Correspondence

On the Relationship Between Two Systolic Array Design Methodologies

Matthew T. O'Keefe, Jose A. B. Fortes, and Benjamin W. Wah

Abstract—**The parameter method [1] and data dependency method [2] have been proposed as systematic design methodologies for systolic arrays. We describe the relationship between the two methodologies and show that the parameter method applies to a subclass of the algorithms that can be processed by the dependency method. The optimization procedure of the parameter method can be applied, in a restricted sense, within the framework of the dependency method. This procedure is used to derive an optimal array for the deconvolution algorithm which improves on a previously proposed design [1].**

Index Terms— **Constraint equations, deconvolution, dependency method, optimization, parameter method, space equa+tions, systolic arrays.**

I. Introduction

Systematic design methodologies have recently been proposed to overcome the limitations of heuristics and designer intuition and to provide a formal framework for mapping algorithms to systolic architectures [1]–[7], [11]–[13]. The goals are to reduce the design time and to determine mappings that are optimal in some selected sense. In this correspondence we show how the mathematical expressions in the parameter and dependency methods [1], [2] are related. In particular, optimal design procedures are discussed and an optimal design for deconvolution is derived as an improvement over the design proposed in [1].

The parameter and dependency methods are two seemingly distinct methods and, despite the fact that both use recurrence equations as the underlying algorithm models, explicit relationships between the two formalisms are not obvious. A procedure for the derivation of optimal systolic designs was first developed in the parameter method, and this method has been used by other researchers to derive and prove the optimality of actual designs [7]. The dependency method has been extensively studied, extended, and applied by other researchers [4], [6], [12]–[13]. The knowledge of relationships between these two methods can be used to apply extensions and improvements of the dependency method to the parameter method, besides other implications discussed in this correspondence. Section II provides a short description of both methodologies. Section III establishes equivalences between the mathematical expressions used to systematically design systolic arrays in the two methods. The equivalences of Section III are used in Section IV to develop optimization procedures. An improved systolic array for the deconvolution algorithm is also described in Section V. Section VI is dedicated to conclusions.

Manuscript received June 15, 1988; revised April 9, 1991. This work was supported in part by the National Science Foundation under Grants DMC-8419745 and MIP 88-10584 and in part by the Office of Naval Research under Contract 00 014-88-k-0723.

M. T. O'Keefe was with the School of Electrical Engineering, Purdue University, West Lafayette, IN 47907. He is now with the Department of Electrical Engineering, University of Minnesota, Minneapolis, MN 55455.

J. A. B. Fortes is with the School of Electrical Engineering, Purdue University, West Lafayette, IN 47907.

B. W. Wah is with the Department of Electrical and Computer Engineering and the Coordinated Science Laboratory, University of Illinois, Urbana, IL 61801.

IEEE Log Number 9200323.

II. Introduction to the Parameter and Data Dependency Methods

A. Parameter Method [1]

This methodology considers the design of optimal pure planar systolic arrays for a class of linear recurrences which take the general form

$$z_{i,j}^{k} = f\left[z_{i,j}^{k+\delta}, x_{\hat{x}(i,k)}, y_{\hat{y}(k,j)}\right], \qquad \delta = \pm 1 \tag{1}$$

where f is the function to be executed by each cell of the array and $\hat{x}(i,k)$, $\hat{y}(k,j)$ are linear indexing functions for the two-dimensional input variables $\boldsymbol{X}$ and $\boldsymbol{Y}$. In the following presentation, the coefficients of i, j, k are either 1 or -1. One-dimensional recurrences have the general form

$$z_{i}^{k} = f\left[z_{i}^{k+\delta}, x_{\hat{x}(i,k)}, a_{\hat{a}(k)}\right], \qquad \delta = \pm 1. \tag{2}$$

Three sets of parameters are used to characterize a systolic array: velocities of data flow, data distributions, and periods of computation. The velocity of a datum x is the directional distance passed by that datum in one clock cycle and is denoted by $\vec{x}_d$. The distance between two processing elements is defined to be one. Thus, $\vec{x}_d$ must be less than or equal to one because nonlocal communication is not allowed in pure systolic arrays. Data distributions are defined using row and column displacements. For two-dimensional input and output matrices, the elements along a row or column are arranged in a straight line and the distance between adjacent elements in a row or column remains constant as the data flows through the array. To define the row displacement of array $\boldsymbol{X}$, suppose that the row and column indexes of $\boldsymbol{X}$ are i and j, respectively. The row displacement of $\boldsymbol{X}$ is the directional distance between $x_{\hat{x}(i,j)}$ and $x_{\hat{x}(i+1,j)}$ and is written as $\vec{x}_{is}$. Similarly, the column displacement is the distance between $x_{\hat{x}(i,j)}$ and $x_{\hat{x}(i,j+1)}$ and is written as $\vec{x}_{js}$.

Periods of computation are described using two functions, τ_c and τ_a. τ_c is defined as the time at which a computation is performed, whereas τ_a defines the time at which a variable is accessed. The periods of i and j for two-dimensional outputs are defined as

$$t_i = \tau_c(z_{i+1,j}^{k}) - \tau_c(z_{i,j}^{k}) \tag{3}$$

$$t_j = \tau_c(z_{i,j+1}^{k}) - \tau_c(z_{i,j}^{k}) \tag{4}$$

$$t_k = \tau_c(z_{i,j}^{k+1}) - \tau_c(z_{i,j}^{k}). \tag{5}$$

It will be assumed that t_k is positive. If this is not true for a given recurrence, the recurrence can be rewritten to satisfy this condition. In computing $z_{i,j}^{k}$, $x_{\hat{x}(i,k)}$ and $y_{\hat{y}(k,j)}$ are accessed and two additional periods can be included to describe this interaction. They are

$$t_{kx} = \tau_a(x_{\hat{x}(i,k+1)}) - \tau_a(x_{\hat{x}(i,k)}) \tag{6}$$

$$t_{ky} = \tau_a(y_{\hat{y}(k+1,j)}) - \tau_a(y_{\hat{y}(k,j)}). \tag{7}$$

Depending on the order of access, t_{kx} and t_{ky} may be negative. Since operands to be used in a computation must arrive at a processing element simultaneously, the magnitude of the periods must equal t_k, i.e., it must be true that

$$t_k = |t_{kx}| = |t_{ky}|.$$

The periods are independent of the indexes i, j, and k, and they must be greater than or equal to one to prevent nonlocal communication. These parameters (velocity, data distribution, and periods) can be combined into a set of equations which describe the operations of a systolic array. These equations, for the two-dimensional case, are

$$t_{kx}\vec{x}_d + \vec{x}_{ks} = t_{kx}\vec{z}_d \tag{9}$$

$$t_{ky}\vec{y}_d + \vec{y}_{ks} = t_{ky}\vec{z}_d \tag{10}$$

$$t_i\vec{x}_d + \vec{x}_{is} = t_i\vec{y}_d \tag{11}$$

$$t_i\vec{z}_d + \vec{z}_{is} = t_i\vec{y}_d \tag{12}$$

$$t_j\vec{y}_d + \vec{y}_{js} = t_j\vec{x}_d \tag{13}$$

$$t_j\vec{z}_d + \vec{z}_{js} = t_j\vec{x}_d. \tag{14}$$

For the one-dimensional case, the last two equations are removed from the set of systolic equations given above.

The optimal systolic array for a given recurrence can be found by systematically enumerating the possible solutions using a search order that guarantees that the first feasible solution found is, in fact, the optimal one. Consider optimizing T, the total time needed to complete the computation. Let $|t_k||\vec{z}_d| = k_1$, $|t_i||\vec{y}_d| = k_2$, and $|t_j||\vec{x}_d| = k_3$. The constants k_i represent the distance between the PE where a variable is generated and the PE where it is used. First set $k_1 = k_2 = k_3 = 1$ [1] or, if a particular variable p is to remain in the same processing element, set the associated $k_p = 0$. Then, set the magnitudes of the periods t_i, t_j, and t_k equal to one and determine if a feasible solution exists. If a feasible solution is found then it is the optimal solution for T because T is a linear function of the periods that grows monotonically with increases in the magnitude of these periods. If no feasible solution is found with $t_i = t_j = t_k = 1$, then one of the periods is increased by one and the search for a feasible solution is repeated. If no feasible solution can be found with $k_1 = k_2 = k_3 = 1$, one of the $k_i, 1 \le i \le 3$, is increased by one and the search begins again. A flowchart describing this procedure is shown in [9]. A similar procedure is used to optimize the AT^2 measure [1], [9].

B. Data Dependency Method [2]

The essence of the data dependency method is the representation of the dependency structure of an algorithm in concise, matrix form. Let Z^n denote the nth Cartesian power of Z, the set of nonnegative integers. To describe the structure of an algorithm a tuple $A = (J^n, D)$ is used where $J^n \subset Z^n$ is the index set and D is a matrix whose columns are dependence vectors. The data dependencies describe the structure of the algorithm and are such that if the computation indexed by $\bar{j}$ requires the variable generated at index $\bar{j} - \bar{d}$ as an operand then $\bar{d}$ is a column of D.

Linear indexing functions [8] describe how variables are referenced. A linear indexing function $\bar{F} : J^n \to Z^m$ is defined by an equation of the form $\bar{F}(\bar{j}) = \bar{C}_0 + C\bar{j}$ where $\bar{C}_0 \in Z^{(m\times 1)}$ is called the index displacement and $C \in Z^{(m\times n)}$ is called the indexing matrix. For example, the variable $a(j_1 - j_2, j_2 + j_3 - 1, j_3 - j_1)$ has a linear indexing function for which

$$C = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & 1 \\ -1 & 0 & 1 \end{bmatrix} \text{ and } \bar{C}_o = \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix}.$$

A transformation matrix T can be used to describe a mapping which transforms the dependency matrix and index set of an algorithm so that it can be executed in a VLSI array. T can be partitioned into two matrices, π and S:

$$T = \begin{bmatrix} \pi \\ S \end{bmatrix}.$$

The π matrix defines the time transformation whereas S defines the space transformation to be applied to the dependence matrix and index set of an algorithm. The time at which a computation indexed by $\bar{j}$ is executed is determined by $\pi\bar{j}$, while $S\bar{j}$ specifies which processor is to execute this computation. Of course, π and S must satisfy certain conditions if they are to be considered valid transformations. Let m be the number of columns in the dependency matrix. Time transformations must satisfy $\pi\bar{d}_i > 0, i = 1, 2, \cdots, m$, where $\bar{d}_i$ is a column vector in the dependence matrix. This constraint results from the requirement that a variable must be generated before it is used in a computation. The time of execution of a computation with index $\bar{j}$ is given by

$$f_\pi(\bar{j}) = \left\lceil \frac{\pi\bar{j} - \min\{\pi\bar{j} : \bar{j} \in J^n\} + 1}{disp\pi} \right\rceil$$

where $disp\pi$, the displacement of the ordering determined by π, must satisfy $disp\pi \le \min(\pi\bar{d}_i : i = 1, \cdots, m)$. Intuitively, the displacement describes the number of parallel wavefronts that simultaneously sweep over the index set to complete the computation. In this paper, unless otherwise stated, the displacement is considered to be one, since the parameter method considers only this case. The space transformation S maps the computation indexed by $\bar{j}$ into processor $S\bar{j}$. This assumes a processor array model consisting of a grid which has the dimensionality of the array. Each point of the grid corresponds to a processor and the coordinates of the point are the index of the processor. Certain restrictions must be placed on possible solutions for S due to the limited interconnections available in VLSI arrays. These restrictions can be embodied in the P and K matrices. The P matrix describes the interconnection primitives available within an array, i.e., the vector differences between indexes of connected processors. The utilization matrix K describes the interconnections used by the transformed algorithm during execution. The relationship between K, P, S, and D is

$$SD = PK \tag{15}$$

where the entries of K must satisfy the following constraint

$$\sum_{j=1}^{r} k_{ji} \le \pi\bar{d}_i, \qquad i = 1, \cdots, m. \tag{16}$$

This last constraint requires that the time between the generation and use of a variable must be greater than or equal to the number of interconnection primitives needed by the datum to travel from the processing element in which it was generated to the processing element in which it will be used. Optimization procedures for the dependency method are given in [2].

III. Equivalences Between the Parameter and Dependency Methods

Lemmas 1–3 provide equivalences between the different parameters of the two methods, while Lemma 4 describes the form of

the dependency matrices for algorithms considered in the parameter method. These lemmas are then applied in Theorem 1 to show that the space equations and constraint equations are equivalent. Proofs are provided in [9]. The first lemma gives expressions for the data distribution and velocity vectors of the parameter method in terms of the transformations and indexing matrices of the data dependency method. Let S, π be as defined previously in Section II-B, and let v be any of the variables x, y, z as given for the parameter method. Also, let C^v represent the indexing matrix for variable v.

Lemma 1: The following relationships hold for the two-dimensional case:

$$S\begin{bmatrix} C^v \\ \pi \end{bmatrix}^{-1}\begin{bmatrix} \pm 1 \\ 0 \\ 0 \end{bmatrix} = \vec{v}_{is}.$$

$$S\begin{bmatrix} C^v \\ \pi \end{bmatrix}^{-1}\begin{bmatrix} 0 \\ \pm 1 \\ 0 \end{bmatrix} = \vec{v}_{js}.\ S\begin{bmatrix} C^v \\ \pi \end{bmatrix}^{-1}\begin{bmatrix} 0 \\ 0 \\ \pm 1 \end{bmatrix} = \vec{v}_d$$

and for the one-dimensional case, the following relationships apply:

$$S\begin{bmatrix} C^v \\ \pi \end{bmatrix}^{-1}\begin{bmatrix} \pm 1 \\ 0 \end{bmatrix} = \vec{v}_s.\ S\begin{bmatrix} C^v \\ \pi \end{bmatrix}^{-1}\begin{bmatrix} 0 \\ \pm 1 \end{bmatrix} = \vec{v}_d.$$

The next lemma describes the relationship between the π vector of the data dependency method and the periods t_i, t_j, t_k of the parameter method. The relationship is remarkably simple.

Lemma 2: The π vector of the data dependency method and the periods t_i, t_j, t_k of the parameter method are related by the equation

$$\pi = [t_i t_j t_k].$$

Thus, the periods of the parameter method are the elements of the π matrix. The next lemma relates the elements of the data dependency method's K matrix and the constants, $k_i (1 \le i \le 3)$, as defined in Section II-B; i.e., $|t_k||\vec{z}_d| = k_1$, $|t_i||\vec{y}_d| = k_2$, $|t_j||\vec{x}_d| = k_3$.

Lemma 3: Let $\hat{k}_i$ be the single nonzero entry of the ith column of K. Then the following equation holds:

$$\hat{k}_i = k_i \qquad 1 \le i \le 3.$$

The next lemma describes the form of the dependency matrices for the class of recurrences considered in the parameter method.

Lemma 4: The dependency matrices for the class of recurrences considered in the parameter method have the following structure:

Two-dimensional Recurrence:

$$D = \begin{bmatrix} \pm 1 & 0 & 0 & \\ 0 & \pm 1 & 0 & \bar{d}_1 \cdots \bar{d}_r \\ 0 & 0 & \pm 1 & \end{bmatrix}$$

One-dimensional Recurrence:

$$D = \begin{bmatrix} \pm 1 & 0 & c_1 & \\ & & & \bar{d}_1 \cdots \bar{d}_r \\ 0 & \pm 1 & c_2 & \end{bmatrix}, \quad |c_1| = |c_2| = 1$$

where $\bar{d}_1, \cdots, \bar{d}_r$ are dependency vectors which are a function of the recurrence, as is r, the total number of these additional dependencies. From this lemma it is apparent that the parameter method considers a subclass of the algorithms processed by the dependency method. This subclass of algorithms is characterized by a dependence matrix containing the identity matrix, possibly with negative diagonal entries, as a submatrix. The previous four lemmas can be used in a sequence a transformation steps which transform the constraint equations (9)–(14) of the parameter method into space equations in the dependency method (15). These space equations are a subset of those given in (15).

Theorem 1: The constraint equations (9)–(14) of the parameter method are equivalent to the space equations, $SD' = PK$, of the data dependency method, where D' corresponds to an identity matrix with diagonal entries that may be negative.

Hence, Theorem 1 shows that the constraint equations used in the parameter method are a subset of the space equations in the dependency method. However, using the equivalences given Lemmas 1–4 and the transformation steps used in the proof of Theorem 1, it is possible to add constraint equations corresponding to the additional dependency vectors $\bar{d}_1, \cdots, \bar{d}_r$ that are a function of the algorithm [9], [10].

IV. Optimization Procedures

Optimization procedures for the parameter method were discussed previously in Section II. By directly translating the parameters and constraints of this method into the corresponding elements of the dependency method, we can devise a similar procedure which is applicable to the class of algorithms considered in [1]. However, by using a slightly different approach, it is possible to propose a related optimization procedure applicable to all cases for which $disp\pi = 1$ in the dependency method. It differs from that proposed for the parameter method in that it checks all possible values of K before considering longer execution times (i.e., different π's). In the parameter method, a set of periods that minimize the completion time for a given set of constants k_i are sought. If none can be found, one of the k_i is increased by 1, and this process repeats. Both techniques are exhaustive enumerations of the search space. The flowchart of Fig. 1 describes the new optimization procedure. It starts by finding all transformations π which minimize execution time. This is relatively easy, since only the case $disp\pi = 1$ is considered and execution time is therefore a monotonic function of the entries of π. This condition holds even for those algorithms with dependence matrices that do not conform to the structure given in Lemma 4. One can start with all entries of π being zero and progressively increase their absolute values considering all possible combinations of signs and magnitudes (while, of course, checking for the validity of each π). Possible π's, which might result from further increases in the absolute value of the entries of a particular π, for which execution time is larger than the known minimum, need not be considered due to monotonicity property mentioned above. Thus, the search space is finite, and, in fact, rather small for most cases. Once the set of π's is known, it is necessary to check if there exists a solution to the equation $SD = PK$ for at least one of the possible values of K. If a solution is found, then the corresponding π (as well as the design determined by π and S) is optimal with respect to execution time. Otherwise, a new set of π's must be found which increase execution time by the least amount and the process is repeated again. The procedure always terminates, since, in the worst case, serial execution is reached as a feasible solution. A similar reasoning can be used to optimize measures combining area (i.e., number of processors) and execution time, e.g., AT or AT^2. Fig. 2 illustrates such a procedure. It differs from that of Fig. 1 in that the search space is reduced to the set of π's which result in execution time bounded above by a constant factor [9]. In this finite space, all valid values of π and S are considered and those which optimize the combined measure of area and time determine the optimal solution. This is exactly the same approach used in the parameter method. The key idea consists of limiting the search space by choosing bounds for π and, thus, for the execution time.

V. Systolic Design for Deconvolution

Deconvolution is the inverse of FIR filtering and can be expressed

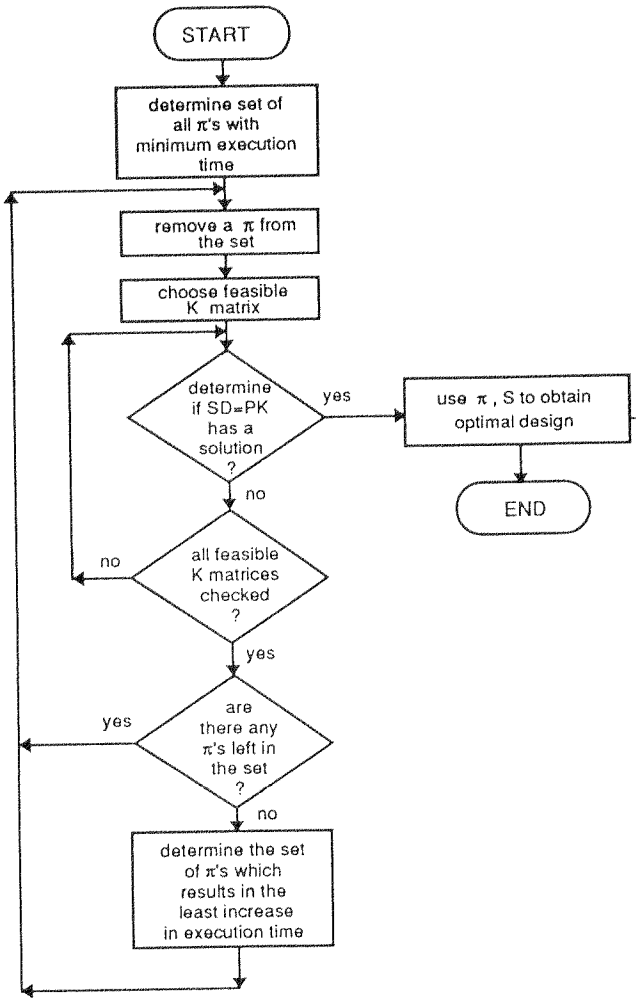

Fig. 1. Optimization procedure for obtaining a systolic array design with minimum execution time (T) using the dependency method.

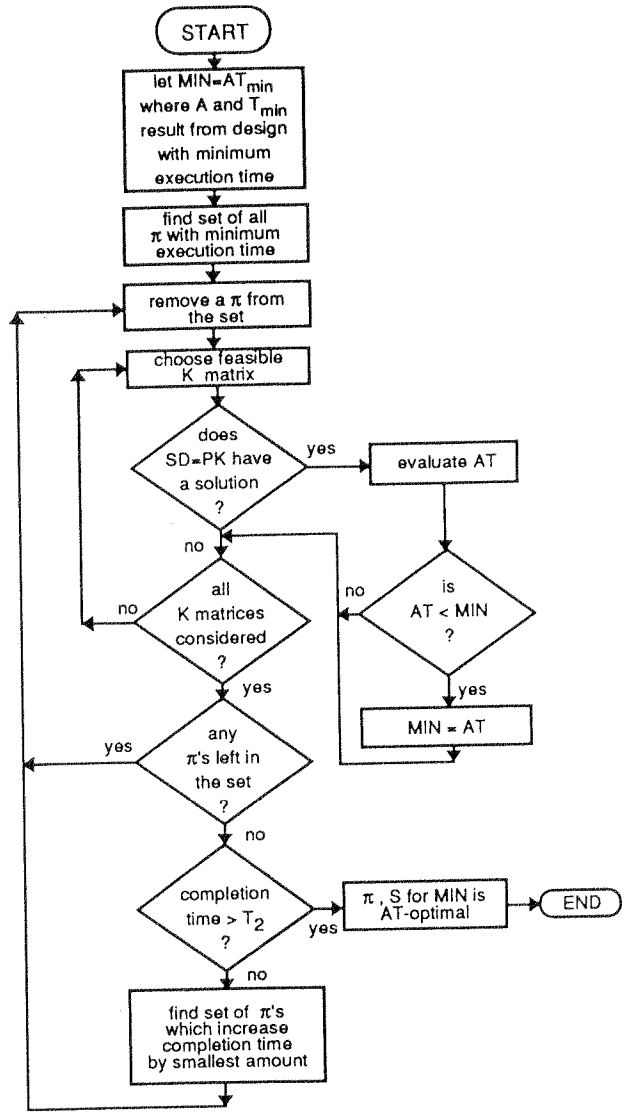

Fig. 2. Optimization procedure for obtaining a systolic array design with minimum area-execution time (AT) using the dependency method.

[1] as the following recurrence with temporary variable z_i

$$z_i^0 = y_i \qquad 1 \le i \le n$$
$$z_i^k = z_i^{k-1} - a_{m-k+1}x_{i+m-k}$$
$$1 \le k \le m-1, 1 \le i \le n, x_j = 0 \text{ for } j > n$$
$$x_i = \frac{z_i^{m-1}}{a_1} \qquad 1 \le i \le n.$$

Two different systolic arrays for deconvolution are now given. The first design was developed using the parameter method, and was first described in [1]. Another design has been developed with the dependency method which has a completion time $(1/2)m$ time units less than that given in [1]. This section provides an example of the use of each method to design an array for the same algorithm.

A. Design Using the Parameter Method

The parameter method was applied to develop a systolic array which performs deconvolution [1]. The array must perform division to obtain x_i, and x_i's are used in the computation of the z_i's. The division operation may take more time than multiplication, and this fact should be considered in the design process. Assume the delay of a division processing element is w and the delay of other processing elements is 1. This yields the equation $|t_i| = w + 1$. Analysis of the feedback condition of datum x_i yields an additional systolic equation

$$\vec{x}_d = w\vec{z}_d - \vec{z}_s.$$

These two equations can be included in the optimization; letting $w = 2$, $\vec{a}_d = 0$, $\vec{a}_s = -1$, $t_i = -3$, and observing that $t_{kx} = t_{ka}$, a possible solution to the constraint equations yields $t_k = -3/2$, $\vec{z}_d = 2/3$, $\vec{z}_s = 2$, $\vec{x}_d = -2/3$, and $\vec{x}_s = -2$. Note that the velocities of data flow have been averaged over three clocks cycles. A systolic array corresponding to these parameters, with $m = 4, n = 5$, is shown in Fig. 3(a).

B. Design Using the Dependency Method

The dependencies for the deconvolution algorithm consist of $[01]^T$ for the pair of variables (z_i^k, z_i^{k-1}), the buffering vector $[\pm 10]^T$ for the inputs a_{m-k+1}, and the dependence $[k - mk - m + 1]^T$ for the pair (z_i^k, z_{i+m-k^m-1}) which results from replacing x_{i+m-k} by its value z_{i+m-k}^{m-1} as defined in the recurrence. The latter dependency can be replaced by two constant dependencies, i.e., the buffering vector $[\pm 1 \pm 1]^T$, since x_{i+m-k} is used in more than one computation, and

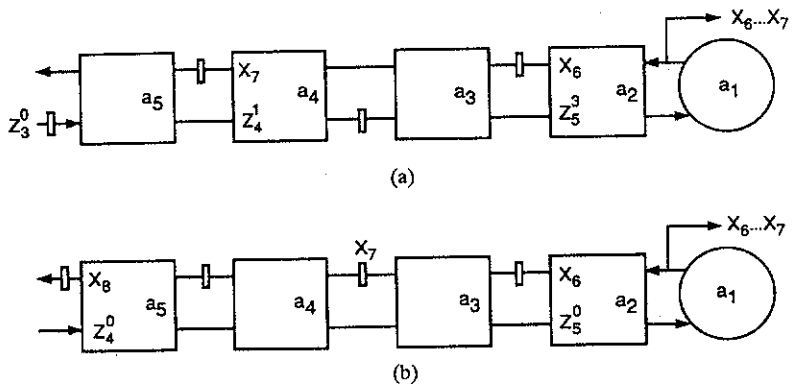

Fig. 3. Systolic arrays for deconvolution. (a) Parameter method. (b) Dependency method.

the dependence vector $[-10]^T$ which results from assigning the value $m-1$ to k in the dependence $[k-mk-m+1]^T$. In other words, x_{i+m-k} is generated according to the latter dependence $[-10]^T$, and then buffered according to the buffering vector $[\pm 1 \pm 1]^T$. The resulting dependency matrix is

$$D = \begin{bmatrix} \pm 1 & 0 & \pm 1 & -1 \\ 0 & 1 & \pm 1 & 0 \end{bmatrix}.$$

Since x_{i+m-k} is generated by a division and this operation is assumed to take w time units, π must be such that $w+1$ units of time elapse between generation and usage of this variable. In other words, we must have $\pi[-10]^T = w+1$ and with $w=2, \pi_1 = -3$. Using the proposed optimization procedure, $\pi_2 = 1$, the smallest possible value; possible S values include $S = [0\ 1]$, which optimizes space. This means that the generation and usage of z_i^{m-1} and x_i, respectively, occur in the same processing element. This array is optimal with respect to completion time T and AT^2. It could be developed using the parameter method, where, from Lemma 2, $t_i = -3$ and $t_k = 1$. A systolic array which conforms to this π and S for $m=4, n=5$, is shown in Fig. 3(b); the completion time of this array is $T = |\pi_1|(n-1) + |\pi_2|(m-1+w)$, which is $(1/2)m$ time units less than the array developed using the parameter method. Note that this optimal design can be found by using either of the methodologies. In the original design using the parameter method, an additional unnecessary constraint equation was included. Without this constraint, the solution given here would also have been obtained. This is a concrete example of the benefits resulting from the knowledge of the relations between methods derived in this paper. The deconvolution array of Fig. 3(a) has $\pi = [-3\ 3/2]$ and $S = [0\ 1]$. Lemmas 1–4 can be applied to show equivalence between the deconvolution arrays designed using the different methods.

VI. Conclusion

Explicit mathematical relations were established between the parameters, equations and constraints of two important methods for the design of systolic arrays. Despite the seemingly distinct formalisms used by the parameter and dependency methods, it is possible to define exact correspondences between them. We showed in Lemma 4 that the parameter method considers a subclass of the algorithms processed by the dependency method, and gave the form of the dependency matrix for these algorithms. It was also shown that the constraint equations can be transformed into a subset of the space equations; additional constraint equations may be added to represent additional dependence vectors. In a restricted sense, it is possible to use the optimization techniques of the parameter method in the dependency method framework. Since the completion of this work many extensions to the dependency method and related systolic array design methodologies have been developed. The relevant references are too many to include in this correspondence and the interested reader is referred to [14] and the references therein. Many of these extensions could be transferred to the parameter method using the relations described in this paper. In addition, the parameter method has been extended by McCanny and McWhirter to design bit level systolic arrays [7].

References

[1] G. J. Li and B. W. Wah, "The design of optimal systolic arrays," *IEEE Trans. Comput.*, vol. C-34, pp. 66–77, Jan. 1985.

[2] D. I. Moldovan and J. A. B. Fortes, "Partitioning and mapping algorithms into fixed size systolic arrays," *IEEE Trans. Comput.*, vol. C-35, pp. 1–12, no. 1, Jan. 1986.

[3] J. A. B. Fortes, B. W. Wah, and K. S. Fu, "Systematic approaches to the design of algorithmically specified systolic arrays," in *Proc. 1985 Int. Conf. Acoust., Speech, Signal Processing*, Tampa, FL, Mar. 1985.

[4] S. K. Rao, "Regular iterative algorithms and their implementation on processor arrays," Ph.D. dissertation, Stanford Univ., Stanford, CA, 1985.

[5] M. C. Chen, "Synthesizing VLSI architectures: Dynamic programming solver," in *Proc. 1986 Int. Conf. Parallel Processing*, Aug. 1986, pp. 776–784.

[6] S. Y. Kung, *VLSI Array Processors*. Englewood Cliffs, NJ: Prentice-Hall, 1987.

[7] J. V. McCanny and J. G. McWhirter, "The derivation and utilization of bit level systolic array architectures," in *Proc. 1986 Int. Workshop Systolic Arrays*, Oxford, England, July 1986, pp. 47–59.

[8] J. A. B. Fortes and D. I. Moldovan, "Data broadcasting in linearly scheduled array processors," in *Proc. 11th Annu. Int. Symp. Comput. Architecture*, June 1984.

[9] M. T. O'Keefe and J. A. B. Fortes, "A comparative study of two systematic design methodologies for systolic arrays," Masters Thesis, School of Electrical Engineering, Purdue Univ., May 1986.

[10] ——, "A comparative study of two systematic design methodologies for systolic arrays," in *Proc. 1986 Int. Conf. Parallel Processing*, Aug. 1986, pp. 672–675.

[11] C. Guerra and R. Melhem, "Synthesizing non-uniform systolic designs," in *Proc. 1986 Int. Conf. Parallel Processing*, Aug. 1986, pp. 765–772.

[12] J. Delosme and I. Ipsen, "Efficient systolic arrays for the solution of Toeplitz systems: An illustration of a methodology for the construction of systolic architectures in VLSI," in *Proc. 1986 Int. Workshop Systolic Arrays*, Oxford, England, July 1986, pp. 37–46.

[13] P. R. Cappello and K. Steiglitz, "Unifying VLSI array designs with geometric transformations," in *Proc. 1983 Int. Conf. Parallel Processing*, Aug. 1983, pp. 448–457.

[14] *Proc. Int. Conf. Application Specific Array Processors*, M. Valero, Y. Kung, T. Lang, and J. A. B. Fortes, Eds., Sept. 1991, IEEE Catalog Number 91–71633.