On Synthesizing Systolic Arrays from Recurrence Equations with Linear Dependencies

UUCS-86-009

Sanjay V. Rajopadhye,
S.Purushothaman,
Richard Fujimoto

Department of Computer Science
University of Utah
Salt Lake City, Ut 84112

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 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.

This work is supported in part by University of Utah Research Fellowships and in part by an IBM faculty development grant.

is clear that a URE defines a dependency graph for the computation. It is assumed that the function g 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 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 such timing and allocation functions. He also presents a constructive proof for determining the timing function, under the restriction that 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 (q_i 's) must be constants, irrespective of the particular point in the domain. 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 a potential 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. We must therefore *explicitly* pipeline the data flow in the new architecture. Explanation of this two-step process constitutes the principal thrust of this paper. The rest of this paper is organized as follows. In the following section (Sec II) we formally define RELDs and introduce some of the notation we shall be using later. We then discuss (in Sec III) the notion of first reorganizing the dependency graph by syntactic restructuring and then introducing pipelining to obtain local communication. This two-step technique is illustrated in the following section (Sec IV) by synthesizing the systolic array for a well known example -- LU-decomposition (i.e. factorizing a band matrix into lower and upper diagonal matrices).

II Recurrence Equations with Linear Dependence

Definition: A Recurrence Equation with Linear Dependence (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 many important problems cannot be easily described as UREs, asnd 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 the p.

Note that if in an RELD the A_i's are identity matrices this becomes a URE. Thus UREs are merely a subset of RELDs. Thus one way of viewing (a part of) the results presented here is a formalization of the *ad hoc* "massaging" of the initial specification that other researchers do [4, 7, 8, 10]. As an example of RELDs, consider the dynamic programming problem as applied to optimum parenthesization of a string. This problem was discussed by Kung *et. al.* [9] who have described a systolic architecture for it. The problem involves the computation of a cost function specified as follows.

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

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 parameter, and expressing the computation as an iteration as follows, we can obtain an RELD that performs the same computation.

Example III.1

$$c(i,j) = f(i,j,1)$$

where

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

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

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

A. Notation

An RELD as defined above is viewed as defining a dependency graph. The nodes in the graph are the points in D and the arcs are given by the A_i and $\dot{b}_i$'s. We shall now introduce some terminology for such graphs. A *path* $\pi = (p_1, p_2 \ldots)$ is a sequence of nodes such that for each i, $p_{i+1} = A_j\, p_i + \dot{b}_j$ for some j. If the sequence π is finite, having $t+1$ nodes then we say that the path has *length* t and denote it by $l(\pi) = t$. If π is infinite, we say that $l(\pi) = \infty$. A path whose length is finite is called a *cycle* if the first and last nodes in it are identical. If all the nodes in a cycle π are distinct then π is a called a *simple cycle*.

Our objective in the synthesis problem is to "reorganize" this graph into an alternate configuration that preserves the output functionality of the RELD, and which corresponds to a systolic array (i.e. one which is at most two dimensional and has nearest-neighbor interconnections). To do this we must examine in precedence relations between the evaluation of f at various points $p \in D$. We say that point p *depends directly* on point q, denoted by $p \xrightarrow{1} q$ if and only if $p \in D$ and $q = A_i\, p + \dot{b}_i$ for some i. Thus $p \xrightarrow{1} q$ if and only if $f(q)$ is one of the arguments in $f(p)$. Now, *t-step dependence* is defined inductively as follows: $q \xrightarrow{0} q\ \forall\ q$; and $p \xrightarrow{t} q$ if there exists r such that $p \xrightarrow{t-1} r$ and $r \xrightarrow{1} q$. Also, we say that $p \rightarrow q$ if $p \xrightarrow{t} q$ for some positive integer t.

III Outline of the synthesis technique

We now introduce the notion of timing and allocation functions for the RELD. A *timing function t* is a mapping of all points p in D to the positive integers such that if $p \rightarrow q$ then $t(p) > t(q)$. This means that no computation can be performed until its arguments have been computed. $t(p)$ may naturally be interpreted as the time at which $f(p)$ is computed, with the assumption that the evaluation of the function g requires unit time. It thus serves as a schedule for the computations defined by the RELD. An *allocation function a* is a mapping of all points p in D to the domain $I \times I$ of a two dimensional mesh (note that a linear array is merely a special case of this, and a hexagonal array can be represented as a two dimensional mesh with diagonal interconnections).

Synthesizing a systolic array from an RELD can be viewed as a two-step process. Once we have a timing and an allocation function, we have obtained a planar architecture. However, the communication in such an architecture is in general, neither spatially nor temporally local. Thus the next step is to localize the communication by pipelining the data flow. These two issues are addressed in the next two subsections.

A. Step I: Timing and Allocation Functions

As defined above, the timing function $t(p)$ is interpreted as the time instant at which $f(p)$ is computed. The following statement is thus obvious from the inductive definition of the dependency relation "$\rightarrow$".

$t(p)$ will be a timing function for a RELD iff

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

and (ii) $\forall\, p \in D \quad t(p) > t(A_j p + \dot{b}_j)$ for $j = 1, 2 \ldots m$ that satisfy $A_j p + \dot{b}_j \in D$

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.
$t(p)$ will be a timing function for a RELD if

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

and (ii) $\forall\, p \in D \quad t(p) > t(A_j p + \dot{b}_j)$ for $j = 1, 2 \ldots m$

In the following, we shall restrict our attention to what are called *affine* timing functions (hereafter referred to as ATFs). Such a function is a scalar function of the form[1]

$$t(p) = \lambda^T{}_t p + \alpha_t$$

and is specified by a pair $[\lambda_t, \alpha_t]$. Here λ_t is a constant vector and α_t is a scalar constant. Intuitively the reason for restricting our attention to linear timing functions is as follows. We are interested in synthesizing systolic arrays not for a single instance of the problem specified by the RELD, but for a class of problems, which are defined by a single set of dependency matrices and a *family* of parameterized domains. Typically the parameter, n represents the size of the problem input. We would like the architectures that we derive to be "linearly extensible" i.e. be able to solve problems of larger size merely by adding more processors. This implies that the same timing and allocation functions should be applicable to the entire family. This extensibility is difficult to achieve if the timing function is non-linear.

Allocation functions are mappings of the problem domain D to a new (processor) domain D_a. Intuitively, an allocation function a(p) defines the processor on which the computation denoted by point p is performed. The processor domain D_a is restricted to be two-dimensional since we are dealing with systolic arrays and each processor is connected to a nearest neighbor according to a particular interconnection scheme. The interconnection is one of two possible scheme -- to four immediate neighbors, corresponding to mesh arrays (and linear for the one-dimensional case); and to six neighbors, corresponding to hexagonal arrays. An important constraint that the allocation function must satisfy is *conflict freedom* as defined below.

Definition: The timing function t and the allocation function a of an RELD are said to be free of conflict if

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

The reason for this constraint is that we cannot perform two different computations

[1]Henceforth, a subscript T indicates the transpose of a matrix or a vector

(represented by the two points p and q in the original domain) on the same processor at the same time instant. As in the case of timing functions we shall concentrate on affine allocation functions. Thus the allocation function is defined as

$$a(p) = [x, y] = [\lambda^T_x p + \alpha_x, \lambda^T_y p + \alpha_y]$$

and it thus corresponds to a geometric projection of the original domain.

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. Since a(p) and t(p) are conflict-free, it directly follows that this transformation is injective, since two distinct points in the original domain cannot be mapped to the same point in the [x,y,t] domain. We shall now prove a theorem that shows how affine timing and allocation functions permit us to cleanly separate the space and time components of the computation, while still retaining a linear dependency structure.

Theorem III.1:

For any RELD defined by $[A_j, b_j]_{j=1..m}$ the dependency structure induced by the timing function, $[\lambda_t, \alpha_t]$ and the allocation function $[\lambda_x, \alpha_x]$, $[\lambda_y, \alpha_y]$ is also an RELD if $\lambda = [\lambda^T_x, \lambda^T_y, \lambda^T_t]^T$ has an inverse, λ^{-1}.

Proof:

The transformation S defined by the timing and allocation function can be viewed as a geometric manipulation (i.e. a translation and a scaling) of the original dependency structure defined by

$$\begin{bmatrix} x \\ y \\ t \end{bmatrix} = S(p) = \lambda p + \alpha; \quad \text{where} \quad \lambda = \begin{bmatrix} \lambda^T_x \\ \lambda^T_y \\ \lambda^T_t \end{bmatrix} \text{ and } \alpha = \begin{bmatrix} \alpha_x \\ \alpha_y \\ \alpha_t \end{bmatrix}$$

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

$$\begin{aligned} f'[x, y, t] &= f(p) = f(S^{-1}[x, y, t]) \\ &= g(f(A_1 S^{-1}[x, y, t] + \bar{b}_1),\ f(A_2 S^{-1}[x, y, t] + \bar{b}_2), \\ &\qquad \ldots f(A_k S^{-1}[x, y, t] + \bar{b}_k) \\ &\quad) \\ &= g(f'(\lambda(A_1 S^{-1}[x, y, t] + \bar{b}_1) + \alpha), \\ &\qquad f'(\lambda(A_2 S^{-1}[x, y, t] + \bar{b}_2) + \alpha), \\ &\qquad \ldots f'(\lambda(A_k S^{-1}[x, y, t] + \bar{b}_k) + \alpha)) \\ &\quad) \end{aligned}$$

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

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

Since $\lambda A_j \lambda^{-1}$ is a constant 3x3 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 III.2: *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

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

∎

B. Part II: Pipelining onto a systolic implementation

We see that by using appropriate timing and allocation functions, we have reduced the original problem to a three dimensional RELD defined by $[A'_j, b'_j]_{j=1..m}$. This RELD corresponds directly to a two dimensional processor aray. 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). We therefore proceed to the second step of the synthesis procedure, namely pipelining in this array structure. Any dependency in the [x,y,t] 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 enables us to restructure the dependencies in the RELD.

Theorem III.3: ***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 $\vec{\rho}$.

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

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

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 jth dependency is now *uniform*. For it to be computationally equivalent to the original one, the following must hold.

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

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

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

and

$$f_2(p + \vec{\rho}) = f_2(p + 2\vec{\rho})$$

And thus for equivalency with the original RELD

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

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

$$A_j(p + \vec{\rho}) + b_j = A_j p + b_j$$

i.e.

$$A_j\,\vec{\rho} = \vec{0}$$

Thus for the new Uniform RE to be computationally equivalent to the original one $\vec{\rho}$ has to be a zero of the dependency matrix. ▮

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

Corollary III.4: *If all the dependency matrices A_j have zeroes of the form $[a, b, -k]^T$ where k is a positive integer and $[a,b]^T$ is one of $[0, 0]$ $[\pm 1, 0]$, $[0, \pm 1]$ or $[\pm 1, \pm 1]$, the result of making the depedencies uniform yields a systolic array.*

Proof: The proof is obvious, since it is the vector $\vec{\rho}$ that determines the point in the [x, y, t] space that any point depends on. If $\vec{\rho}$ 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. ▮

The synthesis procedure thus consists of the following steps.

1. Determine an RELD for the problem.
2. Find appropriate allocation and timing functions
3. Compute the new RELD in the processor-time domain induced by the timing and allocation functions.
4. Test to see if dependencies are pipelineable
5. If so derive the systolic implementation, otherwise try alternate timing and allocation functions and return to step (3)

IV Application of the Technique to LU Decomposition

The results of the previous section give us a *constructive* technique to synthesize systolic arrays. Once appropriate timing and allocation functions have been determined, the test for *pipelineability* yields the zero, $\vec{\rho} = [a, b, -k]$ of the dependency matrix A_j. This zero, if it exists, automatically determines the architecture, i.e. the processor functionality and the interconnection structure. Any processor in the architecture now performs a set of functions. In addition to "normally" computing the *g* function on its input values, it also performs a pipelining operation by forwarding one of its inputs to its [-a, -b] neighbor over a link that has a delay of k time units ! We shall illustrate the technique by means of an example. Consider the LU decomposition of a band matrix as defined in Figure IV-1.

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

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

As described by Kung and Leiserson [13] 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

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

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

A. 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.

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

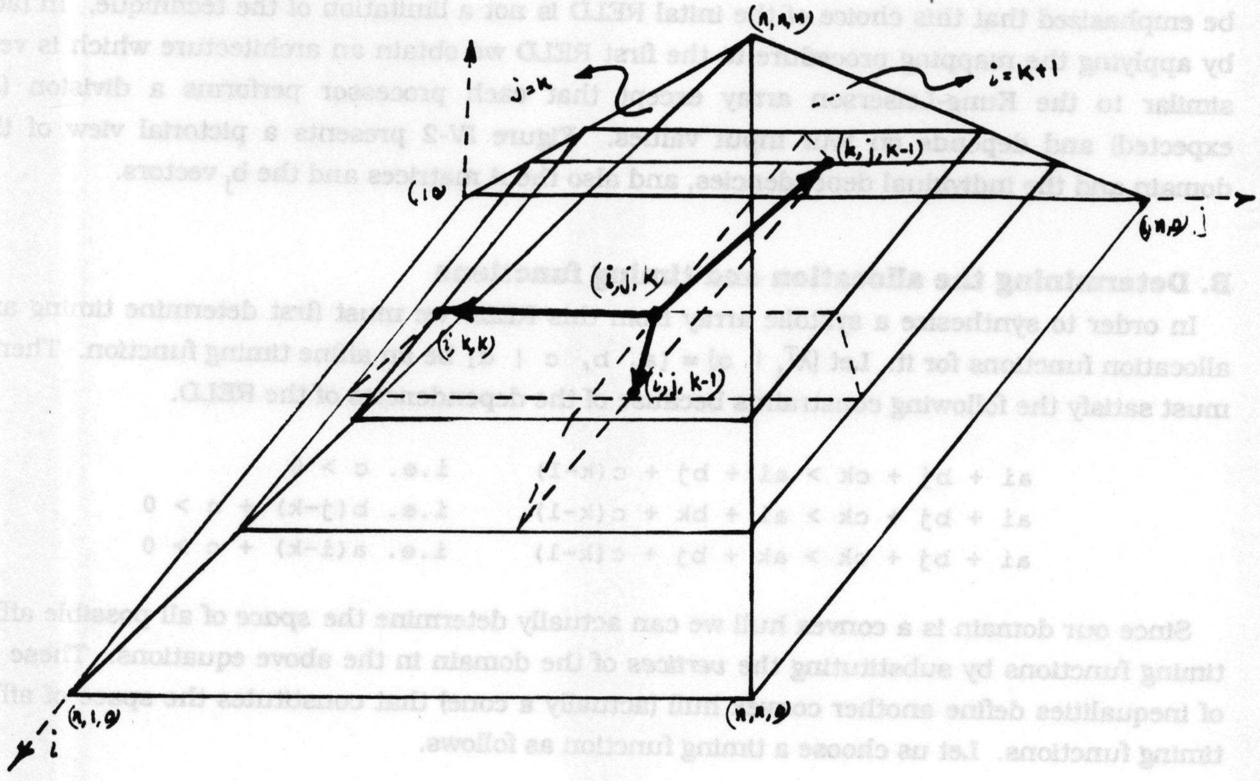
$$a(i,\ j,\ k) = a(i,\ j,\ k-1) - a(i,\ k,\ k-1) \cdot a(k,\ j,\ k-1)/a(k,\ k,\ k-1)$$

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

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

l(i,j,k), with l being meaningful only at the j = k+1 boundary[2]. 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) = if j = k+1 then a(i, j, k)/a(j, j, j-1)
```



$$A_1' = \begin{bmatrix} 1, & 0, & 0 \\ 0, & 1, & 0 \\ 0, & 0, & 1 \end{bmatrix}, \quad \bar{b}_1' = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}; \quad A_2' = \begin{bmatrix} 1, & 0, & 0 \\ 0, & 0, & 1 \\ 0, & 0, & 1 \end{bmatrix}, \quad \bar{b}_2' = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix};$$

$$\text{and } A_3' = \begin{bmatrix} 0, & 0, & 1 \\ 0, & 1, & 0 \\ 0, & 0, & 1 \end{bmatrix} \quad \bar{b}_3 = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}$$

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

[2]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

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 inital 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 IV-2 presents a pictorial view of the domain and the individual dependencies, and also the *A* matrices and the b_j vectors.

B. 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.

$$\begin{array}{ll} ai + bj + ck > ai + bj + c(k-1) & \text{i.e. } c > 0 \\ ai + bj + ck > ai + bk + c(k-1) & \text{i.e. } b(j-k) + c > 0 \\ ai + bj + ck > ak + bj + c(k-1) & \text{i.e. } a(i-k) + c > 0 \end{array}$$

Since our domain is a convex hull we can actually determine the *space* of all possible affine timing functions by substituting the *vertices* of the domain in the above equations. These set of inequalities define another convex hull (actually a cone) that constitutes the space of affine timing functions. Let us choose a timing function as follows.

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

We can easily verify that t(i,j,k) as defined above is indeed a valid timing function. Also, since we require that the allocation function a(i,j,k) that we choose must not be 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

$$\begin{array}{ll} & i + j + k = p + q + r \\ & i - k = p - r \\ \text{and} & j - k = q - r \end{array}$$

Hence $i + j - 2k = p + q - 2r$
Subtracting this from the first eqn yields $3k = 3r \Rightarrow k = r$
Substituting this in the second and third eqns yields $i = p$ and $j = q$. Thus the two points are identical. We thus have

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

C. Pipelining in the processor-time domain

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

$$A_1' = \lambda A_1 \lambda^{-1} = \begin{bmatrix} 1, & 0, & 0 \\ 0, & 1, & 0 \\ 0, & 0, & 1 \end{bmatrix} \quad \text{and since } \alpha = \vec{0},\ \bar{b}_1' = \lambda \bar{b}_1 = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}$$

Similarly

$$A_2' = \begin{bmatrix} 1, & 0, & 0 \\ 0, & 0, & 0 \\ 0, & -1, & 1 \end{bmatrix}, \quad \bar{b}_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad \text{and} \quad A_3' = \begin{bmatrix} 0, & 0, & 0 \\ 0, & 1, & 0 \\ -1, & 0, & 1 \end{bmatrix}, \quad \bar{b}_3' = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix};$$

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 III.B in order to pipeline in this new structure. this requires us to solve

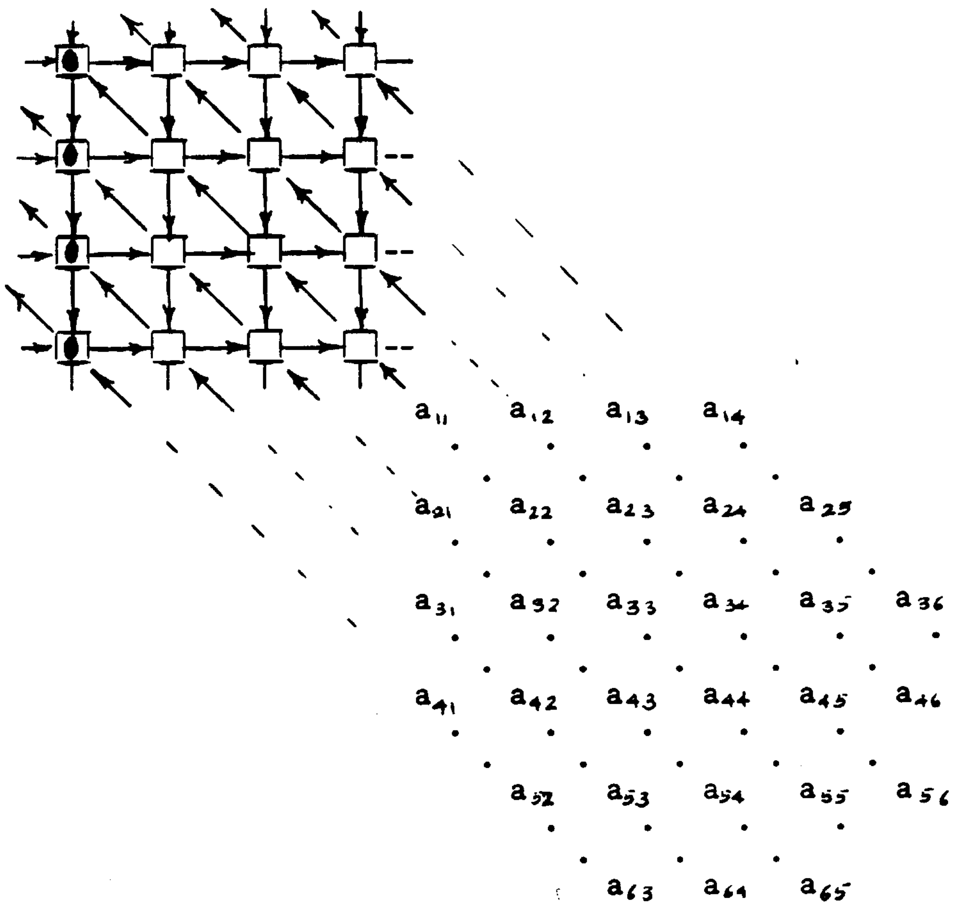
$$A_2'\vec{x} = 0 \qquad \text{(and correspondingly} \quad A_3'\vec{x} = 0)$$

and yields $[-k, 0, -k]^T$ (and respectively $[0, -k, -k]^T$) as a solution. Choosinng k to be 1, both the second and third dependencies can be pipelined, and 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 IV-3 below, which is identical to the one described by Kung and Leiserson.

V Conclusions

We have presented a technique for designing systolic arrays from an initial specfication which is in the form of a Recurrence Equation with Linear Dependencies. The approach that we have taken here may be viewed as an extension of Quinton's approach where only Recurrence Equations with Uniform Dependencies (UREs) are considered. In our approach the class o problems discussed is a superset of UREs. It is essential to include UREs in our class, because the final target that we are interested in is, in fact a URE (since systolic arrays have *local* and *regular* interconnections - this regularity implies UREs). An alternative perspective of our approach is obtained by envisioning the above steps as transforming an RELD into a URE.

Recently (in [3, 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 addional 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



a_{11} a_{12} a_{13} a_{14}
a_{21} a_{22} a_{23} a_{24} a_{25}
a_{31} a_{32} a_{33} a_{34} a_{35} a_{36}
a_{41} a_{42} a_{43} a_{44} a_{45} a_{46}
a_{52} a_{53} a_{54} a_{55} a_{56}
a_{63} a_{64} a_{65}

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

"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.

As an extension to this work, there are three major areas of further research. One important problem that needs to be addressed is alternate pipelining strategies, such as those involving control signals to alter the speed of data flow (while keeping it constant at any given instant). An example of such an architecture is the dynamic programming array presented by Guibas *et al*(in [9]). Another area for further research is the use of the desired pipelining structure to *guide* the choice of allocation and timing functions. Finally, a detailed investigation of the nature of timing and allocation functions is in progress. Preliminary results indicate that for the case when the domain of the RELD is a convex hull, we are able to constructively derive the space of all possible timing functions as a cone in (n+1) dimensional space. Also, other techniques such as results from conventional processor scheduling may be applicable in setting bounds on such timing functions.

References

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

2. Cappello, Peter R. and Steiglitz, Kenneth. Unifying VLSI Array Designs with Geometric Transformations. Proc. IEEE Parallel Processing Conference, Aug, 1983.

3. Chen, Marina. A Parallel Language and its Compilation to Multiprocessor Machines or VLSI. Principles of Programming Languages, ACM, 1986. To appear.

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

5. Chen, Marina C. *Space-Time Algorithms: Semantics and Methodology.* Ph.D. Th., California Institute of Technology, Pasadena, CA, May 1983.

6. Uri Weiser and Alan L. Davis. Mathematical Representation for VLSI Arrays. UUCS-80-111, Department of Computer Science, University of Utah, Sept 1980.

7. Delosme, Jean-Marc and Ipsen Ilse 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, Taiwaan, 1985, pp. 268-273.

8. Delosme, Jean-Marc and Ipsen, Ilse C. F. Efficient Systolic Arrays for the solution of Toeplitz Systems: An illustration of a methodology for construction of systolic architectures in VLSI. YALEU/Dept. Of Computer Science/RR-370, Yale University, Department of Computer Science, June, 1985.

9. 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, Jan, 1979, pp. 509:525.

10. Ramakrishnan, I. V., Fussell, D. S. and Silberschatz, A. "Mapping Homogeneous Graphs on Linear Arrays". *IEEE Transactions on Computers* , (?? 1985), ??-??.

11. S. Lennart Johnsson and Danny Cohen. A Mathematical Approach to Computational Networks For the Discrete Fourier Transform.

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

13. 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.

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

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

16. C.E. Leiserson and J.B. Saxe. Optimizing Synchronous Systems. 22nd Annual ACM Symposium on Foundations of Computer Science, ACM, Oct, 1981, pp. 23:36.

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

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

19. Rajopadhye, Sanjay V. A formal basis for synthesizing systolic arrays: PhD Thesis proposal. UUCS-84-010, Universityof Utah, Computer Science Department, November, 1984.