SYSTOLIC ARRAY SYNTHESIS BY STATIC ANALYSIS OF PROGRAM DEPENDENCIES

Technical Summary

Sanjay V. Rajopadhye
Computer and Information Science Dept.
University of Oregon
Eugene, Or 97403

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

Abstract

We present a technique for mapping recurrence equations to systolic arrays. While this problem has been studied in fairly great detail, the recurrence equations that are analysed here are a generalization of those studied previously. In an earlier paper [14] we have showed how systolic arrays can be synthesized from such generalized recurrence equations by a combination of *affine transformations* and *explicit pipelining*. This paper extends the results in two directions. Firstly, a *multistage* pipelining technique is proposed, which permits the synthesis of systolic arrays with *irregular* data flow. Secondly we develop analysis techniques for the synthesis of systolic arrays whose computation is governed by *control signals* in a systematic manner which is amenable to mechanization. The full paper also discusses how these techniques can be applied to the mapping problem for more general architectures.

1. INTRODUCTION

Systolic arrays are a class of parallel architectures consisting of regular interconnections of a very large number of simple processors, each one operating on a small part of the problem. They are typically designed to be used as back-end, special-purpose devices for computation-intensive processing. A number of such architectures have been proposed for solving problems such as matrix multiplication, L-U decomposition of matrices, solving a set of equations, convolution, dynamic programming, etc. (see [5, 6, 7] for an extensive bibliography).

Most of the early systolic arrays were designed in an *ad hoc*, case-by-case manner. Recently there has been a great deal of effort on developing unifying theories for automatically synthesizing such arrays [1, 2, 3, 8, 9, 10, 11, 12, 13, 16, 17]. The approach is to analyze the program dependency graph and transform it to one that represents a systolic array. The problem of synthesis can thus be viewed as 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, and the usage of variables within it defines the dependencies 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, and (in this case p-q is a *constant* vector independent of p, namely `[0,-2,1]`^{T}). Such a nested-loop program is exactly equivalent to a recurrence equation defined on the same domain. For notational convenience we shall use recurrence equations in the rest of this paper. In most of the earlier work cited above the underlying assumption is that the dependencies are uniquely characterized by a finite set, W of *constant vectors* in Z^n. The recurrence equations in this case 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.

In an earlier paper [14] we have argued that this constraint of uniform dependencies is very restrictive and have proposed and analyzed the case when they are linear functions of the point. Thus, for the simple example above, the body of the loop may contain a statement of the form `a[i,j,k] := 2*b[i',j',k']`, where each of `i'`, `j'` and `k'` are *linear* functions of `i`, `j` and `k`. The recurrence equations characterizing such computations are called Recurrence Equations with Linear Dependencies (RELDs). In this paper we extend these results to permit the synthesis of systolic arrays with *control signals* and *irregular data flow*. To do so we propose a new class of recurrence equations, called Conditional Uniform Recurrence Equations (CUREs). The synthesis problem then involves two steps. It is first necessary to derive such a CURE from the initial RELD. This step is achieved by means of a technique called *multistage pipelining* which is a generalization of the pipelining presented in [14]. The second step involves mapping the CURE to a target systolic array, and involves introducing *control dependencies* and then determining an affine transformation. We illustrate the technique by systematically deriving a well known systolic array for dynamic programming [5]. A satisfactory technique that can synthesize this architecture has not appeared in the literature.[1]

2. RECURRENCE EQUATIONS WITH LINEAR DEPENDENCIES

Definition 2.1: A *Recurrence Equation with Linear Dependence* (RELD) over a domain D is defined as an equation of the form

$$f(p) = g\ (f(A_1 p + b_1),\ f(A_2 p + b_2)\ \ldots\ f(A_k p + b_k))$$

where $p \in D$;
A_i's are constant $n \times n$ matrices;
b_i's are constant n-dimensional vectors;
and g is a strict, single-valued function.

[1]The techniques that do synthesize this array either require one to explicitly enumerate the dependencies and do not give systematic techniques to derive the *affine transformations* [12]; or require an inductive analysis of the *entire* dependency graph [2]. Recently another technique has been presented [4], but that requires a somewhat awkward notation, and is less general than ours.

A *system* of m RELDs over a domain D is defined to be a family of m mutually recursive such equations, with each equation defining one of the f_i's.

Note that if the A_i matrices are all equal to the identity matrix I, the RELD becomes a URE. Thus RELDs are a super-set of UREs. Synthesizing a systolic architecture from an RELD involves two major steps as follows.

2.1. Synthesis by Affine Transformations

The function g is assumed to be computed in one *time instant*, and this represents the granularity of the computation. Thus the problem of synthesizing a systolic array is exactly the problem of assigning each point in D to a processor, and scheduling it at a particular time instant. This is achieved by means of an *affine transformation* (defined by an $n \times n$ matrix λ and an $n \times 1$ vector α). The image p' of p under this transformation is given by

$$p' = \lambda p + \alpha = \begin{bmatrix} \lambda_t \\ \lambda_a \end{bmatrix} p + \begin{bmatrix} \alpha_t \\ \alpha_a \end{bmatrix}$$

It is convenient, as shown above, to separate λ and α into two parts, λ_t and λ_a (and α_t and α_a respectively), where λ_t is an $1 \times n$ vector, λ_a is an $(n-1) \times n$ matrix, λ_a is a scalar constant and λ_a is an $(n-1) \times 1$ vector. The pair $[\lambda_t, \alpha_t]$ defines what is called a *timing function* and $[\lambda_a, \alpha_a]$ defines an *allocation function* for the RELD. Thus $\lambda_t p + \alpha_t$ defines the time instant at which the computation of $f(p)$ is scheduled, and $\lambda_a p + \alpha_a$ defines the processor (in an (n-1)-dimensional processor domain[1]) at which $f(p)$ is to be performed. There are two major constraints that the timing and allocation functions must satisfy. Firstly, the timing function must preserve causality, i.e., if p depends on q then $\lambda_t p + \alpha_t$ must be greater than $\lambda_t q + \alpha_t$. Secondly, the timing and allocation functions must not assign two different points in D to the same processor at the same time, i.e., $\forall p, q \in D \quad t(p) = t(q)$ *and* $a(p) = a(q) \Rightarrow p = q$ Techniques for determining appropriate timing and allocation functions have been discussed elsewhere [15] and are beyond the scope of this summary.

2.2. Explicit Pipelining

The idea of synthesizing systolic arrays by using affine transformations is identical to that used for UREs. For UREs, however, the original problem-dependencies define a *lattice*, and affine transformations are guaranteed to yield a regular interconnection in the final architecture. This is not true for RELDs, and this fact motivates the need for explicitly pipelining the dependencies. The idea behind pipelining is as follows. Consider a dependency $[A_j, b_j]$ in the original RELD. Thus, any point p in D requires the value of f at $q = (A_j p + b_j)$ as its j^{th} argument. It is clear that if there is one (or more) other point p' in D that also depends on q, then the dependency of p on q may be replaced by a dependency of p on p'. It can be shown

---

[1]Since systolic arrays are planar architectures this would seem to restrict the application of our techniques to RELDs that are at most three-dimensional. This is not true, since for a higher dimensional RELD, one may first synthesize an architecture which is n-1 dimensional, and then *optimize* it to obtain a two dimensional one. The interested reader is referred to [15] for details.

that there is a direct correspondence between the set of such points that *share* $f(q)$, and the solution space of the equation $A_j x = 0$. In fact, if ρ is a *basis vector* for this solution space, then the linear dependency can be replaced by a uniform dependency ρ, provided $A_j(A_j p + b_j - p)$ is a constant vector ρ', independent of p (ρ' is called the *terminal dependency*). Intuitively this condition is explained as follows. If p depends on a point q ($= A_j p + b_j$) and if ρ is the basis vector for the space $A_j x = 0$ then the set $S_j = \{ p+k\rho \mid k \in Z \}$ is exactly the set of points that depend on q, henceforth called the *null set* for the j^{th} dependency. Then the condition for direct pipelining merely states that the point q must belong to this set (or be a constant vector away).

These two steps (affine transformations and explicit pipelining) are both independent of each other. In a typical synthesis process one would first determine an appropriate timing function, then pipeline the dependencies, and finally choose an allocation function. Henceforth, the kind of pipelining discussed here will be referred to as *direct pipelining*. It is also important to note that the choice of the pipelining dependency ρ is not unique. In particular, if p and p′ are two ("neighboring") points in D that both depend on some other point q (and $p' - p = \rho$), then it is equally correct to choose to have either p depending on p′ (i.e., have ρ as a dependency) or to have p′ depending on p (i.e., have $-\rho$ as a dependency). As a result, the choice of the *direction* of pipelining is governed by the timing function. The idea is to introduce the new dependencies in a λ_t-*consistent* manner (i.e., if $\lambda_t p > \lambda_t p'$ then p should depend on p′ and thus the new dependency should be ρ, otherwise it should be $-\rho$).[1]

3. CONDITIONAL UNIFORM RECURRENCE EQUATIONS

Definition 3.1: A *Conditional Uniform Recurrence Equation* (CURE) over a domain D is defined to be an equation of the form

$$f(p) = \begin{cases} g_1(f(p - w_{1,1}), f(p - w_{1,2}) \ldots f(p - w_{1,k_1})) & \text{if } \theta_1 p > \pi_1 \\ g_2(f(p - w_{2,1}), f(p - w_{2,2}) \ldots f(p - w_{2,k_2})) & \text{if } \theta_2 p > \pi_2 \\ \vdots & \\ g_{k-1}(f(p - w_{k-1,1}), f(p - w_{k-1,2}) \ldots f(p - w_{k-1,k_{k-1}})) & \text{if } \theta_{k-1} p > \pi_{k-1} \\ g_k(f(p - w_{k,1}), f(p - w_{k,2}) \ldots f(p - w_{k,k_k})) & \text{otherwise} \end{cases}$$

where $p \in D$;
θ_i's are constant vectors in I^n;
π_i's are scalar constants in I;
$w_{i,j}$ are constant n-dimensional vectors;
and g_i's are strict, single-valued functions.

A family of m CUREs is a set of m such mutually recursive equations, each one defining one of m such functions. In addition, we may define CUREs where the condition is not just a single linear inequality, but a conjunction of a number of linear inequalities.

---

[1]The only problem arises when neither ρ nor $-\rho$ can be λ_t-consistent. This will be the case if $\lambda_t p = \lambda_t p'$, i.e., $\lambda_t \cdot \rho = 0$. In this case it is necessary to choose an alternate timing function. This is illustrated in the example of Sec. 6.

A CURE is thus very similar to a URE in the sense that a point p depends on others that are a *constant vector* away. In addition, the domain for a URE has to be explicitly defined, while in CUREs the hyperplanes $[\theta_i, \pi_i]_{i = 1..k}$ automatically define a convex hull. The important distinction is that in UREs the value of f at the boundaries of the domain is a constant (obtained from the external world) while in CUREs the value at a boundary depends on the boundary and is in general a function g_i of the values at some other points in the domain.[1] It is in fact, this *redirection* of the computation back within the domain that makes the determination of an affine timing function (ATF) for an arbitrary CURE a difficult problem in general.

4. DERIVING CUREs FROM RELDs

From the above discussion it is clear that CUREs represent a class of computations that are more general than UREs, but less so than RELDs. As a result they serve as a useful intermediate step in deriving a systolic array from a RELD. As we have discussed earlier, a dependency of an RELD can be *directly* pipelined if $A_j \cdot (A_j p + b_j - p) = \rho'$. However, this condition may not always hold, and this motivates the need for *multistage pipelining*.

4.1. Multistage Pipelining

Multistage pipelining may be applicable if a particular dependency cannot be *directly* pipelined. Let $S_j = \{ p + k\rho_j \mid k \in Z \}$ be the null set for $[A_j, b_j]$. It is clear that S_j *must* intersect boundary (say $\theta_j\, p = \pi_j$) of D. Let the point of intersection be q'.[2] Now, the dependency $[A_j, b_j]$ can be *multistage pipelined* if

$$A_i(A_j p + b_j - q' - \rho_j') = \rho_i'$$

and $[A_i, b_i]$ is a dependency that can be pipelined (with ρ_i' as its terminal dependency), and ρ_j' is a constant. Intuitively, this means that the point q' needs the value of f at $A_j p + b_j$ as it's j^{th} argument (which is obvious since it is a member of S_j), and also requires the same value *as its* i^{th} *argument*, and the i^{th} dependency can be pipelined. The modified RELD (i.e., the RELD with the j^{th} dependency pipelined as above) is as follows.

$$f(p) = g\,(f(A_1 p + b_1),\ f(A_2 p + b_2)\ \ldots\ f_i(p)\ \ldots\ f_j(p)\ \ldots\ f(A_k p + b_k))$$

$$f_j(p) = \begin{cases} f_i(p+\rho_j') & \text{if } \theta_j\, p = \pi_j \\ f_j(p+\rho_j) & \text{otherwise} \end{cases} \quad \text{and} \quad f_i(p) = \begin{cases} f(p+\rho_i') & \text{if } \theta_i\, p = \pi_i \\ f_i(p+\rho_i) & \text{otherwise} \end{cases}$$

The functions f_i and f_j are additional computations to be performed at every point in D. They are "dummy" functions since at all internal points they merely return their value from a neighboring point, and thus achieve the pipelining effect.

---

[1] It is clear that a URE can be easily viewed as CUREs where all except one of the g_i's are constant functions.

[2] This is so because D is a convex hull, and S_j is a straight line passing through p, which a a point inside D. It is also shown in the full paper that q' must be the earliest scheduled point in S_j, if ρ_j is λ_i-consistent.

5. SYNTHESIZING SYSTOLIC ARRAYS FROM CUREs

By the process described above the original RELD is reduced to a CURE for which $[\lambda_t, \alpha_t]$ is a valid ATF.[1] Thus the final step in the synthesis procedure is choosing an appropriate allocation function $[\lambda_a, \alpha_a]$ and deriving from it the final architecture and its interconnections. The constraints that the allocation function must satisfy are that the interconnections induced in the final architecture are nearest-neighbor. This will be true provided that for each of the dependency vectors w_i,

$\lambda_a w_i \in P$; where $P = \{ (0,0), (0,1), (0,-1), (1,0), (-1,0), (1,1), (-1,-1)\}$ is the set of permissible interconnections in the final architecture.

This technique, which is applicable to UREs, can be directly used for CUREs too.[2] This *naive* implementation would require the computation of *each* of the conditional expressions in *every* processor at *all* time instants, and would therefore be hopelessly inefficient. We shall therefore describe an approach whereby the evaluation of these conditions is achieved by means of control signals. We shall also describe how the control signals can be derived in a straightforward manner by a technique very similar to the pipelining of dependencies.

One simple solution to this problem is to again use the notion of pipelining that we have developed in the previous section. Observe that (1) part of the computation that is to be performed at each point is the evaluation of a (set of) boolean expression(s) $\theta\ p = \pi$; and (2) $\theta\ p = \pi$ corresponds to a hyper-plane. Then if we choose a new vector σ such that (1) the dot product $\sigma\ \theta = 0$; and (2) σ is λ_t-consistent; then for any point p, the set $S = \{ p' \mid p' = p + k\sigma, k \in Z\}$ is the set of points for *each* of which the condition $\theta p' = \pi$ is true. The earliest scheduled point in S is a point at the edge of the bounding (hyper)plane of the domain. The value of the conditional expression (i.e., the boolean value *true*) is assumed to be available from the external world at this point. Thus, just as the linear dependencies were replaced by uniform dependencies, the evaluation of conditional expressions can be replaced by the propagation of *control signals*. Now, all that remains in the synthesis procedure, is to choose an allocation function $[\lambda_a, \alpha_a]$ subject to the *additional* constraints that each of the control dependencies σ_i's are also mapped to neighboring processors, i.e., $\lambda_a w_i \in P$.

6. EXAMPLE: OPTIMAL STRING PARENTHESIZATION

We shall now illustrate the entire procedure by synthesizing a systolic architecture for the optimal string parenthesization problem. 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.

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

---

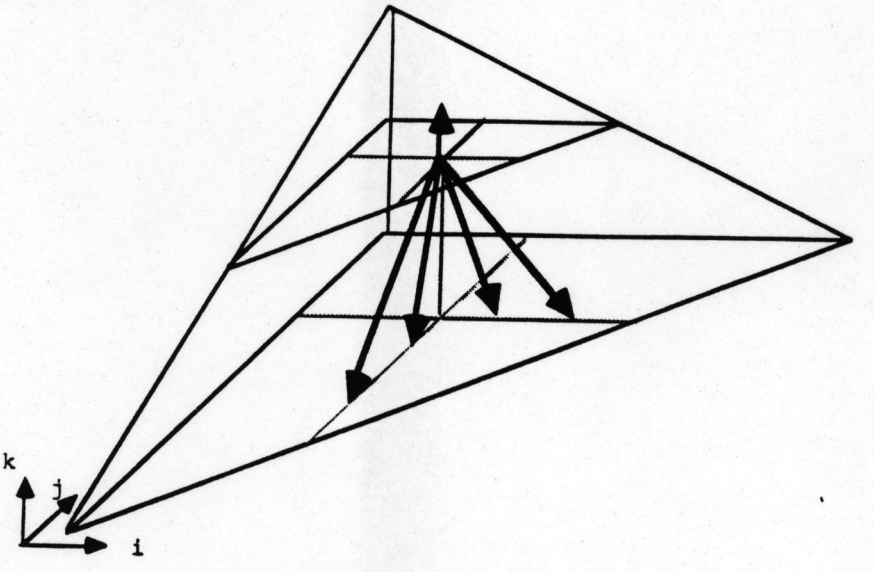
[1]Remember that each of the new dependencies that have been introduced is λ_t-consistent.

[2]Note however, that in the case of CUREs the computation at any point p is much more complicated than merely the computation of a strict function g on a given set of arguments. It is necessary to evaluate one or more of the conditions $\theta_i\ p = \pi_i$, and thus the computation depends in a nontrivial manner on the point p itself.

By introducing an "accumulation index" we can express the computation as the following RELD. A pictorial view of the domain, and the dependencies is shown in Fig. 6-1.

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

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



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

$$A_4 = \begin{bmatrix} 1, & 0, & 0 \\ 0, & 1, & -1 \\ 0, & 0, & 0 \end{bmatrix}, \; b_4 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}; \quad \text{and} \quad A_5 = \begin{bmatrix} 0, & 1, & -1 \\ 0, & 1, & 0 \\ 0, & 0, & 0 \end{bmatrix}, \; b_5 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix};$$

Figure 6-1: *Dependency Structure for the RELD*

The first step in the synthesis procedure is to determine an ATF, [λ, α] (denoted by `[[a,b,c], α]`) for this RELD. It can be shown that this can be determined by a solution to the following set of inequalities.

$b(j-i-k) > -c(k-1)$; $ak < c(k-1)$; $c < 0$; $bk > c(k-1)$; and $a(j-i-k) < c(k-1)$;

Since a, b and c are restricted to be integers, and j-i ≥ 2k (from the definition of the domain for the RELD), these can be reduced to the following.

$$c < 0; \qquad b \geq -c \quad \text{and} \quad a \leq c$$

The optimal ATF thus corresponds to the smallest integer (absolute-valued) solution to the above inequalities, and is given by $\lambda_{opt} = [-1, 1, -1]^T$, $\alpha_{opt} = 1$, i.e.,

$$t(i, j, k) \equiv j - i - k + 1$$

The next step in the synthesis procedure is to pipeline the linear dependencies. For this, the null space of each of A_1, A_2, A_4 and A_5 must first be computed. It is easy to show that the rank of each of these matrices is 2, and hence their null spaces are all one-dimensional, specified by a single basis vector each, say ρ_1, ρ_2, ρ_4 and ρ_5, respectively. It is also straightforward to solve the appropriate systems of equations and obtain the following.

$$\rho_1 = [0, m_1, 0]^T; \qquad \rho_2 = [m_2, 0, m_2]^T; \qquad \rho_4 = [0, m_4, m_4]^T; \qquad \rho_5 = [m_5, 0, 0]^T$$

However, the dot-product $\rho_2 \cdot \lambda_{opt}$ is $[m_2, 0, -m_2]^T \cdot [-1, 1, -1]$ which is zero (similarly $\rho_4 \cdot \lambda_{opt}$ is also zero). This means that it is impossible to obtain a λ_t-consistent basis for the null spaces of either A_2 or A_4. We must therefore choose another λ_t, for which the basis of the null spaces of A_j and A_j can be λ_t-consistent. The vector $[-2, 2, -1]$ satisfies this requirement, and hence a satisfactory ATF is the following (in fact, it can be shown that this is the optimal ATF, given the additional constraints of λ_t-consistency).

$$t(i, j, k) \equiv 2(j - i) - k + 1$$

Now, λ_t-consistent basis vectors for each of the dependencies can be derived, and are $[0, -1, 0]^T$, $[1, 0, -1]^T$, $[0, -1, -1]^T$ and $[1, 0, 0]^T$, respectively.

The next step is to test each of the dependencies for direct pipelining. The values of $A_j(A_j p + b_j - p)$ for each of A_1, A_2, A_4 and A_5 are computed to be $[0, -k+1, 0]^T$, $[-1, 0, 0]^T$, $[0, 1, 0]^T$, and $[-k+1, 0, 0]^T$ respectively. Thus, it is clear that only $[A_2, b_2]$ and $[A_4, b_4]$ can be directly pipelined by introducing a new pair of dependencies, ρ_2 and ρ_4 (along with the terminal dependencies $\rho_2' = [1, 0, 0]^T$ and $\rho_4' = [0, -1, 0]^T$), respectively. It is also a simple matter to determine that both the *null sets* S_2 and S_4 of A_2 and A_4 intersect the domain boundary k=1 at [i+k-1, j, 1] and [i, j-k+1, 1], respectively. Thus the auxiliary (pipelining) functions f_2 and f_4 are as follows.

$$f_2(i, j, k) = \begin{cases} f(i+1, j, k) & \text{if } k = 1 \\ f_2(i+1, j, k-1) & \text{otherwise} \end{cases} \quad \text{and} \quad f_4(i, j, k) = \begin{cases} f(i, j-1, k) & \text{if } k = 1 \\ f_4(i, j-1, k-1) & \text{otherwise} \end{cases}$$

Since $[A_1, b_1]$ and $[A_5, b_5]$ cannot be directly pipelined, it is necessary to test for multistage pipelining of these dependencies. First, we must determine q_1' and q_5', the points at which the two null sets S_1 and S_5 intersect the domain boundary. These can be computed to be [i, i+2k, k] and [j-2k, j, k], respectively. Thus $q_1' - q$ (= Δ_1, say) is [0, k, k-1], and $q_5' - q$ (= Δ_5) is [-k, 0, k-1]. Then, for $[A_1, b_1]$ to be multistage pipelined, it must be the case that either $A_2(\Delta_1 - \rho_1') = \rho_2'$ or $A_4(\Delta_1 - \rho_1') = \rho_4'$ for some constant vector ρ_1'. We see that the latter

condition holds with $\rho_1' = [0, 0, 0]$. Similarly, since $A_2\Delta_5 = \rho_2'$, the dependency $[A_5, b_5]$ can also be multistage pipelined. The resulting CURE is as follows.

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

$$f(i,j,k) = \begin{cases} w_{i,j} & \text{if } j-i = 1 \\ w_{i,j} + \min \begin{pmatrix} f_1(i, i+k, 1) + f_2(i+k, j, 1) \\ f(i, j, k+1) \\ f_4(i, j-k, 1) + f_5(j-k, j, 1) \end{pmatrix} & \text{if } k = 1 \\ \infty & \text{if } 2*k > j-i \\ \min \begin{pmatrix} f_1(i, i+k, 1) + f_2(i+k, j, 1) \\ f(i, j, k+1) \\ f_4(i, j-k, 1) + f_5(j-k, j, 1) \end{pmatrix} & \text{otherwise} \end{cases}$$

where

$$f_2(i,j,k) = \begin{cases} f(i+1, j, k) & \text{if } k = 1 \\ f_2(i+1, j, k-1) & \text{otherwise} \end{cases} \qquad f_4(i,j,k) = \begin{cases} f(i, j-1, k) & \text{if } k = 1 \\ f_4(i, j-1, k-1) & \text{otherwise} \end{cases}$$

$$f_1(i,j,k) = \begin{cases} f_4(i,j,k) & \text{if } 2k = j-i \\ f_1(i, j-1, k) & \text{otherwise} \end{cases} \quad \text{and} \quad f_5(i,j,k) = \begin{cases} f_2(i,j,k) & \text{if } 2k = j-i \\ f_5(i+1, j, k) & \text{otherwise} \end{cases}$$

The final step in the synthesis procedure is to choose an allocation function that satisfies the constraints of locality of interconnections, and if necessary, choose appropriate λ_t-consistent control dependencies σ_t. From the above CURE, it is clear that the various conditional expressions[1] that need to be evaluated are $k = 1$, $j - i = 2k$, and $j - i = 1$.

The data dependencies of the CURE and the associated delays (derived from $\lambda \cdot \rho_i$) are

	[0, -1, 0]	for f_1	with a delay of 2 units
	[1, 0, -1]	for f_2	with a delay of 1 unit
	[0, -1, -1]	for f_4	with a delay of 1 unit
	[1, 0 0]	for f_5	with a delay of 2 units
and	[0, 0, 1]	for f	with a delay of 1 unit

The terminal dependencies are [1, 0, 0] for f_2 and [0, -1, 0] for f_4

It is also clear that the value $w_{i,j}$ is a constant value, to be input from the external world, and that all the points in D need its value when $k = 1$. The most obvious choice is thus a simple vertical projection, i.e., $\lambda_a p + \alpha_a = [i, j]$.

With such an allocation function, the five dependencies above are mapped to [0, -1], [1, 0], [0, -1], [1, 0], and [0, 0], respectively. This means that each processor [x, y] gets two values

---

[1]Note that the condition $2k > j - i$ has not been listed, since it is really concerned with points that are *outside the domain.*

(corresponding to f_2 and f_5) from processor $[x+1, y]$ over lines of delay and 2 time units, respectively; it similarly receives two values (f_1 and f_4) from processor $[x, y-1]$ over lines of delay 1 and 2, respectively. The value corresponding to f remains in the processor (in an accumulator register) and is updated on every cycle. Thus the only remaining problem is to choose appropriate control dependencies for the three control planes k = 1, j - i = 2 k and j - i = 1, specified by $\theta_1 = [0,0,1]^T$ $\theta_2 = [-1,1,-2]^T$ and $\theta_3 = [-1,1,-1]^T$, respectively. However, the third one intersects the domain at only one line $[i, i+1, 1]$ and this line is mapped by the allocation function to the processors $[i, i+1]$.[1] As a result there is no need for a control signal. All the processors $[i, i+1]$ merely output the value $w_{i,i+1}$ at time t = 1. The important control dependencies are thus those corresponding to θ_1 and θ_2. These correspond to control signals σ_1 and σ_2 if $\theta_1 \cdot \sigma_1 = 0$ and $\theta_2 \cdot \sigma_2 = 0$. It is very straightforward to deduce that σ_1 should be $[c_i, c_j, 0]$ where c_i and c_j are arbitrary integers. The conditions of λ_t-consistency yield one constraint, namely $2(c_j - c_i)$ must be negative, and the constraint of locality of interconnections yields another constraint, namely that the vector $[c_i, c_j]$ must be one of the six permissible interconnection vectors $\{(\pm 1, 0), (0, \pm 1), (\pm 1, \pm 1)\}$. This yields only two possible values for σ_1, $[1, 0]$ and $[0, -1]$ and any one of them can be chosen (say the former). This corresponds to a vertical control signal that travels with a delay of two time units. For σ_2, the analysis is similar; $\sigma_2 \cdot \theta_2 = 0$ yields $\sigma_2 = [c_i, c_i+2c_k, c_k]$, λ_t-consistency yields $3c_k < 0$. However, nearest-neighbor interconnection cannot be achieved, since the smallest (absolute) value for c_k that satisfies λ_t-consistency is -1, and that is not nearest-neighbor (any larger absolute value for c_k will correspond to an even more distant interconnection). If this last constraint is relaxed $\sigma_2 = [0, -2, -1]$ is a valid choice. This corresponds to a (horizontal) control signal that connects every alternate processor and travels at a speed of two processors every three time units. The final architecture that this yields is shown in Fig. 6-2 and is identical to the one developed by Guibas et al [5].

7. CONCLUSIONS

We have developed a technique for analyzing the dependencies of a general class of recurrence equations, and synthesizing systolic arrays with control signals and irregular data flow. In the context of this summary we had tacitly assumed that the solution space of Ax = 0 was characterized by a single basis vector ρ. In general, however, this solution space is a hyperplane, and its dimension depends on the rank of A. The case when the rank is n (i.e., A is singular) is interesting, since in that case there can be no pipelining (for every point p there is a unique point $A^{-1}p$ which depends on it), and alternative techniques [16] need to be used. If the rank of A is less than n-1, then the set of points that share a value constitute a (hyper)plane. Both these cases are discussed in greater detail in the full paper.

In the beginning of this summary we had stated that most of the previous work concentrated on an analyzing *uniform* dependencies. There have been a few attempts towards a more

---

[1] In fact this line is the only part of the domain that is mapped to this subset of the processors

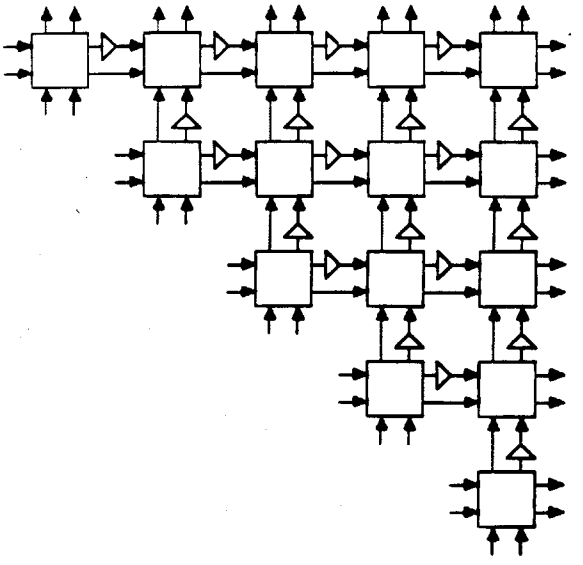


Figure 6-2: *Final Architecture for Optimal String Parenthesization*

generalized approach. Moldovan [12] and Lam and Mostow [8] have proposed the analysis of a general dependencies. In both these approaches it is necessary to *explicitly* enumerate the dependencies at every point in the domain. Out approach of using a *finite* set of matrices is more compact. In addition, Moldovan does not indicate a systematic procedure for determining the transformations, and in the Lam and Mostow approach the user is expected to supply the timing and allocation functions (or the system computes it heuristically). Li and Wah [10] and also Delosme and Ipsen [3] have proposed certain extensions to the uniform dependencies, and have also proposed optimality criteria to guide the *choice* of the timing and allocation functions. Similar criteria which result in linear constraints can be used in our approach too (as was illustrated in Sec. 6. Chen [2] has proposed a technique where the synthesis is performed by *inductively* traversing the dependency graph. Ramakrishnan et al [16] also approach the problem from a graph-theoretic perspective, and there too it is necessary to analyze the entire graph. It is not clear how there techniques can be adapted to handle infinite computations.[1] Recently Guerra and Melhem [4] have also developed a technique where the dependencies do not have to be constant. However instead of being completely arbitrary, the dependencies have to be constant in *all except one* dimension. This makes for a very awkward notation, and although the theory is adequate for some examples (including the optimal string parenthesization which has some interesting properties of data-dependent computation) the general power remains to be seen.

[1]The traditional analysis of UREs [13] can tackle this easily, by ensuring that the timing function has a component along the *ray* of the domain, and the allocation function corresponds to a *finite* projection.

References

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

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

3. 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.

4. Guerra, C. and Melhem, R. Synthesizing Non-Uniform Systolic Designs. Proceedings of the International Conference on Parallel Processing, IEEE, 1986. To appear.

5. 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.

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

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

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

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

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

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

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

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

14. Rajopadhye, S. V., Purushothaman, S. and Fujimoto, R. M. On Synthesizing Systolic Arrays from Recurrence Equations with Linear Dependencies. Proceedings, Sixth Conference on Foundations of Software Technology and Theoretical Computer Science, New Delhi, India, December, 1986. to appear.

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

16. 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.

17. 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.