ON SYNTHESIZING SYSTOLIC ARRAYS from RECURRENCE EQUATIONS WITH LINEAR DEPENDENCIES

Sanjay V. Rajopadhye
Computer and Info. Sc. Dept.
University of Oregon
Eugene, Or 97403, USA

S. Purushothaman
Computer Science Dept.
Penn State University
State College Pa 16802, USA

Richard M. Fujimoto
Computer Science Dept.
University of Utah
Salt Lake City, Ut 84112, USA

Abstract

We present a technique for synthesizing systolic architectures from Recurrence Equations. A class of such equations (Recurrence Equations with Linear Dependencies) is defined and the problem of mapping such equations onto a two dimensional architecture is studied. We show that such a mapping is provided by means of a linear allocation and timing function. An important result is that under such a mapping the dependencies remain linear. After obtaining a two-dimensional architecture by applying such a mapping, a systolic array can be derived if the communication can be spatially and temporally localized. We show that a simple test consisting of finding the zeroes of a matrix is sufficient to determine whether this localization can be achieved by pipelining and give a construction that generates the array when such a pipelining is possible. The technique is illustrated by automatically deriving a well known systolic array for factoring a band matrix into lower and upper triangular factors[1].

1. INTRODUCTION

Systolic arrays are a class of architectures that have shown great promise in exploiting the parallelism available in VLSI (very large scale integrated circuits). They consist of regular and local (usually nearest-neighbor) interconnections of a large number of very simple identical cells (processing elements). They are typically used as special-purpose back-end processors for computation-intensive problems. A number of systolic architectures have been presented for solving various problems such as matrix multiplication, L-U decomposition of matrices, solving a set of equations, convolution, dynamic programming, etc [1, 7, 10, 11, 12]. In most of these cases, the designs have been developed largely in an *ad-hoc*, case by case manner. Recently there has been a great deal of effort focused on unifying theories for analyzing such circuits [3, 8, 16, 25] and techniques for their synthesis [2, 5, 6, 13, 14, 15, 17, 18, 20, 23]. In all these approaches, the idea is to analyze the program dependency graph and transform it to one that represents a systolic array. The problem of synthesis is thus a special case of the **graph-mapping** problem where the objective is to transform a given graph to an equivalent one that satisfies certain constraints. For systolic array synthesis there are two major constraints, namely **nearest-neighbor communication** and **constant-delay interconnections.**

The initial specification for the synthesis effort is typically a program consisting of a set of (say n) nested loops. The indices of each of the loops together with the range over which they vary, define an n-dimensional domain in Z^n (Z denotes the set of integers). The computation in the loop body is performed at every point p in this domain; the usage of variables within it defines the **dependencies**

[1]This work was performed when the first two authors were students at the University of Utah. Their work was supported by University of Utah Research Fellowships and that of the third author by an IBM faculty development grant.

of the point p. For example, if the body contains a statement of the form

```
a[i,j,k] := 2*b[i,j+2,k-1]
```

then the point $p = [i, j, k]^T$ depends on $q = [i, j+2, k-1]^T$. Such a nested loop program can be shown to be exactly equivalent to a **recurrence equation** defined on the same domain *D*. This is the notation that we shall use as an initial specification.

In most of the earlier work cited above (see [22] for a detailed survey) the underlying assumption is that the dependencies are uniquely characterized by a finite set, of **constant vectors** in Z^n as in the example above where p-q is a constant vector independent of p, namely $[0,-2,1]^T$. Other than the notation used most of these approaches are very similar[1]. For example, an equivalence between the approach of Li and Wah [15] and that of Moldovan [18] has been formally proved recently [19]. For such programs, the recurrence equations that describe them are called **Uniform Recurrence Equations** (UREs), and the dependency graph can be shown to be a **lattice** in Z^n. Under these restrictions the problem of synthesizing a systolic array can be solved by determining an appropriate **affine** transformation (i.e., one that can be expressed as a translation, rotation and scaling) of the original lattice.

We shall now briefly describe the previous approaches, using the notation of Uniform Recurrence Equations as discussed by Quinton [20]. It is assumed that the function *g* (corresponding to the computation in the loop body in other approaches) can be implemented on a single processor and can be computed in a single "time step"; *g* thus defines the granularity of the computation. The design of a systolic array then consists of scheduling the computation on all the points of the domain on an appropriate array of processors. This can be defined by means of a **timing function** that maps every point in the domain *D* to a positive integer, and an **allocation function** that maps every point in *D* to a (linear) array of processors. Quinton gives necessary and sufficient conditions for the existence of **affine** timing and allocation functions. He also presents a procedure for determining the timing function, in the case when the domain is a convex hull.

However, the class of problems expressible as uniform recurrence equations is restrictive and a large number of interesting problems cannot be naturally expressed as UREs. The chief reason for this is the restriction that all the dependency vectors must be constants, independently of the particular point in the domain. In addition, we have shown elsewhere [22] that many of the necessary and sufficient conditions for the existence of affine timing functions can always be satisfied if the computation is "well formed". Thus, UREs are too close to the final architecture to be a useful "high-level" specification.

In this paper we therefore, propose a more general class of recurrence equations called Recurrence Equations with Linear Dependence (RELDs). In RELDs, as the name suggests, the dependencies of a particular point are linear (actually affine) functions of the point. This paper addresses the problem of synthesizing systolic arrays from RELDs. As in the case of UREs, our approach is to determine appropriate timing and allocation functions for the recurrence equation. This defines a mapping of the original RELD into a **processor-time domain**, and thus yields an architecture for the problem. We shall prove that the new dependency structure induced by this mapping is also an RELD. Thus, unlike UREs the architecture that we obtain may have non-local interconnections. It is therefore necessary to **explicitly pipeline** the data flow in the new architecture. Explanation of this two-step process constitutes the principal thrust of this paper.

---

[1] In some cases, where the dependencies are not restricted to be constant vectors the method relies on user input or heuristics to determine appropriate transformations.

The rest of this paper is organized as follows. In the following section (Sec. 2) we formally define recurrence equations, UREs and RELDs and introduce some of the notation we shall use later. Then, in Sections 3 and 4 we discuss how, by selecting appropriate timing and allocation functions, a (naive) target architecture is automatically induced. We also discuss the conditions for the existence of such functions. Then, in Section 5 we motivate the necessity for explicit pipelining by showing that the architecture induced by the timing and allocation functions does not have systolic interconnections. We also define explicit pipelining and show that some simple tests on the dependency matrices are adequate to determine whether pipelining can be performed or not. Section 6 then summarizes these results by describing a complete synthesis procedure. The technique is illustrated in (Section 7) by synthesizing the systolic array for a well known example – LU-decomposition (i.e., factorizing a band matrix into lower and upper diagonal matrices).

2. RECURRENCE EQUATIONS WITH LINEAR DEPENDENCIES

Recurrence equations are a well known tool for expressing a large class of computations. The computation involves the evaluation of a function f at all points in a domain D. D is a subset of Euclidean n-space E^n. The recurrence equation specifies how the value of f at p depends on the value of f at other points in the domain. Based on these dependencies recurrence equations are classified as uniform or nonuniform, one- or multi- dimensional, etc.

Definition 1: A **Recurrence Equation** over a domain D is defined to be an equation of the form

$$f(p) = g(f(q_1), f(q_2) \ldots f(q_k))$$

where $p, \in D$; $\quad q_i \in D$, $i = 1 \ldots k$;

and g is a single valued function which is strictly dependent on each of it's arguments.

A **system** of m Recurrence Equations over a domain D is defined to be a family of m mutually recursive such equations, where each f_i is defined by an equation of the form above (with each f_i being, in general, a function of the all the f's).

Two simple examples are the well known factorial and fibonacci functions which are specified by the following equations.

$$f(n) = \begin{cases} 1 & \text{if } n \leq 0 \\ n * f(n-1) & \text{otherwise} \end{cases}$$

and

$$f(n) = \begin{cases} 1 & \text{if } n \leq 1 \\ f(n-1) + f(n-2) & \text{otherwise} \end{cases}$$

These functions are defined on the domain of natural numbers and can therefore be expressed as linear recurrence equations. Since the value of f at any point n depends on f at some other points which are at a **constant distance** from n they also belong to a class of recurrence equations called Uniform Recurrence Equations (UREs). This taxonomy was first proposed by Karp *et al* [9], and is formally defined as follows.

Definition 2: A Recurrence Equation $f(p) = g(f(q_1), f(q_2) \ldots f(q_k))$ as defined above, is called a **Uniform Recurrence Equation** (URE) iff

$q_i = p - w_i, i = 1 \ldots k,$

where w_i's are constant n-dimensional vectors;

As mentioned earlier, this paper proposes a more general class of computations. This class is formally defined as follows.

Definition 3: A **Recurrence Equation with Linear Dependencies** (RELD) is defined as an equation of the form

$$f(p) = g(f(A_1p + \bar{b}_1), f(A_2p + \bar{b}_2) \ldots f(A_kp + \bar{b}_k))$$

where

$p \in D$;

A_i's are constant n by n matrices;

$\bar{b}_i$'s are constant n-dimensional vectors;

and g is a single valued function which is strictly dependent on each of it's arguments.

As we have mentioned above, UREs are too close to a systolic architecture to serve as useful initial specifications. Many important problems cannot be cleanly described as UREs, and a great deal of effort has to be spent in "massaging" an initial problem specification into a URE. However, the class of problems defined by UREs is an important class because every physical systolic array can be expressed as a URE. To understand intuitively why this is so, consider a two dimensional systolic array. It has nearest neighbor interconnections and the links have a constant delay associated (both independently of location in the array). Thus if we imagine "snapshots" taken at every time instant as the computation progresses, we get a three-dimensional dependency structure in a space-time [x,y,t] domain. Any point p in this domain represents a computation that needs values from other points that are a **uniform distance away** independent of p and hence can be described by a URE.

Note that by restricting the dependency matrices A_j in an RELD to the identity matrices we obtain a URE. Thus UREs are merely a subset of RELDs, and one way of viewing some of the results presented here is as a formalization of the *ad hoc* "massaging" of the initial specification that is required in some other approaches [4, 6]. As an example of RELDs, consider the dynamic programming problem as applied to optimum parenthesization of a string. A systolic architecture for this has been described by Kung *et al* [7]. The problem is specified as follows. Given a string of n elements the minimum cost of parenthesizing substring i through j is given by the following.

$$c_{i,j} = \min_{i<k<j} (c_{i,k} + c_{k,j}) + w_{i,j} \quad \text{and} \quad c_{i,i+1} = w_{i,i+1}$$

As expressed above, this specification is clearly not even a recurrence equation (let alone a URE or a RELD) since the **number** of values $c_{x,y}$ that a particular $c_{i,j}$ depends upon is not constant but equal to j-i-1. However, by introducing an additional "accumulation index", and expressing the computation as an iteration we can obtain an RELD that performs the same computation as follows.

$c(1, n) = f(1, n, 1)$ where $f(i, j, k)$ is defined as

$$f(i, j, k) = \begin{cases} w_{i,j} & \text{if } j\text{-}i = 1 \\ w_{i,j} + \min \begin{pmatrix} f(i, j, k+1) \\ f(i, i+k, 1) + f(i+k, j, 1) \end{pmatrix} & \text{if } k = 1 \\ \infty & \text{if } 2*k > j\text{-}i \\ \min \begin{pmatrix} f(i, j, k+1) \\ f(i, i+k, 1) + f(i+k, j, 1) \end{pmatrix} & \text{otherwise} \end{cases}$$

Here, the value of f at (i,j,k) depends on its value at three other points, namely (i,j,k+1), (i,i+k,1) and (i+k,j,1). Thus the dependencies are given by

$$A_1 = \begin{bmatrix} 1, 0, 0 \\ 0, 1, 0 \\ 0, 0, 1 \end{bmatrix} \quad b_1 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}; \qquad A_2 = \begin{bmatrix} 1, 0, 0 \\ 1, 0, 1 \\ 0, 0, 0 \end{bmatrix} \quad b_2 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}; \qquad A_3 = \begin{bmatrix} 1, 0, 1 \\ 0, 1, 0 \\ 0, 0, 0 \end{bmatrix} \quad b_3) = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

3. TIMING FUNCTIONS FOR RELDs

We shall now investigate how the computation specified by an RELD over a domain D can be "scheduled." As mentioned earlier, this is done by defining a timing function t that maps every point in D to a positive integer. Accordingly t (p) is interpreted as the time instant at which f (p) is computed. The following statement is thus obvious from the definition of the dependency relation (and the fact that the relation induces a **partial order** on the points in the domain).

Remark 1: *A function t will be a timing function for a RELD iff*

(i) $\forall\, p \in D \quad t(p) > 0$

and (ii) $\forall\, p \in D \quad t(p) > t(A_j p + b_j)$

for $j = 1, 2 \ldots m$ that satisfy $A_j p + b_j \in D$

Note that we consider the **boundary** points as belonging to the domain, so the second condition is correctly restricted only to those points that explicitly depend on other points **in the domain**. We also have the following more restrictive case where we only have a sufficient condition.

Remark 2: *A function t will be a timing function for a RELD if*

(i) $\forall\, p \in D \quad t(p) > 0$

and (ii) $\forall\, p \in D \quad t(p) > t(A_j p + b_j)$

We have shown elsewhere [22] that a systolic architecture cannot be synthesized from an RELD that does not admit an **affine** timing function. Therefore, we shall henceforth restrict our attention merely to ATFs.

Definition 1: An affine timing functions is defined as

$$t(p) = \lambda_t^{\,T} p + \alpha$$

where

λ_t is a vector over the integers Z^n, and α is an integer constant.

3.1. Timing Functions for Convex Hulls

We also consider a special class of domains called **convex hulls**. Such domains are formed by the intersection in hyperspace of a number of half-hyperspaces. Each such hyperspace is defined by a bounding hyperplane, i.e., an inequality of the form

$$\sum_{i=1}^{k} u_i * x_i > c$$

It is well known [24] that such a domain can be described by a set of unique vertices V and a set of rays R that are unique up to scalar multiples. This means that if such a domain has m vertices (also called basis vectors), it can been shown that any point p in the domain can be uniquely expressed as the sum of a **positive combination**[1] of the rays and a **convex combination** of the vertices. Thus

$$p = \sum_{i=1}^{m} (c_i v_i) \text{ where } c_i \geq 0; \text{ and } \sum_{i=1}^{m} c_i = 1$$

We now present a theorem that gives us sufficient conditions for the existence of an affine timing function.

Theorem 1: *For an RELD defined by the dependencies* $[A_j, b_j]_{j=1, \ldots, k}$ *over a domain D defined by vertices* $V = \{v_1, \ldots, v_m\}$ *and rays* $R = \{r_1, \ldots, r_l\}$*; an affine function t* (p) *specified by* $[\lambda, \alpha]$ *is a valid timing function for the RELD if the following are satisfied*

i.	$\forall v_i \in V$	$\lambda^T v_i + \alpha > 0$	
ii.	$\forall r_i \in R$	$\lambda^T r_i \geq 0$	
iii.	$\forall v_i \in V$	$\lambda^T v_i > \lambda^T (A_j v_i + b_j)$	$j = 1 \ldots k$
iv.	$\forall r_i \in R$	$\lambda^T r_i > \lambda^T A_j r_i$	$j = 1 \ldots k$

Proof: We know that $[\lambda, \alpha]$ will be a valid ATF if it satisfies the two conditions of Remark . Also, it has been shown elsewhere (see [22] Lemma 2.1) that the first condition can be satisfied iff

	(i) $\forall v_i \in V$	$\lambda^T v_i + \alpha > 0$	
and	(ii) $\forall r_i \in R$	$\lambda^T r_i \geq 0$	

Hence, the result follows directly if we can demonstrate that

$$\forall p \in D \; t(p) > t(A_j p + b_j) \qquad \text{for } j = 1, \ldots, k$$

i.e., $\lambda^T p > \lambda^T (A_j p + b_j)$

Since D is a convex hull we know that any point p can be written as

$$p = \sum_{i=1}^{m} (c_i v_i) + \sum_{i=1}^{l} (a_i r_i) \qquad 1.1$$

where $c_i \geq 0$; $a_i \geq 0$; and $\sum_{i=1}^{m} c_i = 1$

Also since we know that $\lambda^T v_i > \lambda^T [A_j v_i + b_j]$ for i = 1, . . . m, we can multiply each such inequality by the c_i's above and add up, yielding the following.

$$\begin{aligned} \sum_{i=1}^{m} c_i \lambda^T v_i &> \sum_{i=1}^{m} c_i \lambda^T [A_j v_i + b_j] \\ &= \sum_{i=1}^{m} c_i \lambda^T A_j v_i + \sum_{i=1}^{m} c_i \lambda^T b_j \\ &= \lambda^T A_j \sum_{i=1}^{m} c_i v_i + \lambda^T b_j \sum_{i=1}^{m} c_i \end{aligned} \qquad 1.2$$

---

[1] If $x_1, x_2 \ldots x_n$ are points in R^n, then $\sum_{i=1}^{n} (a_i x_i)$ is said to be a **positive combination** if a_i are nonnegative real numbers. If in addition $\sum_{i=1}^{n} a_i = 1$ is true then it is called a **convex combination.**

Similarly, since we know that $\lambda^T r_i > \lambda^T A_j$ for i = 1, . . . l,

$$\sum_{i=1}^{l} a_i \lambda^T r_i \quad > \sum_{i=1}^{l} a_i \lambda^T [A_j r_i]$$

$$= \lambda^T A_j \sum_{i=1}^{l} a_i r_i \qquad 1.3$$

Adding up Equations 1.2 and 1.3 and substituting from Equation 1.1 we get

$$\lambda^T p > \lambda^T (A_j p + b_j)$$ ∎

Notice that this does not give us necessary conditions for existence of ATFs, merely sufficient conditions. It is also clear from the above theorem that conditions for determining affine timing functions for RELDs is not as straightforward as for UREs. To understand intuitively why this is so consider a URE defined over a convex hull with a ray *r*. For any two points p and p + μr the **distance** between the points themselves, is exactly equal to the distance between the points that they **depend on**, since p depends on p + w_i and p + μr depends on p + μr + w_i. Thus for such a URE to have an affine timing function, the condition that λ and α must satisfy is that $[\lambda^T (p + w_i) + \alpha] - [\lambda^T p + \mu r + \alpha]$ must be positive. Since the dependencies are uniform, this distance is independent of p. In the case of RELDs however, we must ensure that

$$\lambda^T p > \lambda^T [A_i p + b_i]$$

i.e., $$\lambda^T [(I - A_i)p + b_i] > 0$$

Clearly, this is not independent of p, and in a domain with a ray *r* we must ensure that it will be true for p + μr for arbitrarily large *r*. The fourth condition in the previous theorem ensures exactly this.

4. ALLOCATION FUNCTIONS

Allocation functions are a mapping of the problem domain *D* to a new (processor) domain D_a. Intuitively, an allocation function may be viewed as defining the processor a (p) in D_a at which the computations denoted by point p in *D* is performed. The processor domain D_a is restricted to be two-dimensional since we are dealing with systolic arrays (linear systolic arrays are merely a special case of this), and each processor is connected to a nearest neighbor according to a particular interconnection scheme. The interconnection scheme is one of two possible ones – to four immediate neighbors, corresponding to mesh arrays (and linear for the one-dimensional case); and to six neighbors, corresponding to hexagonal arrays. As in the earlier approaches, we restrict our attention to **linear allocation functions**, which can be defined as

$$a\,(p) = [x, y] = [\lambda^T{}_x p + \alpha_x, \lambda^T{}_y p + \alpha_y]$$

Such a function thus corresponds to a geometric projection of the original domain. By choosing the direction of projection to be along the ray of *D* (remember that *D* is restricted to have at most one ray) it is guaranteed that the projection will be finite. Also if *u* is a vector (representing the direction of projection) then the only additional condition required to ensure that it determines a valid allocation is that $\lambda^T u \neq 0$. This constraint ensures that the timing and allocation functions are **free of conflict** i.e.,

$$t\,(p) = t\,(q) \wedge a\,(p) = a\,(q) \Rightarrow p = q$$

This ensures that no two computations are scheduled on the same processor at the same time.

It is clear that the timing and allocation functions are not unique, and in many instances there are optimality criteria that guide the choice of these functions. One such criterion, in addition to those discussed by Quinton, is the locality of interconnections. The reason one does not find a

uniform treatment of this aspect in the literature is that it is still a matter of opinion in the research community whether an architecture with (say) eight nearest-neighbor interconnections is considered a systolic array. Some researchers view this as outside the scope of systolic arrays because the architecture does not tesselate on a plane, while others view any regular (i.e., repeated) interconnection as appropriate. Each of these viewpoints can be thought of as optimality constraints on the allocation function. The constraint is merely that the projection of each of the dependencies must be a "permissible" interconnection, with a higher cost (which may be infinite, depending on the view that is chosen) for non-tesselating interconnections.

We can view the timing and allocation functions as performing a transformation S of the original problem specification from an n-dimensional domain to a three-dimensional one. Also, by specifying that one of the axes is the "time axis" we have obtained a clear separation of two important facets of an architecture, namely space and time. Henceforth, we shall refer to this space as the [x,y,t] space. If $[\lambda, \alpha]$ represents the affine transformation induced by the timing function $[\lambda_t, \alpha_t]$ and the allocation function $\{[\lambda_t, \alpha_t, \lambda_t, \alpha_t]\}$ we can write $[\lambda, \alpha]$ as follows.

$$\lambda = \begin{bmatrix} \lambda^T_x \\ \lambda^T_y \\ \lambda^T_t \end{bmatrix}, \quad \text{and} \quad \alpha = \alpha = \begin{bmatrix} \alpha_x \\ \alpha_y \\ \alpha_t \end{bmatrix}$$

We see that the following remark is true.

Remark 1: *If λ is nonsingular then there is no conflict between the timing and allocation functions.*

This is so because in this case λ has an inverse, and thus corresponding to any point p in the [x,y,t] space (i.e., on any processor [x,y] at any instant t) there is a unique point (i.e., a unique computation) in the original RELD.

5. DEPENDENCIES IN THE SPACE-TIME DOMAIN - EXPLICIT PIPELINING

As mentioned above, the timing and allocation functions described in the previous two sections can be viewed as a transformation that maps the original RELD to a new dependency structure in the space-time domain. The following theorem defines the properties of this new dependency structure.

Theorem 2: *For any RELD defined by $[A_j, b_j]_{j=1..m}$ the new dependency structure induced by the timing and allocation functions described by $[\lambda, \alpha]$ is also an RELD if λ has an inverse λ^{-1}.*

Proof:

We denote by S the transformation induced by the timing and allocation functions $[\lambda, \alpha]$, i.e.,

$$\begin{bmatrix} x \\ y \\ t \end{bmatrix} = S(p) = \lambda p + \alpha$$

Since λ has an inverse, λ^{-1} the computation of f at any point p in the original domain can be expressed as a computation of another function f' at $[x,y,t]^T$ as follows

$$
\begin{aligned}
f[x,\ y,\ t] &= f(p) = f(S^{-1}[x,\ y,\ t]) \\
&= g(f(A_1S^{-1}[x,\ y,\ t] + \bar{b}_1),\ f(A_2S^{-1}[x,\ y,\ t] + \bar{b}_2), \\
&\quad \ldots f(A_kS^{-1}[x,\ y,\ t] + \bar{b}_k) \\
&\quad) \\
&= g(f(\lambda(A_1S^{-1}[x,\ y,\ t] + \bar{b}_1) + \alpha), \\
&\quad f(\lambda(A_2S^{-1}[x,\ y,\ t] + \bar{b}_2) + \alpha), \\
&\quad \ldots f(\lambda(A_kS^{-1}[x,\ y,\ t] + \bar{b}_k) + \alpha)) \\
&\quad)
\end{aligned}
$$

But since $p = S^{-1}[x,\ y,\ t] = \lambda^{-1}\{[x,\ y,\ t]^T - \alpha\}$, we have

$$
\begin{aligned}
\lambda(A_jS^{-1}[x,\ y,\ t] + \bar{b}_j) + \alpha &= \lambda(A_j\lambda^{-1}\{[x,\ y,\ t]^T - \alpha\} + \bar{b}_j) + \alpha \\
&= \lambda A_j\lambda^{-1}\begin{bmatrix} x \\ y \\ t \end{bmatrix} - \lambda(A_j\lambda^{-1}\alpha + \bar{b}_j) + \alpha
\end{aligned}
$$

Since $\lambda A_j\lambda^{-1}$ is a constant 3×3 matrix and $\lambda(A_j\lambda^{-1}\alpha + \bar{b}_j) + \alpha$ is a constant 3-vector this represents an RELD in the [x,y,t] space. ∎

Since the proof of this theorem is constructive, in we can use the above result to determine the dependencies in the new RELD. We also have the following corollary.

Corollary 3: *For any URE, the transformation induced by affine timing and allocation functions, leaves the dependency structure uniform if the transformation matrix λ has an inverse.*

Proof: Since a URE is an RELD with the dependency matrix A_j being the identity matrix **I**, the transformation yields a new RELD where the corresponding dependency is

$$\lambda A_j\lambda^{-1} = \lambda\ \mathbf{I}\ \lambda^{-1} = \mathbf{I}$$

∎

We thus see that the timing and allocation functions directly define a two dimensional processor architecture. However, this **naive** architecture is not necessarily systolic, since the communication is not local (in fact, it may not even be at a constant distance away). This motivates the second step of the synthesis procedure, namely explicitly pipelining the dependencies in this space-time domain. A dependency of a point $[x,\ y,\ t]^T$ in this domain indicates that at time instant t, the processor [x,y] will need the value that the processor [x',y'] computed at time instant t', where $[x',y',t']^T$ is $A'_j\ [x,y,t]^T + b'_j$. The following theorem describes the underlying idea, that permits us to restructure the dependencies in the RELD.

Theorem 4 Pipelining Theorem: *A particular dependency $[A_j,\ b_j]$ of an RELD in the [x,y,t] domain can be made uniform if the dependency matrix A_j has a nontrivial zero ρ.*

Proof: Consider an RELD defined on the same [x,y,t] domain as follows.

$$
\begin{aligned}
f(p) &= [f_1(p),\ f_2(p)] \\
&\text{where } f_1(p) = g\ (f_1(A_1p + b_1),\ f_1(A_2p + b_2)\ \ldots \\
&\qquad\qquad f_2(p + \rho)\ \ldots\ f_1(A_kp + b_k)) \\
&\text{and}\quad f_2(p) = f_2(p + \rho)
\end{aligned}
$$

If this RELD is restricted to have the same boundaries as the original one, then it also has the same dependency structure except that the j^{th} dependency is now **uniform**. For it to be computationally equivalent to the original one, the following must hold.

$$f_2(p + \rho) = f_1(A_jp + b_j) \qquad (1)$$

But, the computation of f at point $[p + \rho]$ yields the following.

$$f_1(p + \rho) = g\ (f_1(A_1(p + \rho) + b_1),\ f_1(A_2(p + \rho) + b_2)\ \ldots\ f_2(p + 2\ \rho)\ \ldots\ f_1(A_k(p\ \rho) + b_k))$$

and

$$f_2(p + \rho) = f_2(p + 2\ \rho)$$

And thus for equivalency with the original RELD

$$f_2(p + \rho) = f_2(p + 2\ \rho) = f_1(A_j(p + \rho) + b_j) \qquad (2)$$

Since this must be true, regardless of the functions f, f_1 and f_2 we have, from Equations (1) and (2)

$$A_j(p + \rho) + b_j = A_j p + b_j$$

i.e.

$$A_j\ \rho = \overline{0}$$

Thus for the new Uniform RE to be computationally equivalent to the original one ρ has to be a zero of the dependency matrix. ∎

We also have the following corollary, whose proof follows directly from the previous theorem.

Corollary 5: *If all the dependency matrices A_j have zeroes of the form $[a, b, -k]^T$, where k is a positive integer and* [a, b] *is one of* [0, 0] [±1, 0], [0, ±1] *or* [±1, ±1], *the result of making the dependencies uniform yields a structure with systolic interconnections.*

Proof: The proof is obvious, since it is the vector ρ that determines the point in the [x, y, t] space that any point depends on. If ρ has the form described above, then we see that the communication is local, both **temporally** and **spatially**, which is exactly what is required for a systolic implementation. ∎

6. OUTLINE OF THE SYNTHESIS PROCEDURE

Before we present a complete synthesis procedure we shall prove a theorem which demonstrates that the two steps involved, namely determination of appropriate timing and allocation functions, and explicit pipelining may be performed independently.

Theorem 6: *If the matrix λ determining the timing and allocation functions has an inverse λ^{-1} then there exists a one-to-one correspondence between every pipelining dependency ρ in the space-time domain, and an equivalent pipelining dependency ρ' in the original RELD.*

Proof: We know from Theorem 5 that for a timing-function-allocation-function transformation specified by $[\lambda, \alpha]$ the new dependency structure is also an RELD with $[\lambda A_j \lambda^{-1},\ \alpha - \lambda(A_j\lambda^{-1}\alpha + b_j)]_{j=1..k}$ as the new dependencies. The proof then follows directly from a well known result from linear algebra theory that for any singular matrix λ, ρ is a solution for the equation $\lambda A \lambda^{-1} x = 0$ if and only if $\lambda\rho$ is a solution to $Ax = 0$.

As a result we shall henceforth consider, without any loss of generality, that all the pipelining operations are performed on the original RELD. The **pipelining theorem** gives us an important criterion to determine whether a given RELD can be pipelined. Intuitively, this can be explained as follows. Consider a point p in D that depends on another point q given by $A_j p + b_j$. We assume (for the purpose of this paper) that the matrix A_j has rank n-1 and hence the solution space of $A_j x = 0$ is a straight line[1]. This line has ρ as a basis, and intersects the domain boundary at a point $p_\perp$ (which

[1]This can be generalized (see [21, 22]) to deal with dependency matrices with other ranks but is beyond the scope of this paper.

in general, is a function of p and *D*). The pipelining theorem states that this line is exactly the set of all points in *D* that depend on $[A_j p, b_j]$ and hence the **linear** dependency $[A_j, b_j]$ may be replaced by the **uniform** dependency ρ. However, this is possible only if the point q is also in the null space of A_j, i.e., it is on the line with slope ρ passing through p. This is expressed as the following.

Necessary Condition for Direct Pipelining: A linear dependency $[A_j, b_j]$ of an RELD can be directly pipelined if $A_j(A_j p + b_j - p_\perp)$ is zero (for all points p in *D*).

We also see that the basis vector ρ is unique up to sign. Thus, the sign of ρ is chosen such that the resulting basis yields a dependency that is consistent with the timing function, i.e., $\lambda_t \rho$ must be positive. The synthesis procedure is thus described as follows.

1. Specify an RELD for the problem.
2. Determine a valid ATF [λ, α] for the RELD. Initially choose the optimal (i.e., the "fastest") ATF.
3. Determine the null spaces of each of the dependencies of the RELD, and choose λ-consistent basis sets for each of them.
4. Test for the necessary condition for direct pipelining of each of the dependencies, and if it is satisfied, determine the URE for the problem (which will still have [λ, α] as a valid ATF).
5. Choose an allocation function and derive the final architecture. This step is exactly identical to corresponding step when the starting point was a URE, except that the allocation function should now map *all* the (now uniform) dependencies to neighboring processors (including the newly introduced pipelining dependencies).

7. APPLICATION OF THE TECHNIQUE TO LU DECOMPOSITION

We shall now illustrate the technique presented above by means of an example. Consider the problem of factorizing a band matrix into its lower and upper triangular parts (LU decomposition) as defined in Figure 7-1.

$$\begin{bmatrix} a_{11}, & a_{12}, & a_{13} & \cdots & a_{1n} \\ a_{21}, & a_{22}, & a_{23} & \cdots & a_{2n} \\ a_{31}, & a_{32}, & a_{33} & \cdots & a_{3n} \\ . & & & & \\ . & & & & \\ . & & & & \\ a_{n1}, & a_{n2}, & a_{n3} & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} 1, & 0, \ldots & & 0 \\ l_{21}, & 1, \ldots & & 0 \\ l_{31}, & l_{32}, & 1, \ldots & 0 \\ . & & & \\ . & & & \\ . & & & \\ l_{n1}, & l_{n2}, & l_{n3}, \ldots & 1 \end{bmatrix} * \begin{bmatrix} u_{11}, & u_{12}, & u_{13} & \cdots & u_{1n} \\ 0, & u_{22}, & u_{23} & \cdots & u_{2n} \\ 0, & 0, & u_{33} & \cdots & u_{3n} \\ . & & & & \\ . & & & & \\ . & & & & \\ 0, & 0, & 0, & \cdots & u_{nn} \end{bmatrix}$$

Figure 7-1: *LU Decomposition of a matrix*

As described by Kung and Leiserson [11] the natural recurrence that describes this computation is the following[1].

$$a(i, j, 0) = a_{ij}$$

$$a(i, j, k) = a(i, j, k-1) - l_{ik} u_{kj}$$

where

$$l_{ij} = \begin{cases} 0 & \text{if } i<j \\ 1 & \text{if } i=j \\ a(i, j, j-1)/u_{jj} & \text{if } i>j \end{cases}$$

and

$$u_{ij} = \begin{cases} 0 & \text{if } i>j \\ a(i, j, i-1) & \text{if } i \le j \end{cases}$$

[1] We have slightly altered the third subscript to have the initial values available at k=0 rather than k=1

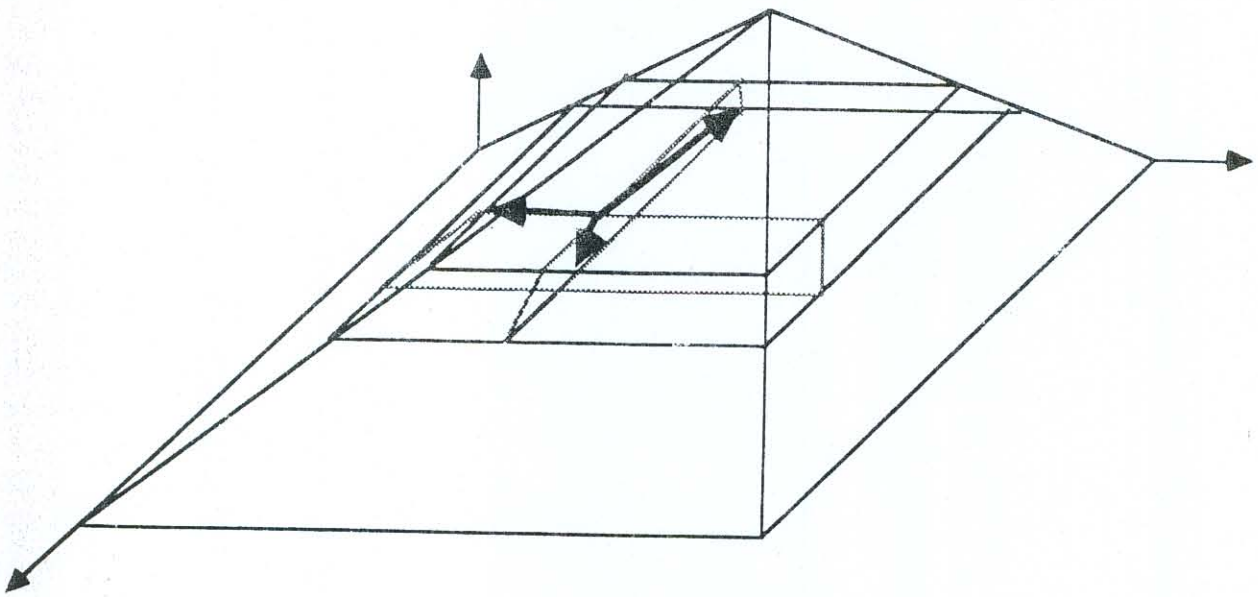
We now illustrate the different steps involved in synthesizing a systolic array for this problem.

7.1. Formulating an RELD for the problem

The domain of the above recurrences is the pyramid bounded by the points (1,1,0), (1,n,0), (n,1,0), (n,n,0) and (n,n,n). Its bounding planes are $k = 0$; $i = n$; $j = n$; $j = k$ and $i = k$. This expression is not an RELD because of the presence of the subscripted $l_{i,k}$ and $u_{k,j}$ terms. To express this as an RELD we can use one of two alternatives. First, by straightforward algebraic manipulation we can completely eliminate the $l_{i,j}$ and the $u_{i,j}$ terms from the three equations above. This yields the following RELD.

$$a(i, j, 0) = a_{i,j}$$

$$a(i, j, k) = a(i, j, k-1) - a(i, k, k-1) \cdot a(k, j, k-1)/a(k, k, k-1)$$



$$A_1 = \begin{bmatrix} 1, & 0, & 0 \\ 0, & 1, & 0 \\ 0, & 0, & 1 \end{bmatrix}, \quad b_1 = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}; \quad A_2 = \begin{bmatrix} 1, & 0, & 0 \\ 0, & 0, & 1 \\ 0, & 0, & 1 \end{bmatrix}, \quad b_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix};$$

$$\text{and } A_3 = \begin{bmatrix} 0, & 0, & 1 \\ 0, & 1, & 0 \\ 0, & 0, & 1 \end{bmatrix} \quad b_3 = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}$$

Figure 7-2: *Dependency Structure and Domain of the LU-Decomposition Recurrences*

Alternatively, we may view the function f computed at each point p = (i,j,k) in the domain as a tuple of two elements. The first of these is the value of a(i,j,k) and the second element is l(i,j,k), with

l being meaningful only at the j = k+1 boundary[1]. The RELD for this computation then becomes

$$f(i,\ j,\ k) = [a(i,\ j,\ k),\ l(i,\ j,\ k)]$$

where

$$a(i,\ j,\ 0) = a_{ij}$$
$$a(i,\ j,\ k) = a(i,\ j,\ k-1) - l(i,\ k,\ k-1)a(k,\ j,\ k-1)$$

and

$$l(i,\ j,\ k) = \text{if } j = k+1 \text{ then } a(i,\ j,\ k)/a(j,\ j,\ j-1)$$

We see that the second alternative is preferable since it does not involve any redundant computation of the l(i,j,k) values. However, it contains what appears to be a cyclic dependency, since the value of (a part of) f(i, j, k) (the l(i, j, k) part) depends on f(i, j, k) (actually, only its a(i, j, k) part). Hence, *as expressed above* this RELD cannot have a timing function. However, we can easily modify it by extending the domain to also include the j = k plane, and letting f(i, j, k) on this plane being l(i, j, k). This yields the following RELD

$$f(i,\ j,\ k) = \begin{cases} a_{ij} & \text{if } k = 0 \\ f(i,j,k-1)\,/\,f(k,j,k-1) & \text{if } k = j \\ f(i,j,k-1) - f(i,k,k)*f(k,j,k-1) & \text{otherwise} \end{cases}$$

This is the RELD that we shall use as the starting point of the mapping procedure. It must be emphasized that this choice of the initial RELD is not a limitation of the technique. In fact, by applying the mapping procedure to the first RELD we obtain an architecture which is very similar to the Kung-Leiserson array except that each processor performs a division (as expected) and depends on four input values. Figure 7-2 presents a pictorial view of the domain and the individual dependencies, and also the values of the A_j matrices and the b_j vectors.

7.2. Determining the allocation and timing functions

In order to synthesize a systolic array from this RELD we must first determine timing and allocation functions for it. Let $[\lambda^T_t \mid \alpha] = [a,\ b,\ c \mid d]$ be an affine timing function. Then it must satisfy the following constraints because of the dependencies of the RELD.

```
ai + bj + ck > ai + bj + c(k-1)      i.e. c > 0
ai + bj + ck > ai + bk + c(k-1)      i.e. b(j-k) + c > 0
ai + bj + ck > ak + bj + c(k-1)      i.e. a(i-k) + c > 0
```

Since our domain is a convex hull we can use Theorem 1 to determine the conditions for the existence of an affine timing function. This yields a set oof inequalities that determie the space of all possible timing functions, and we can choose the following as our timing function.

$$t(i,j,k) = i+j+k$$

Since the allocation function a (i,j,k) must not in conflict with the timing function, we can view a (i,j,k) as a projection of the original domain, that is *non-parallel* to the timing function. We therefore choose the following allocation function.

$$a(i,j,k) = [i-k,\ j-k]$$

It is easy to see that this choice of allocation and timing functions are free of conflict as follows. Let [i,j,k] and [p,q,r] be two points that map onto the same point in the [x,y,t] domain. Then

$$i + j + k = p + q + r$$
$$i - k = p - r$$

and

$$j - k = q - r$$

Hence

$$i + j - 2k = p + q - 2r$$

---

[1]Strictly speaking, f(p) should be a triple [a(p), l(p), u(p)], but the third element, u(i,j,k) is exactly equal to the corresponding a(i,j,k) and we may therefore ignore it

Subtracting this from the first equation yields 3k = 3r; i.e., k=r. Substituting this in the second and third equations yields i = p and j = q. Thus the two points are identical. We thus have

$$\lambda = \begin{bmatrix} 1, & 0, & -1 \\ 0, & 1, & -1 \\ 1, & 1, & 1 \end{bmatrix}, \text{ and hence } \quad \lambda^{-1} = \frac{1}{3}\begin{bmatrix} 1, & 2, & -1 \\ 1, & -1, & 2 \\ 1, & -1, & -1 \end{bmatrix} \quad \text{Also, } \alpha = \bar{0}$$

7.3. Pipelining in the processor-time domain

We can then use Theorem 5 to determine the new dependencies in the processor-time domain as follows.

$$A_1' = \lambda A_1 \lambda^{-1} = \begin{bmatrix} 1, & 0, & 0 \\ 0, & 1, & 0 \\ 0, & 0, & 1 \end{bmatrix} \text{ and since } \alpha = \bar{0}, \; b_1' = \lambda b_1 = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}$$

Similarly

$$A_2' = \begin{bmatrix} 1, & 0, & 0 \\ 0, & 0, & 0 \\ 0, & -1, & 1 \end{bmatrix}, \quad b_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad \text{and} \quad A_3' = \begin{bmatrix} 0, & 0, & 0 \\ 0, & 1, & 0 \\ -1, & 0, & 1 \end{bmatrix}, \quad b_3' = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix};$$

We see that although the first dependency has remained uniform under this transformation, neither A_2' nor A_3' has been reduced to the identity matrix. However, since both $|A_2'|$ and $|A_3'|$ are zero we can apply Theorem 4 in order to pipeline in this new structure. this requires us to solve

$$A_2' x = 0 \quad \text{(and correspondingly } A_3' x = 0\text{)}$$

and yields $[-k, 0, -k]^T$ (and $[0, -k, -k]^T$) respectively as a solution. We see that by choosing k to be 1, both the new dependencies $[-1, 0 -1]^T$ and $[0, -1 -1]^T$ are λ-consistent any processor [x, y] can obtain the required values from processors [x-1, y] and [x, y-1] over links of unit delays. Using this pipelining structure yields the architecture shown in Fig 7-3 below, which is identical to the one described by Kung and Leiserson.

8. CONCLUSIONS

We have presented a technique for designing systolic arrays from an initial specification which is in the form of a Recurrence Equation with Linear Dependencies. The class of problems that our technique can analyze is a superset of that covered by earlier (problems that had a **constant** dependency structure). For such a generalized class of *initial specifications* we have shown that appropriate choice of affine timing and allocation functions yields a target architecture. However, we have also shown that this architecture does not have nearest-neighbor interconnections and constant delays, and is thus not systolic. We have presented a technique called **explicit pipelining** by which the dependencies can be made uniform. In this paper, where the primary intention was to introduce RELDs and the idea of explicit pipelining, we have considered a somewhat restrictive form of pipelining, namely **direct pipelining**. In [22, 21] these ideas are presented in greater detail and we discuss techniques for synthesizing systolic arrays which have *time-dependent* data flow that is governed by *control signals* in the array.

Recently (in [5, 4]) Chen has presented an inductive technique to derive systolic architectures from what are defined as First Order Recursion Equations (FOREQs). We can show that these are merely a subset of Uniform Recurrence Equations with additional constraints specifying that the dependencies must be *local* in addition to being constant. Thus the class of problems that can be designed is restrictive, and most of the effort is spent in "massaging" the original problem specification into a FOREQ. Chen (in [4]) has presented a new architecture for LU-Decomposition, which is one and a half times faster than the one designed by Kung and Leiserson. It is our conjecture that merely by an appropriate choice of timing and allocation functions we should be able to derive this architecture as well.

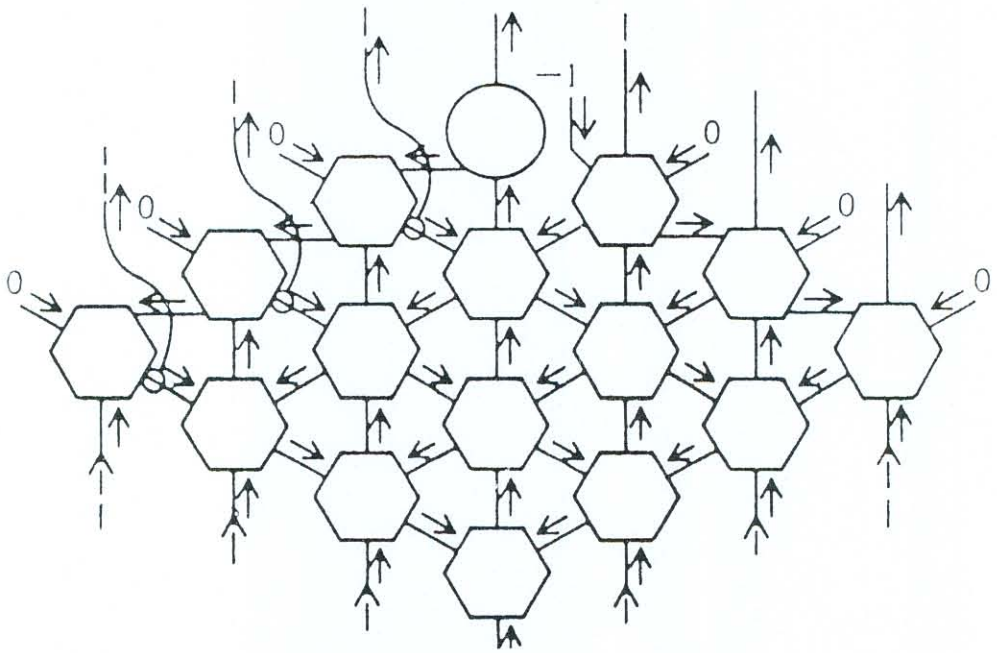


Figure 7-3: *Derived Architecture for LU Decomposition*

References

1. Brent, R. P. and Kung, H. T. Systolic VLSI Arrays for Linear-Time GCD Computation. VLSI 83, August, 1983, pp. 145-154.

2. Cappello, P. R. and Steiglitz, K. "Unifying VLSI Designs with Linear Transformations of Space-Time". *Advances in Computing Research* (1984), 23-65.

3. Chen, M. C. *Space-Time Algorithms: Semantics and Methodology.* Ph.D. Th., California Institute of Technology, Pasadena, Ca, May 1983.

4. Chen, M. C. Synthesizing systolic designs. YALEU/Dept. Of Computer Science/RR-374, Yale University, March, 1985.

5. Chen, M. C. A Parallel Language and its Compilation to Multiprocessor Machines or VLSI. Principles of Programming Languages, ACM, 1986.

6. Delosme, J. M. and Ipsen I. C. F. An illustration of a methodology for the construction of efficient systolic architectures in VLSI. International Symposium on VLSI Technology, Systems and Applications, Taipei, Taiwan, 1985, pp. 268-273.

7. Guibas, L., Kung, H. T. and Thompson, C. D. Direct VLSI Implementation of Combinatorial Algorithms. Proc. Conference on Very Large Scale Integration: Architecture, Design and Fabrication, January, 1979, pp. 509-525.

8. Johnsson, S. L., Weiser, U. C., Cohen, D. and Davis, A. L. Towards a Formal Treatment of VLSI Arrays. Proceedings of the second Caltech Conference on VLSI, January, 1981, pp. 375-398.

9. Karp, R. M., Miller, R. E. and Winograd, S. "The Organization of Computations for Uniform Recurrence Equations". *JACM 14*, 3 (July 1967), 563-590.

10. Kung, H. T. Let's design algorithms for VLSI. Proc. Caltech Conference on VLSI, January, 1979.

11. Kung, H. T. and Leiserson, C. E. Algorithms for VLSI Processor Arrays. In Mead, C. and Conway, L., Ed., *Introduction to VLSI Systems*, Addison-Wesley, Reading, Ma, 1980, Chap. 8.3, pp. 271-292.

12. Kung, H. T. "Why Systolic Architectures". *Computer 15*, 1 (January 1982), 37-46.

13. Lam, M. S. and Mostow, J. A. "A Transformational Model of VLSI Systolic Design". *IEEE Computer 18* (February 1985), 42-52.

14. Leiserson, C. E. and Saxe, J. B. "Optimizing Synchronous Systems". *Journal of VLSI and Computer Systems 1* (1983), 41-68.

15. Li, G. J. and Wah, B. W. "Design of Optimal Systolic Arrays". *IEEE Transactions on Computers C-35*, 1 (1985), 66-77.

16. Melhem, R. G. and Rheinboldt, Werner C. "A Mathematical Model for the Verification of Systolic Networks". *SIAM Journal of Computing 13*, 3 (August 1984), 541-565.

17. Miranker, W. L. and Winkler, A. "Space-Time Representation of Computational Structures". *Computing 32* (1984), 93-114.

18. Moldovan, D. I. "On the Design of Algorithms for VLSI Systolic Arrays". *Proceedings of the IEEE 71*, 1 (January 1983), 113-120.

19. O'Keefe, M. T. and Fortes, J. A. B. A Comparative Study of two Systematic Design Methodologies for Systolic Arrays. International Conference on Parallel Processing, IEEE, St. Charles, Ill, August, 1986.

20. Quinton, P. The Systematic Design of Systolic Arrays. 216, Institut National de Recherche en Informatique et en Automatique [INRIA], July 1983.

21. Rajopadhye, S. V. and Fujimoto, R. M. Systolic Array Synthesis by Static Analysis of Program Dependencies. UUCS-86-0011, University of Utah, Computer Science Department, August, 1986. Submitted for Publication.

22. Rajopadhye, S. V. *Synthesis, Otimization and Verification of Systolic Architectures.* Ph.D. Th., University of Utah, Salt Lake City, Utah 84112, September 1986.

23. Ramakrishnan, I. V., Fussell, D. S. and Silberschatz, A. "Mapping Homogeneous Graphs on Linear Arrays". *IEEE Transactions on Computers C-35* (March 1985), 189-209.

24. Rockafellar, R. T.. *Convex Analysis.* Princeton University Press, 1970.

25. Weiser, U. C. and Davis, A. L. A Wavefront Notational Tool for VLSI Array Design. VLSI Systems and Computations, Carnegie Mellon University, October, 1981, pp. 226-234.