau_c(\mathcal{Z}^k_{i+1,j}) - \tau_c(\mathcal{Z}^k_{i,j}) \tag{1.24}$$

$$t_j = \tau_c(\mathcal{Z}^k_{i,j+1}) - \tau_c(\mathcal{Z}^k_{i,j}) \tag{1.25}$$

The period of *iterative computation*(k) for 2-dimensional outputs is defined as

$$t_k = \tau_c(\mathcal{Z}^{k+1}_{i,j}) - \tau_c(\mathcal{Z}^k_{i,j}) \tag{1.26}$$

Since the recurrence is assumed to be in backward form, t_k is always positive. In computing $\mathcal{Z}_{i,j}$, data items of $\mathcal{X}$ indexed by $x_{i,k}$ and $x_{i,k+1}$ are accessed successively. So is the case with $y_{k,j}$ and $y_{k+1,j}$. So we could define periods of $\mathcal{X}, \mathcal{Y}$ with respect to k as,

$$t_{kx} = \tau_a(\mathcal{X}(x_{i,k+1})) - \tau_a(\mathcal{X}(x_{i,k})) \tag{1.27}$$

$$t_{ky} = \tau_a(\mathcal{Y}(y_{k+1,j})) - \tau_a(\mathcal{Y}(y_{k,j})) \tag{1.28}$$

t_{kx} and t_{ky} may be negative depending on the order of access defined in subscript-access functions. Since the data needed in the computation of $\mathcal{Z}^{k+1}_{i,j}$ after the computation of $\mathcal{Z}^k_{i,j}$ must be assembled in time t_k, it must be that

$$t_k = |t_{kx}| = |t_{ky}| \tag{1.29}$$

Note that the absolute value of the periods must be greater than or equal to 1 since there is no broadcasting of data.

Parameter 2 : Velocity

Each variable has its own velocity along its dependence direction. Velocity of a datum x is defined as the directional distance passed during a clock cycle and is denoted as x_d. Since PE's are at unit distance from its neighbors, and buffers (if present) must be equally spaced between PE's the magnitude of the velocity must be a rational number of the form i/j where i, j are integers and $i \leq j$ (to prevent broadcasting). This implies that in j clock cycles, x propagates through i PE's and $j - i$ buffers.

Parameter 3 : Spacing or Data distribution

The third set of parameters is spacing or data distribution of the inputs and outputs. For a 2-dimensional array the elements of a row or column are arranged in a straight line and are equally spaced as they pass through the systolic array, and their relative positions are iteration independent. Other forms of data spacing are not considered here. Suppose i and j are row and column indices respectively, the row displacement of $\mathcal{X}$ is defined as the directional distance between $x_{i,j}$ and $x_{i+1,j}$ and is denoted by $\vec{x_{is}}$. Similarly, the column displacement of $\mathcal{X}$ ($\vec{x_{ks}}$) is defined as directional distance between $x_{i,j}$ and $x_{i,j+1}$. If $\mathcal{X}$ is 1-dimensional then the item displacement of $\mathcal{X}$ ($\vec{x_s}$) is defined as directional distance between x_i and x_{i+1}. Since data are spaced equally along row or column the displacements are independent of i or j. Note that the data distribution parameters are defined in a subscript-increasing direction with magnitudes which are rational numbers due to pipelining.

For example, figure 1.1 shows the systolic array for 2-dimensional matrix multiplication. The array A referenced through indices i and k has data distribution vectors defined by $\vec{a_{is}}$ and $\vec{a_{ks}}$. Similarly, array B has $\vec{b_{ks}}$ and $\vec{b_{js}}$. The periods are t_k, t_i, t_j are all equal to 1.

1.2.4 Constraint equations

There are a total of 12 parameters defined for 3-dimensional uniform recurrence in equation 1.10, of which 3 are velocities of data flow, x_d, y_d, z_d, six are data spacings, $\vec{x_{is}}, \vec{x_{ks}}, \vec{y_{ks}}, \vec{y_{js}}, \vec{z_{is}}, \vec{z_{js}}$ and 3 are periods, t_i, t_j, t_k. For 1-dimensional problems only 8 parameters $x_d, a_d, z_d, x_s, a_s, z_s, t_k, t_i$ exist. These parameters cannot be chosen arbitrarily and are related by constraint equations to ensure correctness of the resulting design. The performance of the design (like the completion time or number of PE's) can be expressed in terms of the parameters, which can be chosen to optimize the performance objective.

The following theorem states the relationships among these parameters for the correctness of the target systolic design.

Theorem 1 *Suppose the two-dimensional recurrence in equation 1.10 is implemented in a systolic array, then the velocities, data spacings and periods must satisfy the following vector equations:*

$$t_k \vec{z_d} = \vec{x_{ks}} + t_k \vec{x_d} \tag{1.30}$$

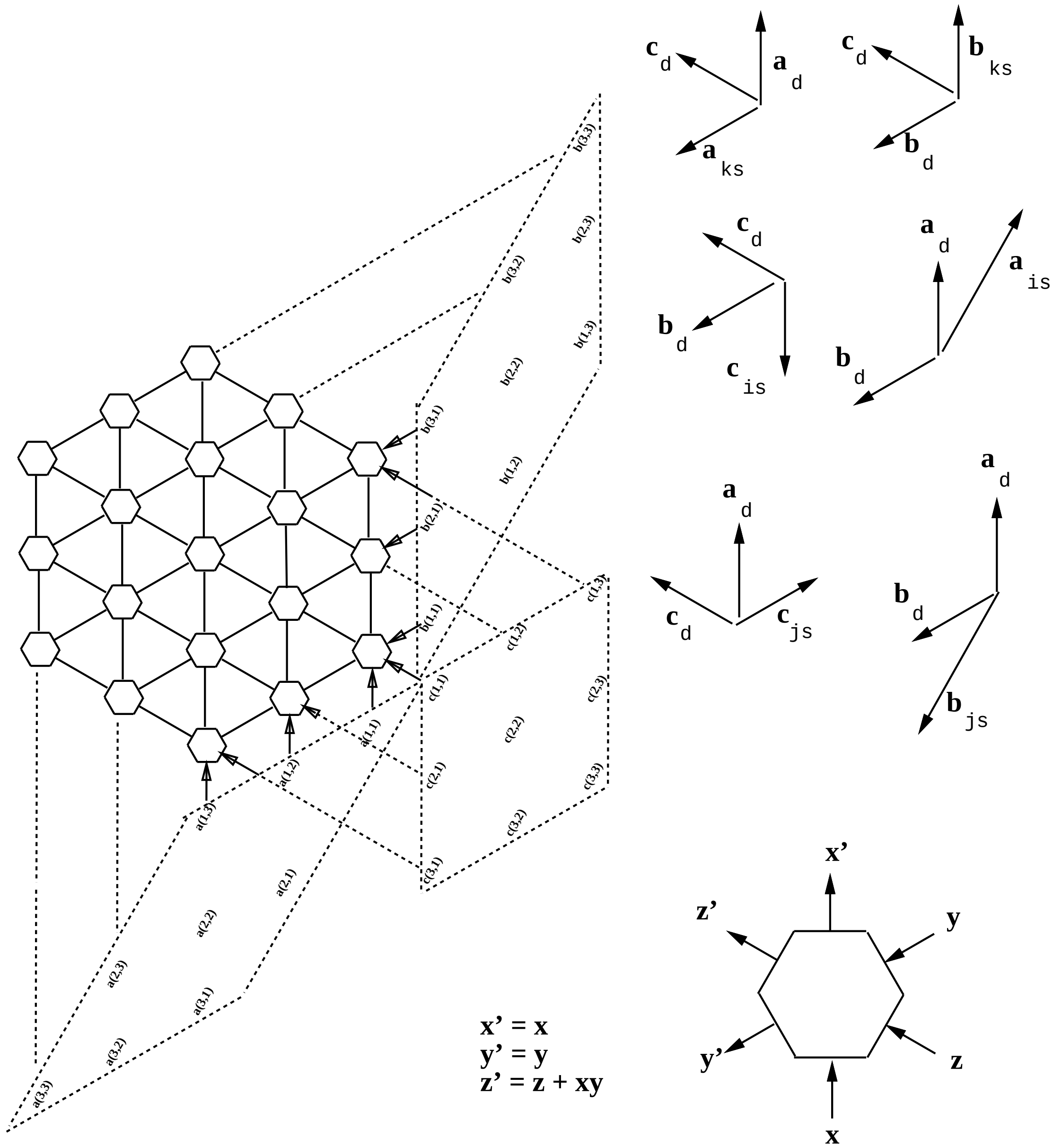

Figure 1.1 **Two dimensional systolic array for multiplying two 3x3 2-dimensional matrices. Also included are the vector equations and a single PE**

$$
\begin{aligned}
t_k \vec{z_d} &= \vec{y_{ks}} + t_k \vec{y_d} && (1.31)\\
t_i \vec{y_d} &= \vec{x_{is}} + t_i \vec{x_d} && (1.32)\\
t_i \vec{y_d} &= \vec{z_{is}} + t_i \vec{z_d} && (1.33)\\
t_j \vec{x_d} &= \vec{y_{js}} + t_j \vec{y_d} && (1.34)\\
t_j \vec{x_d} &= \vec{z_{js}} + t_j \vec{z_d} && (1.35)
\end{aligned}
$$

For 1-dimensional problems, only the first 4 of the above equations are necessary.

The proof of this theorem can be obtained as a special case of the proof for the general recurrence given later in section 1.3.

1.2.5 Design Methodology

The design of the optimal systolic array can be formulated as an optimization problem. The constraints of the optimization provided by Theorem 1, show the fundamental space-time relationships that govern the correctness of the design. The objective function to be optimized is expressed in terms of the parameters and problem size. The design problem is formulated as follows.

$$\text{Minimize } \#\text{PE x } T^2 \text{ or } \#\text{PE x } T \text{ or } \#\text{PE or } T$$

subject to constraints in equations 1.30-1.35. #PE and T refer to the number of processors used and the execution time on the array respectively. Additional constraints on the optimization are

$$
\begin{aligned}
\frac{1}{t_{jmax}} \leq |\vec{x_d}| \leq 1 \quad &or \quad |\vec{x_d}| = 0 && (1.36)\\
\frac{1}{t_{imax}} \leq |\vec{y_d}| \leq 1 \quad &or \quad |\vec{y_d}| = 0 && (1.37)\\
\frac{1}{t_{kmax}} \leq |\vec{z_d}| \leq 1 \quad &or \quad |\vec{z_d}| = 0 && (1.38)
\end{aligned}
$$

$$
\begin{aligned}
&1 \leq |t_k| \leq t_{kmax} \;\; ; \;\; 1 \leq |t_i| \leq t_{imax} \;\; ; \;\; 1 \leq |t_j| \leq t_{jmax} && (1.39)\\
&|t_k|\,|\vec{z_d}| = k_1 \leq t_{kmax} \;\; ; \;\; |t_i|\,|\vec{y_d}| = k_2 \leq t_{imax} \;\; ; \;\; |t_j|\,|\vec{x_d}| = k_3 \leq t_{jmax} && (1.40)\\
&|\vec{x_{is}}|\,,\; |\vec{x_{ks}}|\,,\; |\vec{y_{ks}}|\,,\; |\vec{y_{js}}|\,,\; |\vec{z_{is}}|\,,\; |\vec{z_{js}}| \neq 0 && (1.41)
\end{aligned}
$$

t_{imax}, t_{jmax}, t_{kmax} are the maximum values of the periods that have to be considered in the optimization. These bounds can be derived by requiring the completion time $T \leq T_{serial}$. Then by using the minimum value of 1 for two of the three periods in the inequality, the bound on the third period can be obtained.

k_1, k_2, k_3 are integers representing the distance traveled between computations. Since a computation must be performed in a PE which are at integer locations, the distance traversed must be integral. The upper bounds on k_1, k_2, k_3 are t_{kmax}, t_{imax}, t_{jmax}, respectively, because the maximum permissible values of speeds are 1. Velocities smaller than the lower bound need not be considered as there is a more efficient way of computing the recurrence in a single PE. The other constraints follow directly from the definitions. The velocities and spacings are 2-dimensional (resp. 1-dimensional) vectors for 3-dimensional (resp. 2-dimensional) recurrences.

In our recurrence equation, as there are only 3 variables the data flow has to be in 1, 2 or 3 independent directions. When data flows in two independent directions, without loss of generality they can be assumed to be orthogonal (90^o) to each other. In the case of 3 independent directions of data flow, again without loss of generality they can be assumed to be 120^o to each other.

Since the magnitudes of the velocities and periods are chosen from a finite set, the search space for the parameters is bounded. There are $O(t_{kmax}^2)$ possible combinations of t_k and k_1. Similarly for t_i and $|\vec{y_d}|$, t_j and $|\vec{x_d}|$, there are $O(t_{imax}^2), O(t_{jmax}^2)$ combinations of values, respectively. Therefore, the optimization problem has a finite search space of complexity $O(t_{kmax}^2 t_{jmax}^2 t_{imax}^2)$.

There are two ways to further reduce this complexity. First, instead of requiring that $T \leq T_{serial}$, the requirement that $T \leq O(T_{serial}/\#PE)$ may be used to reduce the upper bounds on the periods. Secondly, the constraint equations for correctness of the design are independent of problem size. Therefore, to reduce the search complexity, an optimal design for a smaller problem can be found and used to extend to a larger version of the same problem. However, the scheme will not lead to an optimal solution for the larger problem in general. This is because the objective function increases monotonically with problem size and the fact that the objective function is better at a given problem size does not imply that this design is better at a different problem size.

1.2.6 Enumeration Procedure

The optimal solution to the design problem can be found by a systematic enumeration over the space of parameters. The procedure for minimizing the completion time T (which is usually a linear function of the periods) is as follows:

1. Compute the upper bounds on periods $t_{kmax}, t_{jmax}, t_{imax}$.

2. Select values of periods t_i, t_j, t_k that minimize the completion time.

3. Choose $k_1 = k_2 = k_3 = 1$ and evaluate the velocities.

4. Solve for the the six unknown spacing parameters from the constraint equations.

5. If no feasible solution is found, increment one of k_1, k_2, k_3 and repeat step 4.

6. If no feasible solution is found still, find another set of periods which increase the completion time by the minimum possible amount. Go to step 3.

Clearly the first feasible solution will also be the optimal one in terms of the completion time.

To minimize $\#\mathrm{PE}.T^2$, it is necessary to know the lower bound on the #PE. The lower bound on #PE for 2-dimensional $(n \times n)$ inputs can be one (both inputs are serial), n (one input is serial and the other has n streams of data flow) or n^2 (both inputs have n streams of data flow). The number of streams of data flow of a matrix $\mathcal{X}$ is the number of distinct parallel lines that must be drawn in the direction of the data flow so that each element is in exactly one stream. It is easy to show that serial inputs do not lead to feasible solutions. and the lower bound is n^2. To determine the design with minimum $PE.T^2$ product, the procedure for minimizing completion time is used to get feasible design. Suppose the feasible design requires P_1 PE's and T_1 time. Then any design with $T_2 \geq \sqrt{P_1}.T_1/n$ will not lead to a better solution and hence can be eliminated from consideration. The search is then continued to find better solution with completion times between T_1 and T_2.

Thus, by systematically enumerating and reducing the search space that is a polynomial in the size of the problem, the optimal design can be found very efficiently.

The basic steps in the design process are listed below.

1. Write the recurrence equation for the problem to be solved

2. Write the objective function based on the design requirements in terms of the parameters and problem size.

3. Write the constraint equations for correct systolic processing and the constraints on the values of the parameters.

4. Find the parameter values that minimize the objective function and satisfy the constraints by enumerating over the limited search space.

5. Design the basic cell for the systolic array and find a possible interconnection of the cells and buffer assignment from the parameters. Eliminate cells that don't perform any useful computation.

In the following section, two examples are provided to illustrate the application of parameter method. First one is matrix multiplication which is a 3-dimensional recurrence. Following it is the example of deconvolution which is a 2-dimensional recurrence with feedback.

1.2.7 Examples

Two-dimensional Matrix Multiplication

The recurrence equation for the multiplication of two 2-dimensional matrices each of size $n \times n$ is given in equation 1.17. The computation time, which is the objective function in this example, is related to the parameters as follows.

Lemma 1 *The computation time T for $n \times n$ matrix multiplication is* $1 + (n-1)(t_k + |t_i| + |t_j|)$

Proof: The critical path in the execution is the computation of $c_{n,n}$. It takes $(n-1)\,|t_i|$ steps to start the computation of $c_{n,1}$ after the start of the computation of $c_{1,1}$. Further $(n-1)\,|t_j|$ steps are needed to start the computation of $c_{n,n}$ from the start of $c_{n,1}$. Additional $(n-1)t_k + 1$ steps are needed to complete the computation of $c_{n,n}$. Thus the total time needed is $1 + (n-1)(t_k + |t_i| + |t_j|)$.

■

The completion time is minimized when the periods are as small as possible. The search is started with $t_k = t_j = t_i = 1$ on all combinations of data flows. For this example, when data flows are in three different directions (120^o to each other without loss of generality) and the periods are $t_k = t_j = t_i = 1$, a feasible solution is obtained that satisfies all the constraint equations. Since the search progressively uncovers solutions with increasing completion time, the first feasible solution is also optimal. The final VLSI structure, basic cell design and solution vectors obtained are shown in Figure 1.1 for $n = 3$. The completion time for general n is $3n - 2$ with $3n(n-1) + 1$ PE's. This is the fastest 2-dimensional systolic array for matrix multiplication as the lower bound on time from the dependence graph is $3n - 2$.

On the other hand, if $\#PE \times T^2$ is to be minimized, the search has to be continued to find all the feasible designs with completion time less than $\sqrt{19\text{x}7^2/3^2} = 10.2$. It turns out that in the optimal solution, the output matrix is stationary, with periods $t_k = t_i = t_j = 1$. The design uses n^2 PEs and the completion time is $4n - 2$ units with $3n - 2$ units of computation time and n units of drain time.

Note that the recurrence for two-dimensional tuple comparison (equation 1.19) is identical to that for matrix multiplication except for the operations performed in the PE's. Hence the systolic array for matrix multiplication could be applied in this case. In fact, matrix multiplication is an instance of a class of algorithms called 3-D Cube graph algorithms [7] whose dependence graph is in the form of a 3-dimensional cube. Other examples of this class are L-U decomposition, triangular Matrix Inversion, 3-pass Transitive closure [2], 2-dimensional tuple comparison. The systolic array for matrix multiplication, with appropriate modifications to the PE design, can be applied to other members in this class also.

Deconvolution

This example illustrates the design of systolic array for recurrences with feedback. When outputs are routed back to the array in the form of feedbacks the direction of data flows has to be changed. The correctness of feedbacks is hard to express in constraint equations because the format of feedback depends on the way outputs are generated. A possible method is to treat the feedbacks as a separate independent input and design the systolic

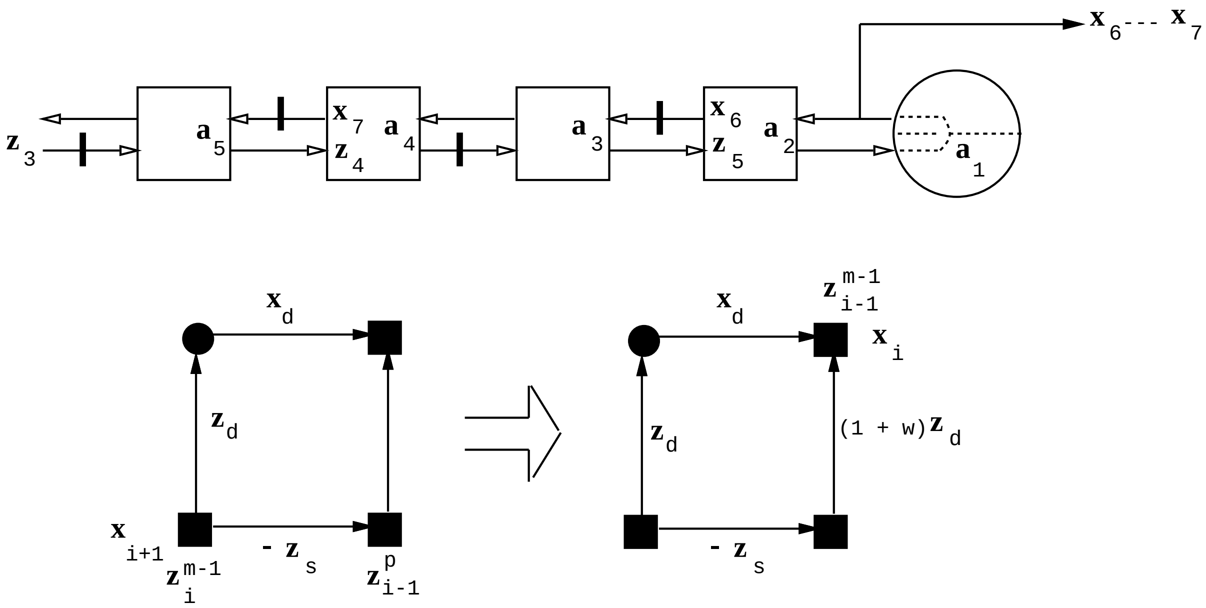

Figure 1.2 Systolic array for deconvolution for $n = 5$, $m = 4$ after 6 clock beats

array in the usual way. After a good design is obtained the feedbacks are checked to determine if they are generated before they are fed back. The process is repeated until a correct design is found.

In the discussion so far, the processing times in the PE's are assumed to be identical. This implies the periods of data flow are independent of the PE along the direction of the flow. However in some applications, different operations may be performed in PE's along a direction of data flow. In the deconvolution example considered, the last operation is a division, which may be bottleneck of data flow. The recurrence for deconvolution shows that the x_i's are fed back into the pipeline for future computations. Let the delay of a PE performing division be w and the delay of other PE's be 1. The last iteration of computing $\mathcal{Z}_i$ followed by the division of $\mathcal{Z}_i$ to obtain x_i takes $w + 1$ units of time. Hence, the period t_i of input $\mathcal{X}$ is given as

$$|t_i| = w + 1 \tag{1.42}$$

In this particular example, the fact that the inputs and feedbacks are one-dimensional lets us write down the relationships of data flows formally. For the following discussion refer to Figure 1.2.

Let PE A be the one where the last iteration of $\mathcal{Z}_i$, $\mathcal{Z}_i^{m-1}$ is computed using x_{i+1} before being sent to the division unit (PE D). The token $\mathcal{Z}_{i-1}$ exists in PE B in its pth iteration ($p < m - 1$) as $\mathcal{Z}_{i-1}^p$. In $(w + 1)$ units of time, $\mathcal{Z}_i^{m-1}$ is sent to PE D converted to x_i and sent to PE C for the computation of $\mathcal{Z}_{i-1}^{m-1}$. In that time, the token $\mathcal{Z}_{i-1}^{p}$ arrives at PE C after completing $(m-1)$ iterations. Hence, a constraint equation can be written as

$$\vec{z_d} + \vec{x_d} = -\vec{z_s} + (w+1)\vec{z_d} \rightarrow \vec{x_d} = w\vec{z_d} - \vec{z_s} \tag{1.43}$$

Equations 1.43 must be included in the optimization with the other constraint equations applicable to 2-dimensional recurrences. Note that $\mathcal{X}$ is a 1-dimensional vector with one spatial-distribution parameter $\vec{x_s}$, although the subscript access function depends on i and k.

A possible solution is

$$\begin{aligned} w &= 2,\ t_k = -3/2,\ t_i = 3, && (1.44)\\ |\vec{a_d}| &= 0,\ |\vec{z_d}| = 2/3,\ |\vec{x_d}| = 2/3 && (1.45)\\ |\vec{a_s}| &= -1,\ |\vec{z_s}| = 2,\ |\vec{x_s}| = -2. && (1.46)\end{aligned}$$

However the problem with this solution is the feedback cannot be generated at the right time from $\mathcal{Z}$. The following values for the parameters yields a solution that minimizes completion time $T = (m-1+w)t_k + (n-1)\,|t_i|$.

The values are

$$\begin{aligned} t_i &= -3,\ \ t_k = -3/2, && (1.47)\\ |\vec{z_d}| &= 2/3, && (1.48)\\ |\vec{z_s}| &= 2, && (1.49)\\ |\vec{x_d}| &= -2/3, && (1.50)\\ |\vec{x_s}| &= -2 && (1.51)\end{aligned}$$

Figure 1.2 shows the systolic array for $m = 4,\ n = 5$ with a feasible assignment of buffers. Note that the velocities represent average over three cycles.

1.3 General Parameter Method

In this section, extension of the parameter method to a synthesize systolic arrays for N-dimensional recurrences is presented. The general form of N-dimensional recurrence considered is

$$\mathcal{Z}(I) = \phi\left[\mathcal{Z}(I-d_1), \mathcal{Z}(I-d_2), \ldots, \mathcal{Z}(I-d_q)\right] + \psi\left[\mathcal{X}_1(\hat{x_1}(I)), \ldots, \mathcal{X}_r(\hat{x_r}(I))\right] \tag{1.52}$$

where I denotes a point (N-dimensional vector) in the domain of the recurrence, d_i, $1 \le i \le q$ are the dependence vectors associated with any point in the domain (uniform recurrence) and $\mathcal{X}_j(\hat{x_j}(I)), 1 \le j \le r$ is the j-th input that is needed in the computation of the function $\mathcal{Z}$. $\hat{x_j}(I)$ is the linear subscript access (indexing) function for the array indices of input $\mathcal{X}_j$. So in the specific recurrence considered in earlier section,

$N = 3$, $M = 2$, $r = 2$, $q = 1$. Note that without loss of generality the dependence vectors d_i can be taken to be distinct.

As in section 1.2.2 if the subscript access function of input $\mathcal{X}_j$ does not involve a particular index variable i_l, then each input token of input $\mathcal{X}_j$ is used at all index points which differ along the index i_l. Thus the input element has to be pipelined through the set of index points it is used. If a subscript access function is such that an input token (element) is used in only one point in the domain then it does not make sense to talk of data flows for that input. That token is used in one computation only and is not needed any more. Hence it could be preloaded and stays in the PE it is going to be used in, for the duration of the computation.

Therefore, a dependence vector is associated with each input which then fixes the sequence of PEs that a token, belonging to that input, has to visit. Thus, r dependencies may have to be introduced to pipeline the flow of the inputs between the computation points.

As an illustration for the discussion on the general parameter method the following recurrence will be used as a running example. This example is used only for illustrating the terminology and the equations derived. The example for which an array is given is the matrix multiplication example considered in section 1.2.

Example A 3-dimensional recurrence with $N = 3, q = 3, r = 2$

$$\mathcal{Z}(i,j,k) = \mathcal{Z}(i,j,k-1) + \mathcal{Z}(i,j-2,k) + \mathcal{Z}(i-3,j-1,k) + \mathcal{X}(j,k)\,\mathcal{Y}(k,i) \quad (1.53)$$

where $\mathcal{Z}, \mathcal{X}, \mathcal{Y}$ are 2-dimensional $n \times n$ matrices. The index space is a 3-dimensional cube of size $n \times n \times n$ if k ranges from 1 through n.

The indexing functions for the inputs are

$$\hat{x} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (1.54)$$

$$\hat{y} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \quad (1.55)$$

$\hat{x}(I)$ and $\hat{y}(I)$ gives the indices for the inputs $\mathcal{X}$ and $\mathcal{Y}$ respectively.

The dependence vectors are collected into a matrix as follows:

$$D = \begin{bmatrix} 0 & 0 & 3 & 1 & 0 \\ 0 & 2 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (1.56)$$

$$\mathcal{Z} \quad \mathcal{Z} \quad \mathcal{Z} \quad \mathcal{X} \quad \mathcal{Y} \quad (1.57)$$

Pipelining leads to dependencies (1, 0, 0) and (0, 1, 0) for variables $\mathcal{X}$ and $\mathcal{Y}$ respectively.

$$\hat{x}(i,j,k) = \hat{x}(i-1,j,k) \quad (1.58)$$

$$\hat{y}(i,j,k) = \hat{y}(i,j-1,k) \quad (1.59)$$

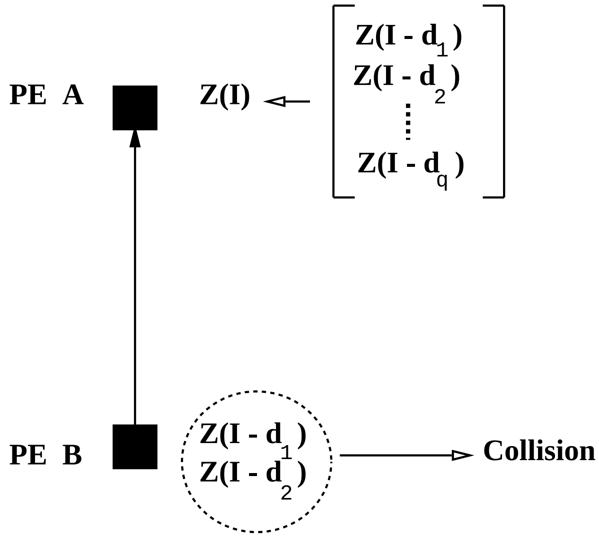

Figure 1.3 Demonstrates the need for different velocities along different dependence directions. If the velocities of $\mathcal{Z}(I-d_1)$ and $\mathcal{Z}(I-d_2)$ are the same collision occurs in PE B

■

Next we present an informal discussion on why the parameter set considered earlier in section 1.2 needs to be augmented for the case of general recurrence. After the inputs have been pipelined, there are $q+r$ dependence directions for the given recurrence in the domain. We need to have a different velocity for data motion along each dependence direction. Suppose for contradiction, let the velocity of recurrence variable $\mathcal{Z}$ be the same along directions d_1 and d_2. Then in the computation of $\mathcal{Z}(I)$ at PE A (illustrated in figure 1.3), $\mathcal{Z}(I-d_1)$ $\mathcal{Z}(I-d_2)$ are needed at the same processor (A). Since their velocities are the same they must have traveled together and must have been generated at the same time at processor B. This violates our *fundamental* assumption that no two computation can be mapped to the same processor at the same time.

Since the velocities are different along each dependence direction, the data distribution or spacing has to be different for variable $\mathcal{Z}$ in each direction. In addition a different period of computation has to be associated with each dependence direction to avoid computational conflicts. Thus for each dependence direction there is a velocity of data flow, period of computation and distribution parameters for the data flowing in that direction.

From the structure of the recurrence given in equation 1.52, the value $\mathcal{Z}(I)$ is computed at point I ($\in$ *domain*) by combining q other values $\mathcal{Z}(I-d_i)$ ($1 \leq i \leq q$) generated at other points in the domain. Similarly the value $\mathcal{Z}(I)$ is used at q other points (called consumer points), $\mathcal{Z}(I+d_i), 1 \leq i \leq q$. There are 3 ways of sending this value of $\mathcal{Z}(I)$

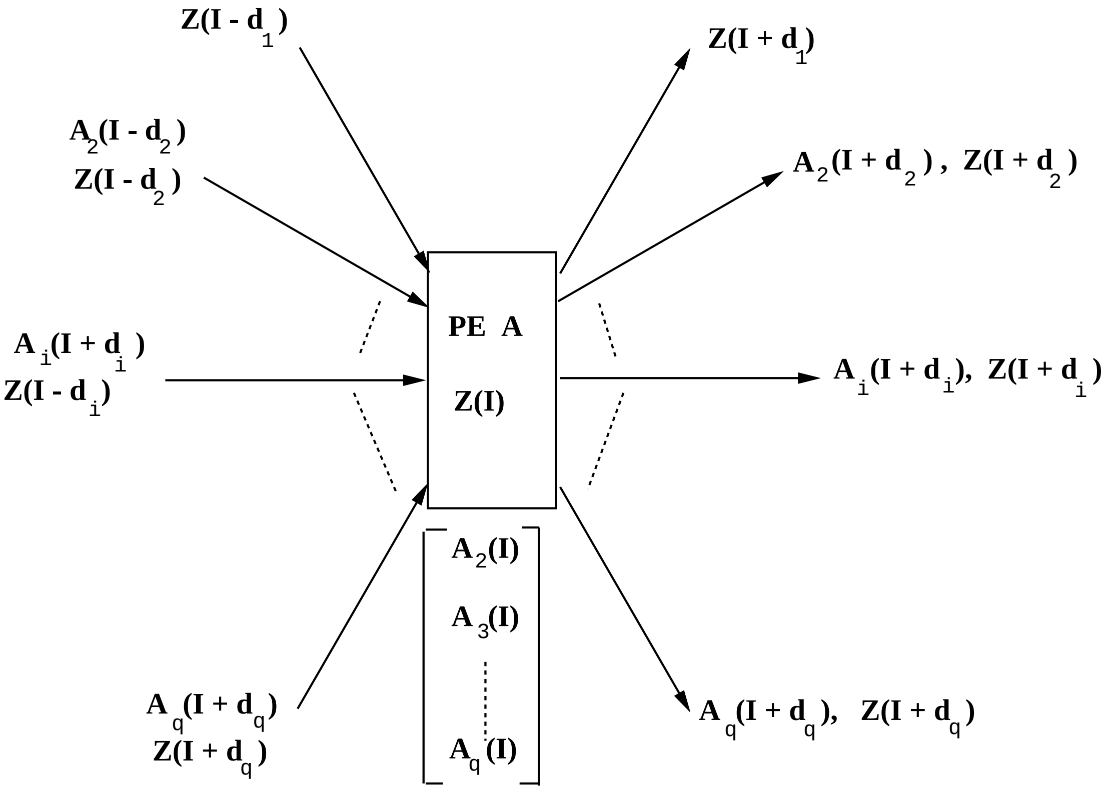

Figure 1.4 A single PE computing the recurrence in the general parameter method

to the points where it is used (Figure 1.4).

1. *Full Replication:* q copies of $\mathcal{Z}(I)$ are sent to the q points which need it.

2. *Full Propagation:* Only copy of $\mathcal{Z}(I)$ exists and this copy is sequenced through its q consumer points. Thus these q consumers can be thought of to be on a chain of size q.

3. *Mixed:* $k(< q)$ copies of $\mathcal{Z}(I)$ are made at PE P, where $\mathcal{Z}(I)$ is generated and each copy is sequenced through a subset of the q consumer points. Thus multiple smaller chains of size less than q are formed.

Note that the way in which $\mathcal{Z}(I)$ is sent to its consumers could change the structure of the target systolic array significantly. In this discussion only the case of *full replication* is considered.

Since we need to have a different set of parameters (velocity, spacing and period) along each dependence direction, the recurrence is rewritten in the following manner to explicitly bring this out and simplify the reasoning about the recurrence.

$$\begin{aligned}\mathcal{Z}(I) &= \phi\left[\mathcal{Z}(I-d_1), \mathcal{A}_2(I-d_2), \mathcal{A}_3(I-d_3), \ldots, \mathcal{A}_q(I-d_q)\right] \\ &\quad + \psi\left[\mathcal{X}_1(\hat{x_1}(I)), \ldots, \mathcal{X}_r(\hat{x_r}(I))\right] &(1.60)\\ \mathcal{A}_i(I) &= \mathcal{Z}(I), \quad i = 2, \ldots, q &(1.61)\end{aligned}$$

With this form of the recurrence each dependence direction is associated with exactly one variable. Each of the $\mathcal{A}_i$ indicate the movement of $\mathcal{Z}$ along the direction d_i. Note that the rewritten form is functionally identical to the original recurrence in terms of the computations performed. The rewritten form is, however, conceptually simpler to deal with. The elements of each of the variables move periodically between computations with constant velocity. Thus the distance between the elements of that variable remains constant and the data distribution can be captured by a set of spacing parameters.

Example The rewritten form of the recurrence is

$$\begin{aligned}\mathcal{Z}(i,j,k) &= \mathcal{Z}(i,j,k-1) + \mathcal{A}_2(i,j-2,k) + \mathcal{A}_3(i-3,j-1,k) \\ &\quad + \mathcal{X}(\hat{x}(i,j,k))\, \mathcal{Y}(\hat{y}(i,j,k)) &(1.62)\\ \mathcal{A}_2(i,j,k) &= \mathcal{Z}(i,j,k) &(1.63)\\ \mathcal{A}_3(i,j,k) &= \mathcal{Z}(i,j,k) &(1.64)\end{aligned}$$

Each dependency is associated with one variable now.

■

1.3.1 Constraint equations

For the general form of recurrence the parameter set is as below. For ease of notation the variables $\{\mathcal{Z}, \mathcal{A}_i, \mathcal{X}_j\}$ are ordered, so that,

$$i-th \text{ Variable refers to } \begin{cases} \mathcal{Z} & if \;\; i = 1, \\ \mathcal{A}_i & if \;\; i = 2, 3, \ldots, q \\ \mathcal{X}_{i-q} & if \;\; i = q+1, q+2, \ldots, q+r \end{cases}$$

For each $i \in (1, 2, \ldots, q+r)$, $t_i = \tau_c(\mathcal{Z}(I)) - \tau_c(\mathcal{Z}(I-d_i))$ is the period between the computations of variable i in the ordering defined above. $\tau_c(\mathcal{Z}(I))$ is the time at which computation $\mathcal{Z}(I)$ is executed on the array. $\vec{V_i}$ is the velocity of flow of the token of i-th variable in the direction d_i. For instance, variable i moves along direction d_i through the points (called dependence chain of i) in the domain of the recurrence as follows.

$$\ldots \mathcal{Z}(I-2d_i) \rightarrow \mathcal{Z}(I-d_i) \rightarrow \mathcal{Z}(I) \rightarrow \mathcal{Z}(I+d_i) \rightarrow \mathcal{Z}(I+2d_i) \ldots$$

The spacing or data distribution parameters are denoted by S_i^d, and defined as the directional distance between the elements of variable i along direction $\vec{d}$ *i.e* between elements $\mathcal{X}_i(\hat{x}_i(I-d))$ and $\mathcal{X}_i(\hat{x}_i(I))$. The number of the spacing parameters will be clear once the constraint equations are given. Note that the magnitude of all velocities $\left|\vec{V_i}\right| \leq 1$ since broadcasting is not permitted.

Example $\mathcal{Z}$ is the 1-st variable, $\mathcal{A}_2$ is the 2-nd variable, $\mathcal{A}_3$ is the 3-rd variable, $\mathcal{X}$ is the 4-th and $\mathcal{Y}$ is the 5-th variable in the ordering defined above. The parameters t_1, t_2, t_3, t_4, t_5 are the periods, $\vec{V_1}, \vec{V_2}, \vec{V_3}, \vec{V_4}, \vec{V_5}$ are the velocities.

$$t_1 = \tau_c(\mathcal{Z}(i,j,k)) - \tau_c(\mathcal{Z}(i,j,k-1)) \tag{1.65}$$
$$t_2 = \tau_c(\mathcal{Z}(i,j,k)) - \tau_c(\mathcal{Z}(i,j-2,k)) \tag{1.66}$$
$$t_3 = \tau_c(\mathcal{Z}(i,j,k)) - \tau_c(\mathcal{Z}(i-3,j-1,k)) \tag{1.67}$$
$$t_4 = \tau_c(\mathcal{Z}(i,j,k)) = \tau_c(\mathcal{Z}(i-1,j,k)) \tag{1.68}$$
$$t_5 = \tau_c(\mathcal{Z}(i,j,k)) = \tau_c(\mathcal{Z}(i,j-1,k)) \tag{1.69}$$

■

Theorem 2 *The parameters velocities, spacings and periods must satisfy the following constraint equations for correct systolic processing of the general recurrence:*

$$\begin{aligned} \vec{V_i}t_i &= \vec{V_j}t_i + S_j^{d_i-d_j}, \quad 1 \leq i \leq q+r, \quad 1 \leq j \leq q, \quad j \neq i \\ &= \vec{V_j}t_i + S_j^{d_i}, \quad 1 \leq i \leq q+r, \quad q+1 \leq j \leq q+r, \quad j \neq i \end{aligned} \tag{1.70}$$

Proof: Consider the computation of $\mathcal{Z}$ at the index point $I \in D$ (domain). Let the PE where it is computed be denoted by A. To compute $\mathcal{Z}$ all the $q+r$ variables are needed at A at the same time. For simplicity consider the movement of 1-st variable $\mathcal{Z}(I-d_1)$ to PE A. Let B denote the PE where it was generated. In the time $\mathcal{Z}(I-d_1)$ moves from PE B to PE A, the other $q+r-1$ variables must move from their respective locations to PE A. Consider the movement of one such variable j, where $1 \leq j \leq q+r$.
Case 1: $1 \leq j \leq q$
Without loss of generality, consider the motion of the 2-nd variable ($\mathcal{A}_2$). Since $\mathcal{Z}(I-d_1)$ was generated at PE B, $\mathcal{A}_2(I-d_1)$ resides at PE B. When $\mathcal{Z}(I-d_1)$ was generated, $\mathcal{A}_2(I-d_2)$ might not exist in the array. In general, let $\mathcal{A}_2(I-pd_2)$, $p \geq 1$ be the latest value generated along the dependence chain passing through $\mathcal{Z}(I)$. Therefore if $t_2 > t_1$ then $p > 1$ else $p = 1$.
Case 1a: $p = 1$.
This is depicted in Figure 1.5. $S_2^{d_1-d_2}$ denotes the distance between $\mathcal{A}_2(I-d_1)$ and $\mathcal{A}_2(I-d_2)$. Therefore by vector composition we get equation 1.70.
Case 1b: $p > 1$.
As depicted in Figure 1.6, the distance between $\mathcal{A}_2(I-d_1)$ and $\mathcal{A}_2(I-pd_2)$ is now given by $S_2^{d_1-pd_2}$. Hence it appears that we need to know the value of p or how far has the computation of $\mathcal{A}_2$ has proceeded along the computation chain. But crucial idea is that the token or element of the variable $\mathcal{A}_2$ which is now called $\mathcal{A}_2(I-pd_2)$ will eventually go

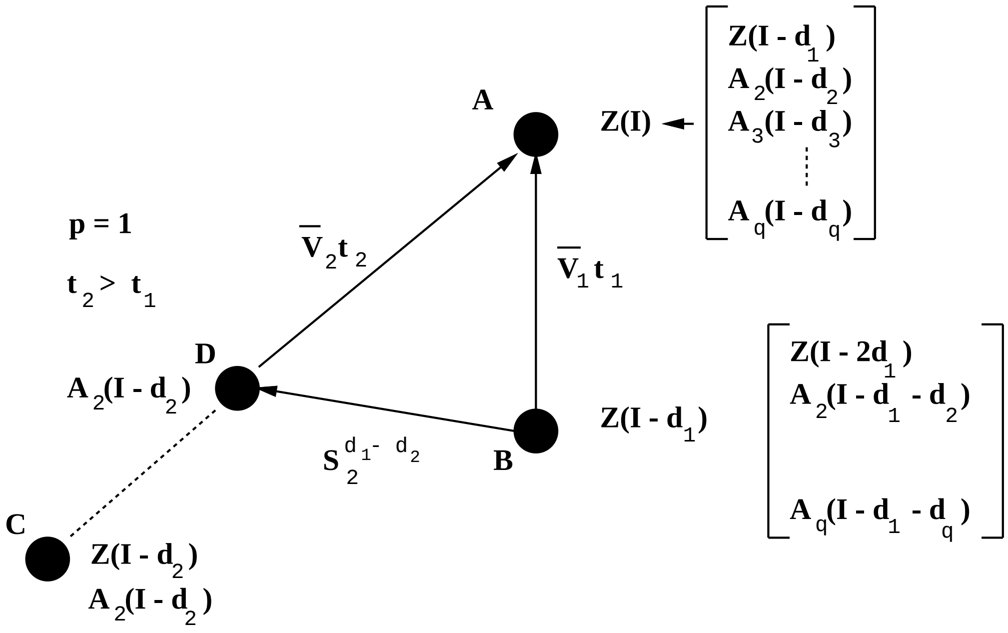

Figure 1.5 General Parameter Method : Data movement between variable 1 and 2. The dashed line indicates the movement in the past *i.e.,* $\mathcal{A}_2(I - d_2)$ **has been generated**

through $p-1$ computations along direction d_2 and be called $\mathcal{A}_2(I-d_2)$. Thus $\mathcal{A}_2(I-pd_2)$ for all p is the *same* element or token of the data stream $\mathcal{A}_2$. This can be visualized as a register (data element) propagating through the array and having different values at different times in the execution. The apparent difficulty was because the register was referred to by the value it contains and this value kept changing through the execution. Therefore, irrespective of the value of p, we can say that the token which will be called $\mathcal{A}_2(I - d_2)$ in future, is at a distance $S_2^{d_1-d_2}$ from PE B which has $\mathcal{A}_2(I - d_1)$ at the present time. Thus again by vector composition the theorem is proved.
Case 2: $q + 1 \leq i \leq q + r$.
Arguments similar to case 1 can be provided to prove the theorem.

$\blacksquare$

Example The constraint equations are

$$\vec{V_1}t_1 = \vec{V_2}t_1 + S_2^{(0,-2,1)} \tag{1.71}$$

$$= \vec{V_3}t_1 + S_3^{(-3,0,1)} \tag{1.72}$$

$$= \vec{V_4}t_1 + S_4^{(0,0,1)} \tag{1.73}$$

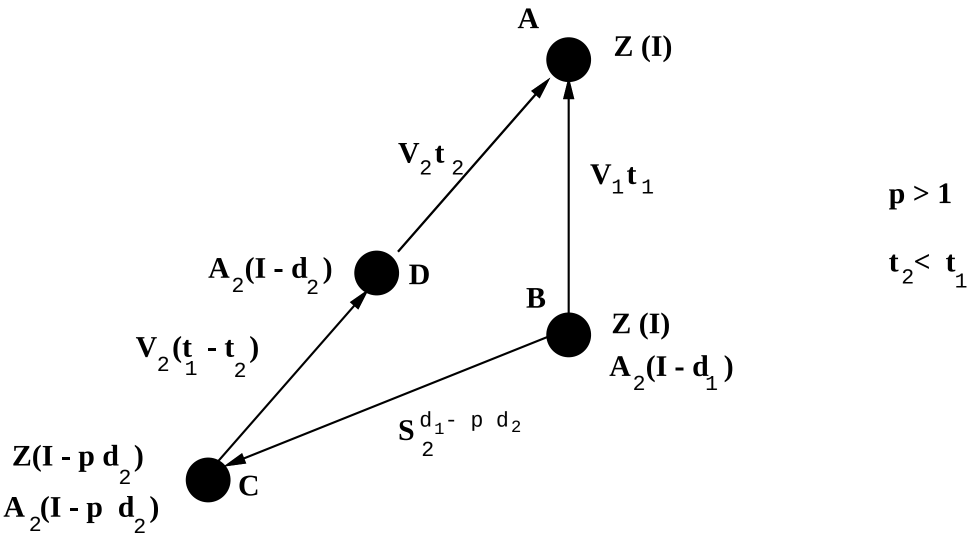

Figure 1.6 Data movement between variables 1 and 2 for $p > 1$. The token $\mathcal{A}_2(I - p.d_2)$ becomes $\mathcal{A}_2(I - d_2)$ as it travels towards PE A

$$= \vec{V}_5 t_1 + S_5^{(0,0,1)} \tag{1.74}$$

Similarly, there are 16 additional equations for the 2-nd, 3-rd, 4-th and 5-th variables. There are 4 spacing parameters with each variable. For the 2-nd variable, $S_2^{(0,-2,1)}, S_2^{(3,-2,0)}, S_2^{(1,-2,0)}, S_2^{(0,-1,0)}$ are the spacings and $S_2^{(0,-2,1)}$ is the directional distance from $\mathcal{A}_2(i,j,k-1)$ and $\mathcal{A}_2(i,j-2,k)$ and so on.

1.3.2 Generating lower dimensional arrays

In the previous subsection, the theorem provided constraint equations based on recurrence being mapped without any idea of the dimension of the target systolic array. The key observation is that the systolic constraint equations are vector equations. So if the target systolic array is M-dimensional the velocity and spacing parameters are M-dimensional too. Hence, the constraint equations are solved in M-dimensions to get M-dimensional solutions.

To determine the number of parameters to be chosen, let us do a count of the number of equations available and the number of parameters defined. For each one of the $q + r$ variables, there is a constraint equation relating its movement to the movement of other $q + r - 1$ variables. Hence there are $(q + r)(q + r - 1)$ vector equations. For each variable there is total of $q + r - 1$ spacing parameters each of which appears in the equations relating its movement to every one of other of $q + r - 1$ variables. In addition each variable has a velocity and period of computation. Thus there are $(q + r + 1)$ parameters for each variable, and $(q+r)(q+r+1)$ parameters in total. The number of parameters exceeds the number of equations by $2(q+r)$ and therefore $2(q+r)$ of the parameters have to be chosen. The remaining $(q+r)(q+r-1)$ parameters can be determined from the $(q+r)(q+r-1)$ constraint equations.

To decide which of the $2(q+r)$ parameters have to be chosen, we look at how the performance of the design such as total execution time T or number of PE's (#PE) is related to the parameters. The performance of the design is usually a function of the periods and velocities only. Hence, a strategy could be to choose the $2(q+r)$ periods and velocities to optimize a performance criterion and then determine the the other $(q+r)(q+r-1)$ spacing parameters from the constraint equations. It should be pointed out that any objective that is expressible in terms of the parameters defined, can be used in the optimization instead of completion time or the #PE in the design. However, the enumeration procedure might have to be modified appropriately if the objective function is not monotonically increasing with the parameters such as periods or velocities.

Example There are 5 variables and each variable has 1 period, 1 velocity and 4 s-pacings. So total number of parameters is 30. Each variable has 4 vector equations and so the total number of equations is 20. So 10 of the 30 parameters are chosen and the remaining 20 parameters are determined from the equations. We can choose the 5 velocities and 5 periods to optimize completion time or number of PE's in the array.

■

The goal is to generate a M-dimensional array for the N-dimensional recurrence where M is in the range $(1, 2, 3, \ldots, N-1)$. (For $M \geq N$ we have a PE at each computation point in the N-dimensional domain and there is no need for any synthesis). In order to obtain a clear understanding, consider the cases for different values of M, q, r. The discussion below is with respect to any arbitrary input j. Let the dimension of the input Matrix be H. Let the size of the input matrix be $(L_1 \times L_2 \times \ldots \times L_H)$.

Since the dimension of the input Matrix is H, the position of every element can be described by a basis of size H *i.e.,* H linearly independent vectors. But since the target array is M-dimensional the matrix is fed as a M-dimensional array. A total of $(q+r-1)$ spacing parameters are obtained from the constraint equations for systolic processing. It is possible that these spacings are such that two elements of the input matrix are sent into the matrix together into the same PE and travel together throughout the execution (since they have the same velocity). This is called a *data conflict* as two data tokens are in the same physical location at all times.

Data conflicts are not permissible within the framework, as extra control bits that travel with the two elements are necessary to determine which of the two inputs is to be used in the computation. Therefore, if we concerned only with *pure* systolic arrays without explicit control bits then no two data tokens must travel together into any PE in the array. The following lemma expresses this notion mathematically.

Let G be the minimum of H and $(q+r-1)$. If $H < (q+r-1)$ only H of the spacings are necessary. The additional $(q+r-1)-H$ spacing parameters must be checked to ascertain that they do not conflict with the H chosen parameters. If $H > (q+r-1)$ then the remaining $(q+r-1)-H$ spacings can be chosen arbitrarily. Therefore, there are at least G spacing parameters obtained from constraint equations. Let $S^1, S^2, \ldots, S^G$ be the G spacing parameters, each of which is M-dimensional. Let S be a $G \times M$ which is given by $S = \left[S^1, S^2, \ldots, S^G\right]^t$ where t refers to the transpose of the matrix. Let α, β, γ be vectors with G integer elements.

Lemma 2 *Data conflicts occur in the input Matrix iff* $\alpha S = \vec{0}, \alpha \neq \vec{0}$, *where* $\alpha_i \in [-(L_i - 1), \ldots, (L_i + 1)], \forall i$ *such that* $1 \leq i \leq G$.

Proof:

Data conflicts

$$
\begin{aligned}
&\iff \beta S = \gamma S, \;\; \beta \neq \gamma \;\text{ and }\; 1 \leq \gamma_i, \;\; \beta_i \leq L_i \\
&\iff (\beta - \gamma) S = \vec{0} \\
&\iff \alpha S = \vec{0}, \;\; \alpha = \beta - \gamma, \;\; \alpha_i \in [-(L_i - 1), \ldots, (L_i + 1)], \;\; \alpha \neq \vec{0}
\end{aligned}
$$

■

It is instructive to see how the lemma applies to the simplified recurrence case described in section 1.2. In the simplistic case, $H = 2,\ M = 2,\ N = 3,\ q + r - 1 = 2$. From the structure of the constraint equations it can be seen that if the data flow is in two or three independent directions, the 2-dimensional spacing parameters will along two independent directions and hence linearly independent. Thus, the lemma is automatically satisfied irrespective of the size of the matrix and there are no data conflicts.

Example The dimensions of X, Y, and Z matrices are 2 and their sizes are all $n \times n$. The constraint equations result in 4 spacing parameters to describe 2-dimensional matrices. So 2 of the 4 spacing parameters are dependent on the other two. In this example for input $\mathcal{X}$ with the indexing function $\hat{x}$ let the 4 spacing parameters be $S_4^1, S_4^2, S_4^3, S_4^4$, which describe the distance from $\{(\mathcal{X}(i,j) \rightarrow \mathcal{X}(i,j+1)), (\mathcal{X}(i,j) \rightarrow \mathcal{X}(i+2,j)), (\mathcal{X}(i,j) \rightarrow \mathcal{X}(i+1,j)), (\mathcal{X}(i,j) \rightarrow \mathcal{X}(i+1,j))\}$ respectively. So S_4^3 should be equal to S_4^4 and S_4^2 should be twice S_4^3. These additional constraints have to be introduced to have consistent spacing parameters.

If the array sought is 1-dimensional then the spacing parameters are all one dimensional scalars. So S_4^1, S_4^3 are the two independent spacings for input $\mathcal{X}$. Therefore according to the lemma, data conflicts occur in input $\mathcal{X}$ if and only if

$$
[\alpha_1 \alpha_2] \begin{bmatrix} S_4^1 \\ S_4^3 \end{bmatrix} = 0 \tag{1.75}
$$

where $-(n-1) \leq \alpha_1, \alpha_2 \leq (n-1)$ and $\alpha_1, \alpha_2 \neq 0$.

1.3.3 Enumeration Procedure

As done earlier, we define $k_i = t_i\vec{V_i}, 1 \leq i \leq q + r - 1$ as the distance traversed by the elements of variable i between two computations.

Additional constraints on the optimization are that magnitude of velocities are less than unity and none of the spacing parameters can be zero. These are similar to the ones given for the simplified 3-dimensional recurrence in section 1.2 are not repeated here.

The completion time T is a function (usually linear) of the periods of the variables. The upper bounds for the periods are computed in a similar manner as outlined in section 1.2.5. Let t_{imax} denote the upper bound for period t_i. If the upper bounds are computed naively then t_{imax} is a of the order of sequential execution time T_{serial} , which is polynomial in the size of the problem. The procedure is as follows:

1. Compute the upper bounds $t_{imax}, 1 \leq i \leq (q + r - 1)$.

2. Choose values of periods to minimize T.

3. Choose the distances k_i as unity and compute the velocities.

4. Solve for the spacing parameters from the constraint equations.

5. Check for data conflicts from the spacing parameters (Lemma 2).

6. If no feasible solution is found, increment one of k_i and repeat steps 4,5 until no k_i can be increased.

7. If there is no feasible solution still, find another set of periods which increase the completion time by the lowest possible value. Go to step 3.

The complexity of the search procedure is polynomial in t_{imax}, $1 \leq i \leq q + r$. Thus by a systematic enumeration of the limited search space the optimal design can be found very efficiently.

The next section describes the example of matrix multiplication to illustrate the general parameter method. This example was chosen because it represents an important class of algorithms and deriving optimal arrays for it would be a significant step. The target array sought is linear in this case ($M = 1,\ N = 3$).

1.3.4 Example : 3-dimensional Cube graph Algorithms

These are algorithms with dependence graphs as a 3-dimensional cube. They have $[(1,0,0)^t, (0,1,0)^t, (0,0,1)^t]$ as their dependence vectors. Matrix multiplication is the popular example in this class. In this subsection, a 1-dimensional array (as opposed to the 2-D array) for matrix multiplication is presented.

The recurrence equation for matrix multiplication before pipelining is

$$\mathcal{Z}^k_{i,j} = \mathcal{Z}^{k-1}_{i,j} + x_{i,k}y_{k,j} \tag{1.76}$$

Hence, $N = 3, q = 1, r = 2$ and $M = 1$ since a linear array is sought. Let the size of the matrix be $L \times L$.

There are 3 variables, 6 constraint equations and a total of 12 parameters. The parameters and constraint equations are as described in section 1.2 (equations 1.30-1.35). The vector equations result in two distinct spacing parameters for each variable $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{Z}$. Thus there are no data conflicts if the target array is 2-dimensional. If the target array is linear (1-dimensional) then the two spacings of each variable has to satisfy lemma 2. For this example the conditions for data conflicts can be refined as follows.

Consider one of the variables, say $\mathcal{X}$. The spacings parameters of $\mathcal{X}$ are x_{is} and x_{ks} (they are written as scalars since the array is linear).

Lemma 3 *Data conflicts occur in input $\mathcal{X}$ if and only if,*

$$\left|\frac{x_{is}}{m}\right| < L, \quad \textit{and} \quad \left|\frac{x_{ks}}{m}\right| < L \tag{1.77}$$

where $m = GCD(x_{is}, x_{ks})$ and GCD(a,b) is the greatest common divisor of a and b.

Proof:

($\Longleftarrow$) Since m is the $GCD(x_{is}, x_{ks})$, $x_{is} = m.\alpha_2$ and $x_{ks} = m.\alpha_1$, where α_2, α_1 (which denote the product of the rest of the divisors) are integral.

Therefore,

$$\frac{x_{is}}{\alpha_2} = \frac{x_{ks}}{\alpha_1}, \text{ where } |\alpha_1|, |\alpha_2| < L \tag{1.78}$$

$$\Longrightarrow \left[\ \alpha_1 \alpha_2^{'}\ \right] \begin{bmatrix} x_{is} \\ x_{ks} \end{bmatrix} = 0, \text{ where } \alpha_2^{'} = -\alpha_2, |\alpha_1|, |\alpha_2| < L \tag{1.79}$$

$$\Longrightarrow \quad \text{Data conflicts in X}$$

($\Longrightarrow$) Data conflicts in X

$$\Longrightarrow \quad x_{is}\alpha_1 = x_{ks}\alpha_2 \tag{1.80}$$

$$\Longrightarrow \quad \frac{x_{is}}{\alpha_2} = \frac{x_{ks}}{\alpha_1} \tag{1.81}$$

where $\alpha_1,\ \alpha_2 \in \{-(L-1), \ldots, (L-1)\}$ and $GCD(\alpha_1,\ \alpha_2) = 1$ (if not scale α_1 and α_2 by their GCD)

Note that if $\frac{x_{is}}{\alpha_2}$ is not integral, then x_{js} will not be integral. Therefore, $\frac{x_{is}}{\alpha_2}, \frac{x_{ks}}{\alpha_1}$ are both integral and must be equal to their GCD.

$$\frac{x_{is}}{\alpha_2} = \frac{x_{ks}}{\alpha_1} = m \tag{1.82}$$

$$\Longrightarrow \quad \alpha_2 = \frac{x_{is}}{m} \text{ and } \alpha_1 = \frac{x_{ks}}{m} \tag{1.83}$$

$$\Longrightarrow \quad \left|\frac{x_{is}}{m}\right| < L \text{ and } \left|\frac{x_{ks}}{m}\right| < L \tag{1.84}$$

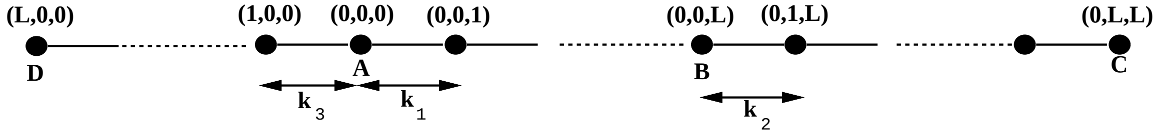

Figure 1.7 PE allocation when $\mathcal{Z}$ and $\mathcal{Y}$ flow to the right and $\mathcal{X}$ moves to the left

■

The following lemma relates the #PE needed in the linear array to the distance parameters k_1, k_2, k_3 which are as defined in section 1.2.5.

Lemma 4 *The #PE needed to execute a $L \times L$ matrix multiplication on a linear array is* $(L-1)(k_1+k_2+k_3)+1$.

Proof: Since all velocities are in 1-dimension (linear array) two of the three velocities should be in the same direction. Without loss of generality assume that the two velocities are to the right (refer to Figure 1.7 for the proof). Assume that k_1 and k_2 are the displacements which correspond to the velocities flowing to the right. Since the constraint equations are symmetric in $\mathcal{Z}, \mathcal{Y}, \mathcal{X}$ no generality is lost here. Let A be the PE where the computation indexed by $(0,0,0)$ occurs. Therefore the computation $(0,0,L)$ is executed at a PE (B) which at a distance of $(L-1)k_1$ from A. Similarly, the computation $(0,L,L)$ is executed at PE C which is $(L-1)k_2$ PE's to the right of B. Now the computation $(L,0,0)$ is at a distance of $(L-1)k_3$ in PE D to the left of PE A (since k_3 corresponds to the left moving variable $\mathcal{X}$). All other computation in the domain are executed by PE's between C and D. Therefore the total number of PE's required are $(L-1)(k_1+k_2+k_3)+1$.

■

Table 1.1 shows the optimal linear designs found by the enumeration procedure when the objective is to minimize completion time. LK is the linear array design proposed by Lee and Kedem in [8]. Their approach is similar to the dependency method outlined in section 1.1. The schedule vector is $(1,2,n-1)$ and the PE allocation matrix (vector in the case of a linear array) is $(1,1,-1)$ for $n \times n$ matrices. But the above transformation is only *asymptotically* optimal (for large n). But using the parameter method the optimal design at every n can be determined very efficiently. The table shows the deviation of the LK design from the optimal which is determined using the parameter method. The performance of a heuristic parametric design (H1) which is also asymptotically optimal but does better than the LK design is included in the table. The parameter set for this design are $(t_k, t_i, t_j) = (1, 2, \lceil \frac{L+1}{2} \rceil)$ and $(k_1, k_2, k_3) = (1, 1, \lceil \frac{L+1}{2} \rceil - 1)$. The spacing parameters can be uniquely determined from the constraint equations and are not given. The linear array together with the data flows for $L = 4$ is depicted in Figure 1.8.

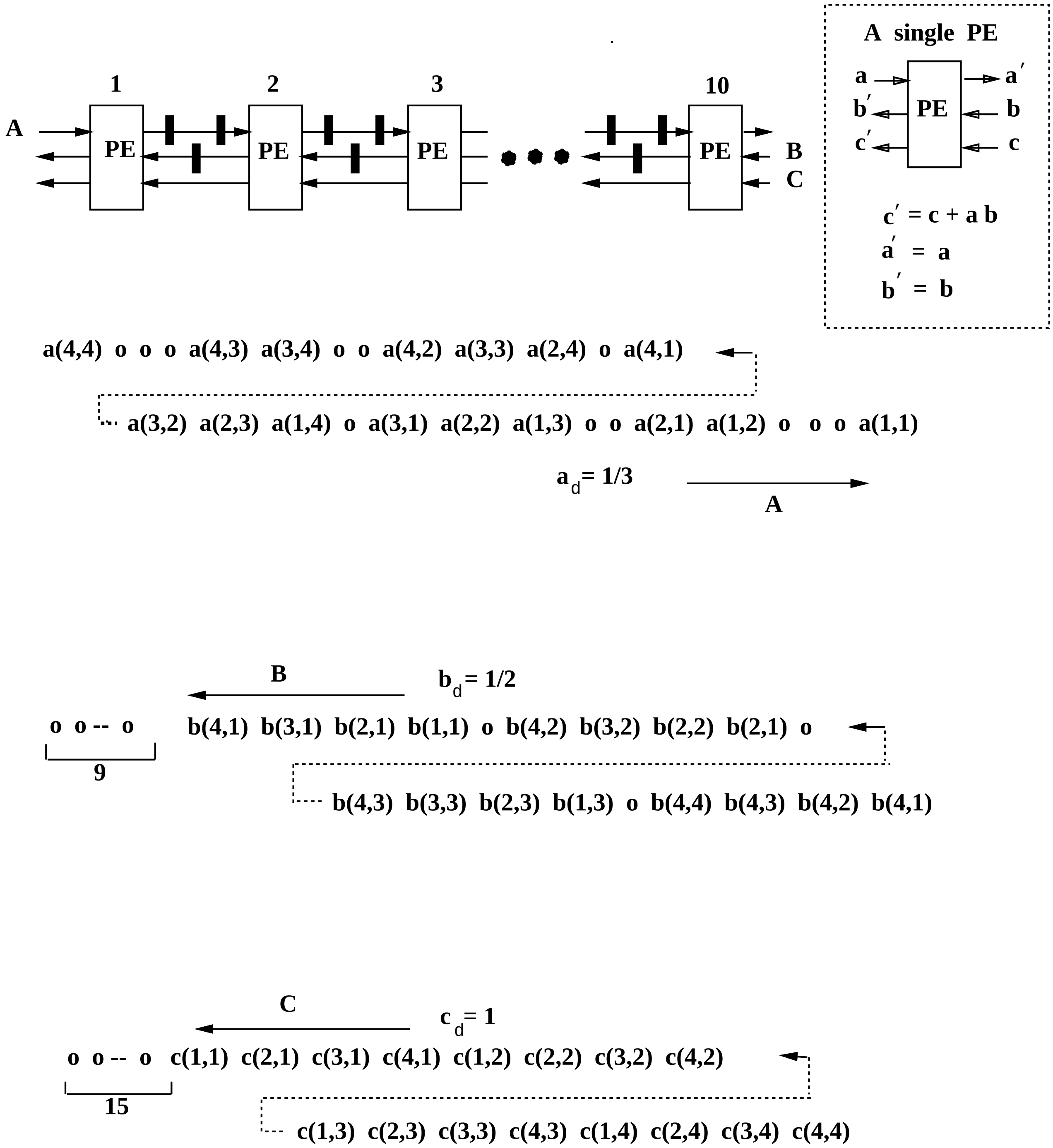

Figure 1.8 Linear array for multiplying two 4x4 matrices. The array is optimal for completion time, #PEs and $\#PE.T^2$ product

Size	Optimal Design								LK	H1
(N)	Periods			Distances					Design	Design
	t_k	t_i	t_j	k_1	k_2	k_3	T	#PE	Dev.	Dev.
3	1	2	2	1	1	1	11	7	0.00%	0.00%
4	1	2	3	1	1	1	19	10	0.00%	0.00%
5	1	2	3	1	1	2	25	17	16.00%	0.00%
10	2	3	4	1	1	3	82	46	32.93%	0.00%
17	3	3	5	1	2	4	177	113	72.32%	9.04%
25	3	4	6	2	7	1	313	193	107.35%	23.00%
40	3	4	9	2	1	8	625	430	162.24%	49.94%
50	4	5	8	3	2	7	834	589	205.64%	70.54%
100	6	7	10	5	4	9	2278	1783	343.33%	134.72%

Table 1.1 Matrix Multiplication : Optimal Linear array synthesis for completion time T. Dev. refers to the deviation from the optimal.

If the objective is to minimize the number of PEs required in the linear array then the following lemma characterizes the optimal design.

Lemma 5 *The set of parameters* $(t_k, t_i, t_j) = (1, 2, L-1)$ *and* $(k_1, k_2, k_3) = (1, 1, 1)$ *results in a linear array with the minimum number of PEs.*

Proof. The parameters k_1, k_2 and k_3 are defined as

$$k_1 = |t_k|\,|\vec{z_d}| \tag{1.85}$$
$$k_2 = |t_i|\,|\vec{y_d}| \tag{1.86}$$
$$k_3 = |t_j|\,|\vec{x_d}| \tag{1.87}$$

Therefore $k_i \geq 1$ and from lemma 2, the values $k_1 = k_2 = k_3 = 1$ gives the minimum #PE (3L - 2). So to prove the lemma, the above given parameter set must be shown to be feasible.

For given values of periods t_i and distances k_i, the magnitudes of the velocities are fixed (can be determined as k_i/t_i). However, each velocity can be in one of two directions (linear array) and there are 8 combinations of data flows for the 3 variables. All that needs to be shown is that one of the eight possibilities leads to a feasible solution. Consider the case when the velocities are

$$\vec{z_d} = 1, \quad \vec{y_d} = 1/2, \quad \vec{x_d} = \frac{-1}{L-1} \tag{1.88}$$

where negative sign refers to the flow along negative x-axis (assuming the array is along the x-axis).

The spacing parameters can be then be determined from the constraint equations as follows:

$$
\begin{aligned}
\vec{x_{is}} &= t_i(\vec{y_d} - \vec{x_d}) = \frac{L+1}{L-1} && (1.89)\\
\vec{x_{ks}} &= t_k(\vec{z_d} - \vec{x_d}) = \frac{L}{L-1} && (1.90)\\
\vec{y_{ks}} &= t_k(\vec{z_d} - \vec{y_d}) = \frac{1}{2} && (1.91)\\
\vec{y_{js}} &= t_j(\vec{x_d} - \vec{y_d}) = \frac{-(L+1)}{2} && (1.92)\\
\vec{z_{is}} &= t_i(\vec{y_d} - \vec{z_d}) = -1 && (1.93)\\
\vec{z_{js}} &= t_j(\vec{x_d} - \vec{z_d}) = -L && (1.94)
\end{aligned}
$$

To determine if there are data conflicts, lemma 3 can be applied to the spacings of each of the variables $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{Z}$. Consider the case of input $\mathcal{X}$. Since the denominators in the fractions of $\vec{x_{is}}$ and $\vec{x_{ks}}$ are identical only the numerators need to be considered. Since the GCD(L,L+1) = 1, $\frac{L}{GCD(L,L+1)} \geq L$ and $\frac{L+1}{GCD(L,L+1)} \geq L$. Therefore, by lemma 3 $\mathcal{X}$ is conflict-free. Similarly, for $\mathcal{Y}$ and $\mathcal{Z}$ since GCD(1, L+1) and GCD(1, L) are 1 and hence they are conflict-free too. So all the inputs are conflict-free and the parameter set is feasible.

■

The above lemma gives a closed form expression for the parameters if the number of PEs is to be minimized in the linear array to solve matrix multiplication (and other 3-D cube algorithms). Figure 1.8 shows the array for $L = 4$. Thus the design is optimal both in terms of number of PEs and the completion time. But for $N > 4$ (table 1.1), the designs for optimal completion time do not minimize the #PE.

If the objective is to design a array with minimum $\#PE.T^2$ product, then the parameters of the optimal design could be in general different from those for optimal completion time and #PE. The search strategy for minimizing $\#PE.T^2$ is to first find a design which minimizes the completion time. Let T_{min} and P_1 be the completion time and #PE of the design. Then, the search space (which is polynomial in L, the size of the problem) is searched for a design that minimizes the $PE.T^2$ product with completion time between T_{min} and $\frac{T_{min}\sqrt{P_1}}{\sqrt{3L-2}}$.

For $N = 4$, since Figure 1.8 minimizes both the completion time and #PE it also minimizes the $\#PE.T^2$ product (or any objective in the form of $\#PE^m.T^n$, $m, n \geq 1$).

1.4 Final Remarks

Hardware implementations of algorithms have a tremendous impact on real-time applications where speed is of utmost importance. Systematic techniques of generating these

<table>
<tr><th rowspan="2"></th><th rowspan="2">GPM</th><th colspan="2">DM</th></tr>
<tr><th>I</th><th>II</th></tr>
<tr><td>Type of Recurrence</td><td>Uniform</td><td>Uniform</td><td>Uniform</td></tr>
<tr><td>Algorithm
- Capability</td><td>N-dim $\rightarrow$ M-dim
$(M \leq N-1)$</td><td>$N \rightarrow N-1$
dim.</td><td>N-dim $\rightarrow$ M-dim
$(M \leq N-1)$</td></tr>
<tr><td>- Complexity</td><td>Polynomial in the size of the problem and the parameters chosen</td><td colspan="2">Integer linear programming to determine the matrix. Hence could be exponential in the worst case</td></tr>
<tr><td rowspan="2">Solution Characteristics</td><td>The target array has uniform data flows and does not require extra control bits to govern the data flows. So simple and modularly expandable design</td><td colspan="2">Solution generated might have irregular data flows. The method only generates the mapping and not the implementation. So control bits might have to be added to achieve the required data movement increasing the complexity of the design.</td></tr>
<tr><td>Able to generate Optimal solutions with respect to time, PE count or any objective function expressible in terms of the parameters.</td><td colspan="2">Only heuristic solutions due to high complexity (for instance the S matrix is heuristically chosen and π vector is then found).</td></tr>
</table>

Table 1.2 Comparison of the parameter method with dependency method

GPM - General Parameter Method
DM I - Dependency Method originally proposed by Moldovan
DM II - Dependency method extended by Lee & Kedem/Fortes to handle lower dimensional arrays

hardware arrays are crucial for the design of large systems involving a number of these arrays, which can be visualized as hardware libraries accessible from general purpose host computer.

In this chapter, a systematic parameter-based approach to generate systolic architectures for problems expressible as uniform recurrences has been described. Table 1.2 contrasts the parameter method with the dependency method. Lower dimensional arrays to solve the recurrence can be generated optimally using the parameter-based approach. However, the size of the target array is a function of the size of the index space defined by the recurrence. This leads to a different array for every problem size which is not desirable from a cost and reusability perspective. There are schemes available to partition and map the recurrence into smaller arrays [11] but these are not optimal. Hence, a number of issues in partitioning of algorithms and mapping them to arrays with fixed I/O or PEs are still to be explored.

The recurrences considered by researchers so far have been mostly uniform ones. It will be of significant interest and value to have systematic techniques to transform non-uniform recurrences (which can be used to represent a number of important problems in optimization) into hardware arrays.

Bibliography

[1] J. A. B. Fortes, K.-S. Fu, and B. W. Wah. Systematic design approaches for algorithmically specified systolic arrays. In V. M. Milutinovic, editor, *Computer Architecture: Concepts and Systems*, pages 454–494. North Holland, 1988.

[2] L. J. Guibas, H. T. Kung, and C. D. Thompson. Direct VLSI implementation of combinatorial algorithms. In *Proceedings of CALTECH Conference on VLSI*, pages 509–525, Jan. 1979.

[3] D. Heller. Partitioning big matrices for small systolic arrays. In S. Y. Kung, H. J. Whitehouse, and T. Kailath, editors, *VLSI and Modern Signal Processing*, pages 185–199. Prentice Hall Inc., 1985.

[4] R. M. Karp and M. Held. Finite state processes and dynamic programming. *SIAM Journal of Applied Mathematics*, 15:693–718, May 1967.

[5] R. M. Karp, R. E. Miller, and S. Winograd. The organization of computations for uniform recurrences. *Journal of the ACM*, 14:563–590, July 1967.

[6] H. T. Kung. Why systolic architectures ? *IEEE Computer*, 15:37–46, Jan. 1982.

[7] S. Y. Kung. *VLSI processor arrays.* Prentice Hall, 1988.

[8] P.-Z. Lee and Z. M. Kedem. Synthesizing linear array algorithms from nested for loop algorithms. *IEEE Trans. on Computers*, C-37(12):1578–1597, Dec. 1988.

[9] G.-J. Li and B. W. Wah. The design of optimal systolic arrays. *IEEE Trans. on Computers*, C-34(1):66–77, Jan. 1985.

[10] D. I. Moldovan. On the analysis and synthesis of VLSI algorithms. *IEEE Trans. on Computers*, C-31(11):1121–1126, Nov. 1982.

[11] D. I. Moldovan and J. A. B. Fortes. Partitioning and mapping algorithms into fixed size systolic arrays. *IEEE Trans. on Computers*, C-35(1):1–12, Jan. 1986.

[12] S. V. Rajopadhye. *Synthesis, optimization and verification of systolic architectures.* PhD thesis, University of Utah, Salt Lake city, Utah, 1986.