Systematic Generation of Linear Allocation Functions in Systolic Array Design*

Xiaoxiong Zhong, Sanjay Rajopadhye
University of Oregon

Ivan Wong
Sun Microsystem, Inc.

CIS-TR-90-10a**
July, 1991

Department of Computer and Information Science
University of Oregon

*Supported by NSF grant MIP-8802454
** Revised version of CIS-TR-90-10

Systematic Generation of Linear Allocation Functions in Systolic Array Design*

Xiaoxiong Zhong, Sanjay Rajopadhye
Computer Science Department
University of Oregon
Eugene, Oregon 97403-1202

Ivan Wong
Sun Microsystems, Inc.
MTV 1-40, 2550 Garcia Avenue
Mountain View, CA 94043-1100

Abstract

Linear allocation functions are commonly used in mapping programs expressed as systems of recurrence equations to systolic arrays. The interconnections in a systolic array are usually required to belong to a small set of **permissible** vectors. Thus, the space of all systolic arrays that can be derived from a given system of recurrences (program) is limited, regardless of the program being mapped. By investigating the nature of this constraint, we derive upper bounds on the number of possible systolic arrays that can be derived. These bounds are surprisingly small: there can be no more than 4 linear systolic implementations of 2-dimensional recurrences, and no more than 13 (purely systolic) planar arrays for a 3-dimensional system of recurrences. We present an efficient procedure to utilize these bounds to generate all possible linear allocation functions for a given system of recurrences, and show how it may be used for the computer-aided design of optimal systolic arrays.

*Supported by NSF grant MIP-8802454. Authors' email address: [lastname]@cs.uoregon.edu

1 Introduction

Systolic arrays can be designed systematically by applying affine (or linear) transformations to algorithms that are expressed as systems of recurrences. Typically [CS83, Mol83], the design consists of three components:

1. A **recurrence** to specify the computation.
2. A **timing function** (or schedule) specifying the time instant for each computation in the recurrence.
3. An **allocation function** that maps computations to processor locations.

For the purpose of this paper, we assume that the initial algorithm is a system of **Uniform Recurrence Equations (UREs)** [KMW67, Qui87] or **Regular Iterative Algorithms (RIAs)** [Rao85] and we are interested in finding valid timing and allocation functions. Even with a generalization of UREs called **Affine Recurrences (AREs)**, there are now standard methods to transform them into a system of UREs [Raj89, Roy88]. In practice, the user is often interested in arrays which are optimal with respect to a number of criteria such as the total computation time, the number of processors and the block pipeline rate [Kun88, CR91] or even the amount of interstage data movement in a multistage systolic array. Some of these criteria, such as the total computation time, depend exclusively on the timing function. Some, such as the processor count, depend on the allocation function alone, while most others depend on a combination of the timing and allocation functions. The problem of finding an optimal (with respect to the computation time) timing function has been studied extensively [SF89, Rao85], and under some standard assumptions, can be formulated as a linear programming problem (or a sequence of linear programming problems).

For an n-dimensional recurrence, the number of valid linear allocation functions is infinite, even for a finite problem domain. This implies that it is impossible to enumerate all possible allocation functions. Hence, for optimizing performance criteria that depend

on the allocation function, researches have been forced to develop strategies for pruning the search space of allocation functions. These strategies have been specific to the particular criterion. For example, in designing arrays with a minimal processor count, Wong and Delosme [WD89] develop and utilize an upper bound of the *length* of the optimal projection vector (a linear allocation function can be uniquely represented by a projection vector). Thus the projection vectors can be generated in a sequence of nondecreasing length, and when the length exceeds the bound, the procedure can stop and claim that the best solution produced so far is the optimal one. In general, to develop such a strategy for a particular performance criterion, one must be able to systematically generate the allocation functions in an order by which one can guarantee that the optimal solution will be found. It is not always clear how to find such a strategy for any given performance criterion, nor is it usually easy to combine the strategies, if multiple criteria are being considered.

It is therefore very important to clearly understand the nature of the space of linear allocation functions, and to first prune it as much as possible *independently of the performance criterion.* In this paper, we obtain upper bounds on the number of possible allocation functions, based on the following constraint: the interconnection links of the derived arrays must belong to a (usually small, and always finite) set of **permissible interconnections.** The bounds that we obtain are surprisingly low: there can be no more than 4 linear systolic implementations of 2-dimensional recurrences, and no more than 13 planar (purely systolic) arrays for a 3-dimensional system of recurrences. If diagonal connections are not permitted the number is 9, and if eight nearest neighbors are allowed, it is 25. For an arbitrary set of permissible interconnection vectors, we also develop an algorithm to determine the set of distinct "topologies" that can be constructed from these interconnections. We then show how these bounds can be used to systematically generate all allocation functions for a given system of UREs. This is achieved by introducing a normal form for these topologies. The average time complexity of the procedure is of the same order as the bound, which is the best that we can expect to do. We conclude this introduction by formally describibg the problem and then giving an outline of the paper.

Moldovan, [Mol83] used the *permissible interconnection matrix* in a very early paper, and a number of authors have proposed similar methods. Recently, Kothari et al. have also viewed the choice of allocation function as the solution of systems of diophantine equations [KOG89]. Both these methods require manual inspection of the derived arrays to remove duplicates. We show that these correspond to precisely the *unimodular affine transformations* of their allocation functions.

1.1 Problem Definition

For the purposes of this paper, a system of UREs is completely described by a set of constant dependency vectors, $\{d_1, d_2, \ldots, d_k\}$ ($d_i \in \mathcal{Z}^n$), and a convex polyhedral domain, $\mathcal{D}$. We denote by $\Delta = [d_1 \mid d_2 \mid \ldots \mid d_k]$ the $n \times k$ matrix formed from all the dependency vectors. The timing function is determined by an integral n-vector, λ, and the allocation function is an $(n-1) \times n$ integral matrix, A. There are two constraints that A must satisfy.

- **Non-conflict:** It does not conflict with the timing function, i.e., no two points are mapped to the same processor at the same time (this implies that A must be of full row rank).
- **Dense array** The derived array must be dense i.e., every integral point in the processor space must be the image of an integral point in the index space of the problem.

Each dependency d_i of the URE is mapped to an interconnection link, $\gamma_i = Ad_i$ in the target array, and we define the **interconnection matrix**, as follows.

$$\Gamma = A\Delta = [\gamma_1 \mid \gamma_2 \mid \ldots \mid \gamma_k] \tag{1}$$

Although there are apparently $n^2 - n$ degrees of freedom in choosing A, this is not really true. Many different matrices yield arrays that are *equivalent* in that they are just a

relabeling of the processors. It is well known [RK86] that A is fully determined by a **projection vector**, u. Any two matrices, A and A' which satisfy $Au = A'u = 0$, yield arrays that are equivalent. Moreover, the non-conflict constraint can be satisfied simply by ensuring that $\lambda^T u \neq 0$. Other than this, any choice of u yields exactly one distinct array (some care must be taken to ensure that u is **reduced** (the gcd of all its elements is 1), and has a **positive leading element**). There are thus infinitely many valid allocation functions for a given system of UREs.

This approach for choosing the allocation functions does not take into account a very common, important constraint. The interconnections in the derived arrays are often required to belong to a set $\mathcal{P}$, of permissible interconnections. The following are four commonly used candidates for $\mathcal{P}$, representing respectively, the linear array, the two-dimensional mesh (four neighbors), the mesh with two diagonals (i.e., hexagonal arrays), and the mesh with eight neighbors.

$$\mathcal{P}_1 = \{0, \pm 1\} \tag{2}$$

$$\mathcal{P}_2 = \left\{ \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \pm \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \pm \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right\} \tag{3}$$

$$\mathcal{P}_3 = \left\{ \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \pm \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \pm \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \pm \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right\} \tag{4}$$

$$\mathcal{P}_4 = \left\{ \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \pm \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \pm \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \pm \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \pm \begin{bmatrix} 1 \\ -1 \end{bmatrix} \right\} \tag{5}$$

If we impose an additional constraint that for each dependency, d_i, $Ad_i = \gamma_i$ must belong to $\mathcal{P}$, we no longer have the freedom to choose *any* matrix that satisfies $Au = 0$ as our allocation function. Indeed, for many values of u, there may be no A for which $\gamma_i = Ad_i \in \mathcal{P}$ for $i = 1 \ldots k$. For others, only *some* of the matrices satisfying $Au = 0$ may yield an array with permissible interconnections. We may now state our problem as follows. Given a set $\mathcal{P}$ of permissible interconnections, develop a systematic procedure to choose

valid allocation functions (i.e., satisfying the non-conflict and dense array constraints) for a URE that will yield arrays whose interconnections belong to $\mathcal{P}$.

Our approach to tackle this problem, and the organization of the paper, is as follows. First, in Sec 2 we investigate the properties of valid interconnections for a systolic array by using integral matrix theory. We also introduce the notion of congruence and similarity relations. In Sec 3, we give a procedure to enumerate all possible "topologies" for a given permissible interconnection set, and in Sec 4 we show that the bounds for the above four common interconnection sets (i.e., P_1, P_2, P_3, P_4) are fairly small. Then, in Sec 5 we develop procedures to utilize these bounds to systematically generate all allocation functions for a given system of UREs. We show how we can use "normal forms" to reduce the time complexity of the procedure to the same order as the number of bounds we derive. Finally we discuss the implications of these results on the development of practical CAD tools for optimal systolic arrays.

2 Notation

We shall now introduce some formal notation which will be used throughout this paper. Many of the ideas are fairly well known, but it is essential to treat them rigorously in order to explain the later development. First, we extend the standard definition of unimodularity* to non-square matrices.

Definition 1 A $m \times n$ matrix U $(m \leq n)$ is said to be **e-unimodular** (for extended unimodular) if the gcd of the determinants of all its $m \times m$ submatrices is 1.

It is well known [Sch88] (pp. 47, Cor 4.1c), that a system of diophantine equations $Ux = I$ has an integral solution for *any* integral vector I, iff U is e-unimodular. Hence, the dense array constraint is satisfied iff the allocation function A is e-unimodular.

*A square matrix is unimodular if its determinant is ± 1.

Lemma 1 *The column Hermite form of an e-unimodular $m \times n$ matrix, A, is*[†] $[I_m\ 0]$.

Proof: By definition of column Hermite form there exists some integral unimodular C such that $AC = [L\ 0]$, where L is a non-negative lower triangular matrix whose diagonal entries are the unique maximal entries of the corresponding columns. Since column operations preserve the gcd of all order i subdeterminants, and A is e-unimodular, L must be unimodular. Hence all its diagonal entries are 1. Moreover, all other entries in any column are strictly less than this, i.e., 0. Hence, $L = I_m$. ∎

Lemma 2 *If A and B are two e-unimodular matrices, their product, AB is also e-unimodular.*

Proof: From Lemma 1 $B = [I_m\ 0]C^{-1}$ for some integral unimodular C. We have $AB = A[I_m\ 0]C^{-1} = [A\ 0]C^{-1}$. Hence, we need to show that $[A\ 0]$ is e-unimodular. This is obviously true, since adding any number of additional columns to an e-unimodular matrix still yields an e-unimodular matrix. ∎

Lemma 3 *For two e-unimodular $m \times n$ matrices, A and B, if there exists a $m \times m$ rational matrix, U such that $A = UB$, then U must be integral and unimodular.*

Proof: From Lemma 1, $BC = [I_m\ 0]$ for some integral unimodular matrix. Hence, $AC = UBC = U[I_m\ 0] = [U\ 0]$. Since AC is integral, U must be integral. Furthermore, AC is e-unimodular, but the only $m \times m$ submatrix of $[U\ 0]$ whose determinant is *not* zero is U. Hence U must be unimodular. ∎

We are interested in UREs whose computation graph (for a given parameter instance) is fully connected. If this were not so, the URE would describe a number of independent computations. Such a URE can always be rewritten as an equivalent one which has a connected computation graph. The following lemma gives necessary and sufficient conditions for this.

Lemma 4 *The computation graph of a URE is connected iff the dependency matrix Δ is e-unimodular.*

[†] I_i denotes the $i \times i$ identity matrix.

Proof: The dag of the computation is connected if and only if that any index point I of the computation space $\mathcal{Z}^n$ can be represented by an integral linear combination of the dependency vectors $d_1, d_2, \ldots, d_k$ (i.e., every point is connected to the origin), i.e., the following equation has an integral solution L for any $p \in \mathcal{Z}^n$.

$$[d_1 d_2 \ldots d_k]L = p$$

This is true iff $\Delta = [d_1\ d_2\ \ldots\ d_k]$ is e-unimodular [Sch88] (p.47, Corollary 4.1c). ∎

Analogously, if the set of permissible interconnections is not rich enough that we can express any integral point as a linear combination of the vectors in $\mathcal{P}$, then it is impossible to construct a dense array. Thus, we will henceforth assume that the matrix, P, whose columns are the elements of $\mathcal{P}$ is e-unimodular. The following relationships will be used to partition the interconnection matrices into classes which are essential to eliminate redundant candidates.

Definition 2 Two full row rank integral matrices, M_1 and M_2 are said to be **congruent** (denoted by $M_1 \doteq M_2$ if $UM_1 = M_2$ for some unimodular integral matrix, U.

Definition 3 Two full row rank integral matrices, M_1 and M_2 are said to be **similar** (denoted by $M_1 \simeq M_2$ if $QM_1 = M_2$ for some non-singular rational matrix, Q.

Note that both similarity and congruence are equivalence relations, and that congruence is a refinement of similarity.

2.1 Topological Equivalence

In the following, we give a mathematical property for two processor arrays to be topological equivalent. The property is fundamental for the derivation of bounds later on. First, let us recall the standard definitions of equivalence of processor arrays. A parallel architecture is described by a graph, the nodes denoting processor labels and edges denoting interconnections. Each processor has a finite number of typed I/O ports. All

edges in the graph are also typed, so any interconnection in the array is between *similarly* typed ports on two processors. Two parallel architectures are defined to be **topologically equivalent** if their graphs are isomorphic to each other.

Regular processor arrays are parallel architectures that satisfy certain constraints. The processor labels are m-dimensional index vectors (moreover, *every* coordinate is a valid processor label), and the edges are of the form $p \leftrightarrow (p + \mu_i)$ for *all* processors, p, where the μ_i's are *constant*, m-dimensional vectors. We assume (for the present) that the space of the processors lables is infinite (there are no boundary processors), and all processors have the same number (say r) of I/O ports. Since all processors have identical interconnection links, associating a type to each edge of the graph is the same as typing the μ_i's. Hence the **topology** of the array is defined by an *ordered set* of such constant vectors, or equivalently, an $m \times r$ integral matrix, $M = [\mu_1 \mid \ldots \mid \mu_r]$. We are interested in arrays that are connected, and so M must e-unimodular (using an argument similar to Lemma 4).

Theorem 1 *Two regular processor arrays, $\mathcal{A}_1$ and $\mathcal{A}_2$ with topologies M_1 and M_2 respectively, are topologically equivalent (denoted by $\mathcal{A}_1 \equiv \mathcal{A}_2$), iff M_1 and M_2 are congruent to each other.*

Proof: Let $M_1 = [\mu_1, \ldots, \mu_r]$, and $M_2 = [\mu'_1, \ldots, \mu'_r]$.

If Part: Consider the linear transformation that maps any processor, p in $\mathcal{A}_1$ to $p' = Up$. Since every edge is of the form $p \leftrightarrow (p+\mu_i)$, it is mapped to $Up \leftrightarrow U(p+\mu_i)$ i.e., $Up \Leftrightarrow Up + U\mu_i$. The range of this transformation is the entire index space (since U is unimodular), and so this represents a regular processor array with the i-th ports of any processor, p, connected to $p + U\mu_i$. Since $U\mu_i$ is precisely the i-th column of M_2, this array is $\mathcal{A}_2$.

Only If Part: The two arrays are topologically equivalent, and hence there exists an isomorphism, say f between them. We first show that f must be linear. Any edge, $p \leftrightarrow (p + \mu_i)$ in $\mathcal{A}_1$ is mapped to $f(p) \leftrightarrow f(p + \mu_i)$ in $\mathcal{A}_2$, and since f is an isomorphism, this must be the edge $f(p) \leftrightarrow f(p) + \mu'_i$. Hence $f(p + \mu_i) = f(p) + \mu'_i$. Similarly, $f(p + k\mu_i) = f(p) + k\mu'_i$ for any integer, k. Because M_1 is e-unimodular, any point p can be expressed as $\sum_{j=1}^{r} k_j \mu_j$, an integral linear combination of its

columns. Hence,

$$f(p) = f(\sum_{j=1}^{r} k_j \mu_j) = f(0 + \sum_{j=1}^{r} k_j \mu_j) = f(0) + \sum_{j=1}^{r} k_j \mu'_j$$

Hence, f must be linear and rational, and must be described by a rational $m \times m$ matrix. By Lemma 3, it must be integral and unimodular, i.e., $UM_1 = M_2$ for some integral unimodular $m \times m$ matrix U. ∎

Thm. 1 implies that any sequence of elementary row operations do not affect the topology of a regular processor array. However, elementary column operations are not permitted. Intuitively, this is so because the topology is defined as an *ordered* set of interconnections, and the order is crucial. For example, the two linear arrays $[1, 0]$ and $[0, 1]$ do not represent the same topology: a convolution array where the weights stay in the processors and the input values move is *not* the topologically the same as one where the weights move and the inputs stay.

Procedure 1

Given: Two $n \times k$ matrices, M_1 and M_2.

Output: *true* if $M_1 \doteq M_2$.

1. Determine an $n \times n$ non-singular submatrix, Γ_1 of M_1. If such a matrix does not exist, return *false*.
2. Check that the corresponding submatrix, Γ_2 of M_2 is also non-singular. If not, return *false*.
3. Determine $Q = \Gamma_2 \Gamma_1^{-1}$. If Q is not integral unimodular, return *false*.
4. If $QM_1 = M_2$ return *true*, else return *false*.

Procedure 1 above, is used to compare if two matrices are congruent. Note that similarity can be tested either by modifying the procedure (removing the test in step 3 above), or as follows. Compute the right null vector ν for the matrix, making sure that ν is chosen so that it is *reduced* and has a *positive leading element*. Then, similarity can be

determined simply by comparing the respective ν's. Another method is to use a canonical form, and compare *these* for syntactic equality. This option will be discussed later. Note that if a regular processor array is derived from a URE $\{\Delta, \mathcal{D}\}$, by an allocation function A, its topology is $\Gamma = A\Delta$. Moreover, we have the following.

Remark 1 *Two allocation functions A_1 and A_2 generate identical arrays iff the corresponding Γ_1 and Γ_2 are* **topologically equivalent**. *Indeed, $\Gamma_1 = U\Gamma_2$ iff $A_1 = UA_2$.*

3 Bounds on the Number of Allocation Functions

Our approach is based on the following simple observation. Instead of first choosing A and *deriving* Γ from it, if Γ is *given*, we can view Eqn (1) as a system of diophantine equations, and *solve* for A. This system has $(n-1) \times k$ equations (one for each element of Γ), and $n^2 - n$ unknowns in A (in fact, there are $n-1$ independent systems of equations, one for each row of A). Since n must be no greater than k for Δ to be e-unimodular, the system is fully (or over) determined, and yields a unique solution (if any) for A. Therefore, if we can enumerate the set of *all* such systems of equations that can possibly occur this will constitute an bound on the number of allocation functions for the problem. Moreover, if this set is reasonably small, we will also have an effective synthesis procedure: each such system is solved to yield an array (if the system has no integral solution, we simply move on to the next one).

As a very crude approximation, we see that for a given problem, the set of all systems of equations of the form $A\Delta = \Gamma$ is precisely the set of all $(n-1) \times k$ matrices Γ that can be formed from the elements of $\mathcal{P}$. Hence, it is easy to see that the number of possible linear allocation functions is no more than the *number* of the system of equations. However, this is fairly large ($|\mathcal{P}|^k$), and we should use additional constraints that can reduce the size of this set. In particular, we know that these Γ's must represent valid interconnection matrices, and hence must be e-unimodular. Moreover, many such matrices represent topologically equivalent arrays. This yields the following procedure to determine the set

of candidate equations of the form of (1).

Procedure 2

Given: A set $\mathcal{P}$ of permissible interconnections.

Output: A set $\mathcal{S}_k^c$ of all interconnection matrices that represent distinct arrays.

1. Construct the set, $\mathcal{S}_k$ of all e-unimodular $(n-1) \times k$ matrices whose columns belong to $\mathcal{P}$.

2. Partition $\mathcal{S}_k$ into equivalence classes under $\doteq$, and let $\mathcal{S}_k^c = \{\Gamma_i\}$ where Γ_i is a representative of each class. $\mathcal{S}_k^c$ is constructed incrementally by comparing each candidate from $\mathcal{S}_k$ with the elements of (the partially constructed) $\mathcal{S}_k^c$, and adding it if it is distinct from the ones so far (as in the sieve of Eratosthenes).

Proc 2 runs in $O(|\mathcal{P}|^n|\mathcal{S}^c|)$. For *arbitrary* $\mathcal{P}$ this is as bad as $O(|\mathcal{P}|^{2n})$ (we can construct pathological cases where $|\mathcal{S}^c| = |\mathcal{S}|$). However, most commonly occurring sets of permissible interconnections have much more regularity. Thus the set of candidate interconnection matrices is the set of e-unimodular $(n-1) \times k$ matrices whose columns are in $\mathcal{P}$, under the equivalence partition induced by congruence. Hence an upper bound on the number of possible allocation functions is simply $|\mathcal{S}_k^c|$. This bound depends on k, the number of dependency vectors in the URE. In addition, when k grows, it can grow very fast. For example, when the permissible interconnection set is $\mathcal{P}_4$ defined in Section 1.1, $|\mathcal{S}_3^c|$ is 25 but $|\mathcal{S}_4^c|$ is 349! Therefore, if we intend to use these bounds to systematically generate all the arrays, we need to further reduce the bounds.

By observing that Eqn 1 depends on Δ, the dependency matrix, we can further tighten this bound as follows. First, we notice that the dependency matrix Δ must be e-unimodular. Since e-unimodularity implies full row rank, Δ must contain an $n \times n$ non-singular submatrix, say Δ_1, (wlog, we assume that Δ_1 consists of the first n columns of Δ). Let $\Delta = [\Delta_1 \mid \Delta_2]$, and correspondingly, $\Gamma = [\Gamma_1 \mid \Gamma_2]$. Equation (1) may therefore be written as:

$$\Gamma_1 = A\Delta_1 \quad (6)$$

$$\Gamma_2 = A\Delta_2 \quad (7)$$

Since A is required to be e-unimodular, Γ_1 must also be of full rank. Note that Γ_1 and Γ_2 are not necessarily e-unimodular themselves (similarly for Δ_1 and Δ_2). But (6) is fully determined, and can yield a solution for A by itself. We can now tighten our bound as follows.

Theorem 2 *Let Γ and Γ' be two candidate interconnection matrices in $\mathcal{S}_k$ i.e., $\Gamma = [\Gamma_1 \mid \Gamma_2]$ and $\Gamma' = [\Gamma'_1 \mid \Gamma'_2]$ where Γ_1 and Γ'_1 are of full row rank. If $\Gamma_1 \simeq \Gamma'_1$ then Γ and Γ' cannot yield distinct allocation functions.*

Proof: Since $\Gamma_1 \simeq \Gamma'_1$, $\Gamma'_1 = Q\Gamma_1$ for some non-singular rational matrix, Q. Let, if possible there be two allocation functions, A and A', induced by Γ and Γ', respectively. Then, since Eqn 6 is fully determined,

$$A = \Gamma_1\Delta_1^{-1}$$
$$A' = \Gamma'_1\Delta_1^{-1} = Q\Gamma_1\Delta_1^{-1} = QA$$

and A and A' must be e-unimodular (even though Δ_1^{-1} may not be integral). Hence, by Lemma 3, Q must be integral unimodular, i.e., the two allocation functions are not distinct. ∎

Hence, the number of distinct allocation functions is no more than the number of equivalence partitions of the set of all full row rank $(n-1) \times n$ matrices whose columns are in $\mathcal{P}$, under the *similarity* relation. This is denoted by $\mathcal{S}^s$. Two points should be mentioned regarding this bound. First, for an *arbitrary* (finite) $\mathcal{P}$, it is not always possible that, given an $(n-1) \times n$ full row rank matrix whose columns belong to $\mathcal{P}$, we can always find $k-n$ additional column vectors belonging to $\mathcal{P}$ such that they form an *e-unimodular* $(n-1) \times k$ matrix. This implies that the bound may not be tight, i.e., for every element in $\mathcal{S}^s$, there may not be a candidate solution Γ. However, for many common cases such as

$\mathcal{P}_1, \ldots, \mathcal{P}_4$, this is always possible. Second, since the bound is independent of the actual dependency matrix, it is also possible that for an $(n-1) \times n$ full row rank matrix Γ_1 (partition), there is no valid allocation functions for all e-unimodular $(n-1) \times k$ matrices whose submatrix of the first n columns is similar to Γ_1. In the following, however, we will show that for the degenerate case where $\mathcal{P}$ is Z^{n-1} (i.e., any integral $n-1$ dimensional vector is a permissible interconnection), one can *always* find an e-unimodular solution to Eqn 1 for *any* $\Gamma_1 \in \mathcal{S}^s$. This indicates that our bound is tight for arbitrary $\mathcal{P}$. The following lemma first shows that any integral matrix is similar to some e-unimodular matrix (and that the similarity transformation involved is integral).

Lemma 5 *Any $(n-1) \times n$ integral matrix Γ with full row rank is similar to some unimodular matrix Γ'. Moreover the matrix, T such that $\Gamma = T\Gamma'$, is integral.*

Proof: Any integral matrix Γ with full row rank can be transformed into its Smith Form by elementary row and column operations[Sch88], i.e., $R\Gamma C = S = [D0] = D[I_{n-1}0]$, where R and C represent elementary row and column operations, and $D = \text{diag}(\delta_1, \delta_2, \ldots, \delta_{n-1})$ is a diagonal matrix. Therefore, $\Gamma = R^{-1}D[I_{n-1}0]C^{-1}$. Take, $T = R^{-1}D$, and $\Gamma' = [I_{n-1}0]C^{-1}$. Since elementary column operations do not change the gcd of the determinants of the submatrices and because $[I_{n-1}0]$ is unimodular, Γ' is unimodular too. ∎

As an aside, the above Lemma indicates that if we restrict our similarity relation to only integral transformations, then, this relation is a strict partial order. Any set of (full row rank) matrices that are congruent to each other has a subset which are *minimal* under this partial order. Moreover, if the set is *closed* under congruence, the minimal subset is a singleton. The following theorem gives us a method to determine an e-unimodular solution to $A\Delta_1 = \Gamma_1'$, for some Γ_1' similar to any candidate Γ_1.

Theorem 3 *For any candidate interconnection matrix $\Gamma_1 \in \mathcal{S}^s$, there exists $\Gamma_1' \simeq \Gamma_1$, such that $A\Delta_1 = \Gamma_1'$ has an e-unimodular solution.*

Proof: The following procedure yields the desired solution.

- Solve 6 as a system of linear (rather than diophantine) equations. So, $A = \Gamma_1 \Delta_1^{-1}$ which is, in general, rational.
- Let A' be the integral matrix obtained by multiplying A by d, the least common multiple of the denominators of all the entries of A
- By Lemma 5 there exists e-unimodular A'' such that $A' = TA''$, and T is a non-singular integral matrix.

Hence $\Gamma_1' = dT^{-1}\Gamma_1$ is similar to Γ_1 and A'' is the desired e-unimodular solution (we say $V = dT^{-1}$). Moreover, since our proof is constructive, we can determine T and A''. ∎

So far, all we have are the upper bounds for the number of allocation functions. The problem of effectively utilizing these bounds to systematically generate all the valid allocation functions still remains to be solved. This is not at all obvious, and in Sec 5, we will develop a procedure to generate all the valid allocation functions which has space and time complexity of $O(|\mathcal{S}_k^e|)$. This can be improved by using various indexing schemes, and standard data structures for searching. Before we discuss this, we first show that for the common cases defined in Sec 1.1, the bounds are surprisingly small.

4 Interconnection Matrices for the Common Cases

Applying the above procedure to the standard sets of permissible interconnections Eqns 2–4) we obtain the following results. There can be no more than 4 linear systolic arrays with nearest neighbor interconnections for any two-dimensional URE. These correspond to the topologies given as follows (see Fig 1).

$$\mathcal{S}_1^s = \{[01], [10], [11], [1-1]\}$$

For $\mathcal{P}_2$, we have the following nine possible topologies (see Fig 2).

$$\mathcal{S}_2^s = \left\{ \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \right.$$

$$\left[\begin{matrix}1&0&0\\0&1&-1\end{matrix}\right]\left[\begin{matrix}1&-1&0\\0&0&1\end{matrix}\right]\left[\begin{matrix}1&0&0\\0&1&1\end{matrix}\right]\left[\begin{matrix}1&0&1\\0&1&0\end{matrix}\right]\left[\begin{matrix}1&1&0\\0&0&1\end{matrix}\right]\Bigg\}$$

If hexagonal connections are permitted (i.e., for $\mathcal{P}_3$) the following 4 topologies are possible in addition to $\mathcal{S}_2^s$, as shown in Fig 3.

$$\mathcal{S}_3^s = \mathcal{S}_2^s \cup \left\{\left[\begin{matrix}1&0&1\\0&1&1\end{matrix}\right]\left[\begin{matrix}1&1&0\\1&0&1\end{matrix}\right]\left[\begin{matrix}1&1&0\\1&0&-1\end{matrix}\right]\left[\begin{matrix}1&0&-1\\0&1&-1\end{matrix}\right]\right\}$$

For eight nearest neighbors, i.e. $\mathcal{P}_4$ there are twelve additional interconnections as shown in Fig 4.

$$\begin{aligned}\mathcal{S}_4^s = \mathcal{S}_3^s \cup \Bigg\{&\left[\begin{matrix}1&-1&1\\1&1&0\end{matrix}\right]\left[\begin{matrix}1&-1&-1\\1&1&0\end{matrix}\right]\left[\begin{matrix}1&-1&0\\1&1&1\end{matrix}\right]\left[\begin{matrix}1&-1&0\\1&1&-1\end{matrix}\right]\\&\left[\begin{matrix}1&1&-1\\0&1&-1\end{matrix}\right]\left[\begin{matrix}-1&1&-1\\0&1&-1\end{matrix}\right]\left[\begin{matrix}0&1&-1\\1&1&-1\end{matrix}\right]\left[\begin{matrix}0&1&-1\\-1&1&-1\end{matrix}\right]\\&\left[\begin{matrix}-1&1&1\\1&0&1\end{matrix}\right]\left[\begin{matrix}-1&-1&1\\1&0&1\end{matrix}\right]\left[\begin{matrix}-1&0&1\\1&1&1\end{matrix}\right]\left[\begin{matrix}-1&0&1\\1&-1&1\end{matrix}\right]\Bigg\}\end{aligned}$$

Note that while $\mathcal{S}^s(\mathcal{P}_4)$ has only 25 elements as shown above, $\mathcal{S}_4^c(\mathcal{P}_4)$ has 349 as can be verified by running Proc 2. This grows very large as k increases and so it is very desirable to obtain a procedure that fully utilizes the tighter bounds.

5 Systematic Derivation of Valid Allocation Functions

In this section, we describe systematic procedures to utilize the bounds obtained in Sec 3 to generate all the valid allocation functions for a problem. First of all, there is a naive procedure to utilize the bounds given by the congruence relation (i.e., $\mathcal{S}_k^c$). The procedure simply tries to solve Eqn 1 for each $\Gamma \in \mathcal{S}_k^c$. Whenever an e-unimodular solution A is found,

it represents a valid allocation function. This procedure runs in time proportional to $|\mathcal{S}_k^c|$, and has space complexity of the same order of magnitude. Also note that since we have assumed that the Δ_1 is of full row rank (which is always possible since we can rearrange the dependency vectors), Γ_1 must be of full row rank. Hence, we can immediately discard those elements of $\mathcal{S}_k^c$, whose first n columns are not of full row rank. We call this set the reduced $\mathcal{S}_k^c$.

Improving this procedure to utilize our tighter bound is not as straightforward as it may seem. As we saw in Section 3, we can first partition the reduced set $\mathcal{S}_k^c$ by the similarity relation according to the submatrices formed by the first n columns. Formally, we define the partition* as follows.

Definition 4 For any two elements Γ and Γ' in the reduced set $\mathcal{S}_k^c$, they are in the same partition class iff $\Gamma_1 \simeq \Gamma_1'$ where Γ and Γ_1' are the submatrices of the first n columns of Γ and Γ' respectively. We call this partition $\mathcal{C}$.

Clearly, $|\mathcal{C}| \leq |\mathcal{S}^s|$. Also note that two matrices Γ and Γ' being in the same partition class of $\mathcal{C}$ does not not necessarily imply that they are similar to each other, let alone congruent (only their first n columns are similar). This raises a problem when we try to improve the above procedure. Suppose that $\Gamma = [\Gamma_1\ \Gamma_2]$ is the representative of a class $\mathcal{R}$ of $\mathcal{C}$, and there is no integral e-unimodular solution A for $\Gamma_1 = A\Delta_1$ (however, because Γ_1 is full row rank, there will always exist a *rational* solution). We cannot simply discard Γ as a candidate that can generate a valid allocation function. It is possible, that a *rational* transformation of A may yield a valid allocation function for some other member Γ' of $\mathcal{R}$.

Example 1 Consider the two candidate interconnection matrices (whose columns are in $\mathcal{P}_4$)

$$\Gamma_1 = \begin{bmatrix} 1 & 1 & 0 \\ -1 & 1 & 0 \end{bmatrix}\ \Gamma_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \text{ and } \Gamma_1' = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}\ \Gamma_2' = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

*A partition of a given set, S, denotes a set of disjoint subsets of S whose union is equal to S.

and $\Gamma = [\Gamma_1\ \Gamma_2]$, $\Gamma' = [\Gamma'_1\ \Gamma'_2]$. It is easy to see that $\Gamma_1 \simeq \Gamma'_1$, since

$$\Gamma_1 = T\Gamma'_1$$

where

$$T = \begin{bmatrix} 1 & 0 \\ -1 & 2 \end{bmatrix}$$

Hence, only one of them, say Γ' will be picked as the representative of the class which contains both Γ and Γ'. Now, consider a problem instance given by

$$\Delta_1 = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{ and } \Delta_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

and $\Delta = \Delta_1\Delta_2$. It is easy to verify that Δ is e-unimodular. Solving Equation $A\Delta_1 = \Delta'_1$ for A yields

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{2} & 0 \end{bmatrix}$$

which is not even integral, let alone e-unimodular. But if we use Γ as the representative of this equivalence class, the solution is

$$A = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \end{bmatrix}$$

It is easy to verify that A is indeed a valid allocation function for Γ.

It is easy to see that, if two $(n-1) \times n$ matrices $\Gamma_1 \simeq \Gamma'_1$, the corresponding *rational* allocation matrices A and A' derived by solving Eqn 6 are similar. Lemma 6 shows that we do not need to solve Eqn 6 for A for each $\Gamma \in \mathcal{R}$.

Let $\mathcal{R}$ be an arbitrary partition class of $\mathcal{C}$, Γ be the representative of $\mathcal{R}$ and Γ_1 be the first n columns of Γ. Furthermore, we know by Theorem 3 there exists $\Gamma'_1 \simeq \Gamma_1$ for which $A\Delta_1 = \Gamma'_1$ has an e-unimodular solution. Let $A_0 = V\Gamma_1\Delta_1^{-1}$ be this solution (see the proof of the theorem for details). We have the following lemma.

Lemma 6 *There is a valid allocation function for class $\mathcal{R}$ iff $A_0\Delta$ is congruent to some element in $\mathcal{R}$.*

Proof: First of all, based on the proof of Theorem 3, A_0 is e-unimodular regardless of what Γ is. To prove the *if part*, suppose $A_0\Delta$ is similar to some element Γ' in $\mathcal{R}$, i.e., $UA_0\Delta = \Gamma'$ for some unimodular matrix U. It is easy to see that UA_0 is a valid allocation function for Γ' since UA_0 is unimodular.

Conversely, suppose that there exists a $\Gamma' = [\Gamma'_1, \Gamma'_2]$ in $\mathcal{R}$ such that there exists a valid allocation function A' (e-unimodular) for it. We prove that $A_0\Delta$ is congruent to Γ'. Since $\Gamma_1 \simeq \Gamma'_1$, there exists a non-singular rational matrix T such that $A_0 = TA'$. But since A_0 and A' are e-unimodular, based on Lemma 3, T must be integral and unimodular. Thus, we know that $A_0\Delta$ is congruent to $A'\Delta = \Gamma'$. ∎

The above discussion implies that, in systematically generating allocation functions, we can not simply discard the elements in a partition class $\mathcal{R}$ represented by a single element. Therefore, it is difficult (if not impossible) to reduce the space complexity (remember that as the number of dependencies increases, $|\mathcal{S}_k^e|$ grows very fast). This difficulty rises from the fact that for any *finite* permissible interconnection set $\mathcal{P}$ is not closed under unimodular transformation. There is much room, however, to improve the time complexity of the procedure. The following is the general skeleton of the procedure to generate all valid allocation functions.

Procedure 3

Given: A set $\mathcal{P}$ of permissible interconnections, a dependency matrix Δ and a partition $\mathcal{C}$.

Output: All the valid allocation functions.

For each class $\mathcal{R} \in \mathcal{C}$,

1. Let Γ be the representative of $\mathcal{R}$.
2. Solve for A_0 as in Lemma 6.
3. If $U := \text{Test}(A_0, \mathcal{R})$ is not **false**, then UA_0 is the valid allocation function for $\mathcal{R}$, otherwise $\mathcal{R}$ does not yield a valid allocation function.

Here, Test checks whether $A_0\Delta$ is congruent to some elements in class $\mathcal{R}$ and returns **false** if it fails; otherwise, i.e., if it finds an element Γ such that $UA_0\Delta = \Gamma$ for some unimodular matrix U, it returns U.

There are two places to in Procedure 3 which need to be refined: one is how to choose a representative of the $\mathcal{R}$'s, and the other is how to implement function Test, which will dominate the time complexity of the algorithm (recall that for many common cases, $|\mathcal{C}|$, the number of the iterations in Proc. 3, is much smaller than than $|\mathcal{S}_k^c|$. In the following, we give several approaches.

The simplest approach is to randomly pick an element from $\mathcal{R}$ as the representative and implement function Test as follows: sequentially compare all the elements of $\mathcal{R}$ with $A_0\Delta = V\Gamma_1[I_{n-1}, \Delta_1^{-1}\Delta_2]$ using Proc 1. As soon as we find one successful match, we can skip the remaining elements of $\mathcal{R}$ and Test returns the corresponding element. This is safe, since in any partition class $\mathcal{R}$, at most one element can yield a valid allocation function. The worst case running time of Test is proportional to $|\mathcal{R}|$, and the average time is half of this.

The implementation of Test in the first approach can be further improved as follows. Note that any element in $\mathcal{R}$ can be written as $[Q\Gamma_1, \Gamma_2]$, for some non-singular rational matrix Q, which can be precomputed (as can Q^{-1}). To test whether $[Q\Gamma_1, \Gamma_2] \in \mathcal{R}$ is congruent to $A_0\Delta$, we have to see whether there is a unimodular transformation U such that $A_0\Delta = U[Q\Gamma_0, \Gamma_2]$. It is easy to show that $U = VQ^{-1}$, and so all we have to do is to test whether VQ^{-1} is unimodular, and whether $A_0\Delta_2 = VQ^{-1}\Gamma_2$. A necessary condition for U to be unimodular is that $\det(V) = \pm\det(Q)$. Hence we need not compare $A_0\Delta$ with *all* the elements of $\mathcal{R}$, only with those whose Q component has the same (absolute) determinant as V. Thus, if we further partition $\mathcal{R}$ into blocks with $\det(Q)$ as a key, we can reduce the time complexity of Test. However, a linear search is still required *within* each block.

It is possible to improve this to logarithmic (or constant) time, based on the following observations. For each element of $\mathcal{R}$, our procedure Test performs an operation which tests for congruence. This is a binary comparision which simply returns true or false.

If we were able to refine this to an *order relation*—if while determining whether two matrices were congruent, we were also able to determine which one was "greater"—we could then sort our candidates according to this order and use a binary search (or some hashing technique) to test for a match. We now introduce the concept of normal forms of interconnection matrices which enables us to devise such an order, and to dramatically reduce the average time complexity of the search procedure.

5.1 Normal Forms for Topologies

As described earlier, the particular candidate we use as our representative of each equivalence class under $\doteq$ is not unique, and may depend on the way our algorithm was implemented. It is useful to have a standard form which is a unique representative of each partition. Furthermore, since the sets of candidate interconnections $\mathcal{R}$ are precomputed, we can reduce the test for congruence to a simple syntactic test for equality if they are stored in such a form. We now derive such a canonical *normal form*, and show that it is unique. Since $\Gamma \doteq \Gamma'$ iff one of them can be reduced to the other using only elementary row operations, we shall obtain a normal form that is similar to Hermite normal form [Sch88] except that rather than column operations, only row operations are permitted. We call this the **row-Hermite** normal form.

Theorem 4 *Every $(n-1) \times n$ $(m < k)$ integral full row rank matrix Γ, is topologically equivalent to its row-Hermite normal form, $\Gamma^* = \begin{bmatrix} A_i & a_{i+1} & B \\ 0 & 0 & C \end{bmatrix}$, where $\begin{bmatrix} A_i & B \\ 0 & C \end{bmatrix} = S$ is a non-singular, non-negative upper-triangular matrix whose diagonal elements are the maximal entries in the corresponding column, and $\begin{bmatrix} a_{i+1} \\ 0 \end{bmatrix}$ is the $(i+1)$-th column of Γ^* (for some i)*

Proof: Our proof is based on the following induction. Suppose we have transformed Γ into the form $\begin{bmatrix} A_1 & A_2 \\ 0 & B_2 \end{bmatrix}$ where A_1 is upper triangular and with positive diagonal. We perform the following elementary row operations to B_2. Make the first column

of B_2 non-negative (if the whole column is zero, proceed to the second column). This can always be done because we can change the signs of a row by multiplying by -1. Pick the smallest positive element s on this column (say it is in row l) and repeatedly subtract row l from the other rows until all their entries in this column are less than s. Repeat this, until all except one element in this column are zeroed out (this is similar to Euclid's algorithm). Exchange this row with the first row.

Since Γ is of full row rank, the case when all the entries in the first column of B_2 are zero will occur at most once. Hence, there are at most two columns with the same "height" of nonzero elements, and all those elements on the diagonal are positive. The final matrix will have the "shape" that we desire, but its off-diagonal elements are not necessarily positive. This too can be easily accomplished by appropriate elementary row operations. The final form is Γ_*. ∎

Theorem 5 *For any two full row rank* $(n-1) \times n$ *matrices* Γ_1 *and* Γ_2, $\Gamma_1 \doteq \Gamma_2$ *iff* $\Gamma_1^* = \Gamma_2^*$.

Proof: *If Part:* $\Gamma_1 \doteq \Gamma_1^* = \Gamma_2^* \doteq \Gamma_2 \Rightarrow \Gamma_1 \doteq \Gamma_2$ (by transitivity).

Only If Part: Let $\Gamma_1 \doteq \Gamma_2$, so $\Gamma_1 = U'\Gamma_2$ for some unimodular U'. Moreover, by definition of row-Hermite normal form, $\Gamma_1 \doteq \Gamma_1^*$ (so $\Gamma_1 = U_1\Gamma_1^*$) and $\Gamma_2 \doteq \Gamma_2^*$ (so $\Gamma_2 = U_2\Gamma_2^*$), for some U_1 and U_2. Hence $\Gamma_1^* = U_1^{-1}U'U_2\Gamma_2^* = U\Gamma_2^*$ for $U = U_1^{-1}U'U_2$. We now show that this implies the U is the identity matrix, i.e., $\Gamma_1^* = \Gamma_2^*$. Let

$$\Gamma_1^* = \begin{bmatrix} A_i & a_{i+1} & B \\ 0 & 0 & C \end{bmatrix}, \Gamma_2^* = \begin{bmatrix} A'_j & a'_{j+1} & B' \\ 0 & 0 & C' \end{bmatrix}, S = \begin{bmatrix} A_i & B \\ 0 & C \end{bmatrix} \text{ and } S' = \begin{bmatrix} A'_j & B' \\ 0 & C' \end{bmatrix}$$

Without loss of generality, let $j \leq i$. We first prove that $i = j$. If it were not so, US is nonsingular, but the corresponding submatrix in Γ_2 includes $\begin{bmatrix} A'_j & a'_{j+1} \\ 0 & 0 \end{bmatrix}$ which makes S singular. Therefore, $i = j$. Now, consider $S' = US$. First of all, it is easy to see that U should be also upper triangular. Notice $u_{11}u_{22}\ldots u_{n-1,n-1} = 1$ (because U is unimodular), $u_{ll}s_{ll} = s'_{ll}$ (for $1 \leq l \leq n-1$) and $s_{ll}, s'_{ll} > 0$, we have $u_{ll} = 1$. Further, consider $s'_{l-1l} = s_{l-1l} + u_{l-1l}s_{ll}$. From $s_{l-1l} + u_{l-1l}s_{ll} \geq 0$, we have $u_{l-1l} \geq 0$ (because s_{ll} is larger than s_{l-1l}) and from $s'_{l-1l} < s'_{ll} = s_{ll}$, we have $u_{l-1l} = 0$. Now, consider $s'_{l-1l+1} = s_{l-1l+1} + u_{l-1l+1}s_{l+1l+1}$, using the same argument, we have $u_{l-1l+1} = 0$. Inductively, for $1 < k < l$, $u_{l-k,l} = 0$. Therefore, U is the identity matrix. ∎

The definition of normal forms can be trivially extended to $m \times k$ integral matrices where $m < k$, and the submatrix formed by the first $m+1$ columns is full row rank (all our

candidate interconnection matrices in the reduced $\mathcal{S}_k^c$ are of this form). We simply convert the first $m+1$ columns to row-Hermite normal form, and apply the same transformations to the remaining columns. It is also easy to show that these normal forms are unique, and Theorem 5 can also be trivially extended.

We now improve the testing function Test in Proc. 3 as follows. We index elements in a partition class $\mathcal{R}$ of $\mathcal{C}$ by their row-Hermite normal form. To test whether $A_0\Delta$ is congruent to an element Γ in $\mathcal{R}$, we first convert $A_0\Delta$ into its normal form N and based on Theorem 5, the testing of whether $A_0\Delta$ congruent to Γ is equivalent to testing whether N is syntactically equal to the normal form of Γ (which is the key of Γ). Thus, the testing problem reduces to a conventional search of an ordered list, and Test can be implemented using standard techniques. A binary search tree improves the time to logarithmic, and hashing may also be used to obtain a constant time algorithm for Test. Proc. 3 for determining all the valid allocation functions will now run in $O(|\mathcal{S}^s|)$, which is a good as we can expect to do. Note that since the reduced $\mathcal{S}_k^c$ is not closed under congruence, the normal form may not itself be a permissible interconnection. In this case we also have to explicitly store the original interconnection matrix in reduced $\mathcal{S}_k^c$ to return a correct unimodular transformation matrix U.

6 The Design of Optimal Systolic Arrays

The results presented in this paper have direct applications in the design of optimal systolic arrays. We illustrate two such examples.

One of the most important criteria in systolic array design is the number of processors of the array. In practice, we intend to minimize the processor count because processors occupy precious resources such as silicon area on chip for the array. Under the conventional framework for systolic array synthesis, choosing different projection vectors (hence different linear allocation functions) yields different processor counts. In general, the processor count of the resultant array is the number of integral points of the image of the

computation domain under the projection. It depends exclusively on the projection vector (linear allocation function) chosen for a given problem. Thus, to minimize the processor count, it is desirable to search for the linear allocation function which yields the minimal processor count. The number of valid linear allocation function for a given problem, however, is infinite even for a finite computation domain. Furthermore, since the processor count cannot be formulated as a linear function (except for two dimensional recurrences), linear programming techniques cannot be directly used as in the minimization of the total computation time (i.e. choosing the optimal timing function) [SF89].

Wong and Delosme [WD89] considered this problem and proposed a method to prune the search space of projection vectors. They proved that there exists an upper bound for the *length* of the projection vector, u (recall that u is the basis for the right-null space of the allocation matrix, A, i.e., $Au = 0$) which yields the minimal processor count. This bound depends on the "shape" of the domain of computation, and Wong and Delosme give a constructive method to find the bound. Using this, one obtains a processor-minimal systolic array by enumerating all candidate projection vectors u that are smaller than (or equal to) this upper bound, and picking the one that yields the best array among only these. Note that since the bound depends on the shape of the computation domain, it may be different for different sizes of the same problem.

In contrast with this, using our approach, one would construct all the arrays that can be derived for the given problem und using a given set of permissible interconnections (such as "pure systolic") using (Proc. 3). Then one would choose the optimal one by comparing the processor counts of each of the arrays. Because of the small bounds for the number of these valid allocation functions, we expect that such a procedure would be more efficient. Furthermore, the search space of valid linear allocation functions is independent of the problem size. One drawback of the method is that it is dependent on the particular choice of permissible interconnections (this is our premise). If a user is interested in the processor-minimal systolic array for a given problem, regardless of what the interconnections are (i.e., if $\mathcal{P} = \mathcal{Z}^{n-1}$) our procedure would be inapplicable.

In practice however, the absolute processor count may not be important. There are

many different performance criteria that may be used in systolic design, including throughput, processor utilization, block pipelining rate, etc. [Kun88, CR91], and many of these are closely related. One such measure is the processor pipelining rate α (if every processor is active in one out of every α clock cycles, the processor pipelining rate is α). It is well known that $\alpha = \lambda^T u$ (recall that λ^T is the schedule vector). Ideally, one would like an array where $\alpha = 1$, it is also well known that by clustering adjacent processors together, one can often achieve this. This problem has been studied by Zhong and Rajopadhye [ZR91] who have shown that such clustering can be deduced automatically. Clustering has the added advantage that the processor count is also reduced by a factor of α. Thus it would seeem that the real cost measure that one sould minimize while designing systolic arrays is not just V, the volume of the projection of the domain of computation, but V/α. It is not obvious how the method proposed by Wong and Delosme can be adapted to this new definition of processor count. Since our results enable the designer to systematically enumerate the (finite) space of all possible arrays that can be derived, it can be easily adapted to the new definition, and to any cost criterion (or indeed any combination of criteria) that the designer chooses.

Another example of how our results can be applied in the design of optimal multistage systolic arrays. Many practical algorithms in signal processing and numerical analysis naturally have several different phases (i.e., there are several different nested loops in the algorithm). For example, a multiplication of three matrices can be decomposed into two multiplications of two matrices. There are two approaches to the array implementation of such algorithms. The first is to design different arrays for different stages and the other is to design a single systolic array for all the different stages. In the first approach, it is desirable to minimize the interstage data movement which are caused because of the mismatch of the input and output boundaries of the arrays for different stages. In the second approach, besides the minimization of the interstage data movement, it is also desirable to minimize the difference of interconnection structures caused by the difference of the computation domains for different stages. Such minimization problems require an exhaustive search in the spaces of linear allocation functions for different stages. By using our results, an efficient procedure can be designed since the search space for each stage is

quite small.

To summarize, the results in this paper can be used to design optimal systolic arrays for *various* criteria even for some fairly complicated design problem. This is because our work shows that there exists an efficient procedure to systematically generate all possible *linear allocation functions*, which are essential to optimize many design criteria.

7 Conclusions

We have shown that the problem of determining valid linear allocation functions for a system of UREs has only finitely many solutions, if one considers the fact that the desired arrays must have interconnections that belong to a (finite) set of permissible interconnection vectors. Moreover, we have given an effective method for constructing a sufficiently tight upper bound on the *number* of distinct valid solutions that can ever be found. These bounds, for the common cases, are surprisingly small. By using the idea of normal forms, we also give a systematic procedure to enumerate all possible distinct valid allocation functions in a time complexity which is clearly the best we can do.

It should be noted that, although we have reduced the time complexity of the procedure to enumerate all distinct allocation functions to the utmost, based on the tight bound, S^s, we have to store S_k^c instead of S^s number of interconnection matrices. Therefore, our precomputed interconnection matrices are related to the number of dependencies k of the problem specification. One way to tackle this problem is to precompute S_n^c which depends only on n, the dimension of the problem domain and then generate S_k^c on the fly. This will dramatically reduce the space required and make our procedure problem independent, at the price of additional time costs. Note that the determination of S_k^c is a one-time computation, so it may be done off line for the common cases and the on-the-fly approach can be used only when the system does not have this information (and this can be stored for later use).

Notice that in the process of enumerating all the allocation functions, the equations

to be solved (i.e. the instances of Eqn 1) all involve only Δ_1, the first n columns of the dependency matrix, Δ. We expect that some properties of Δ (for example, the determinant of Δ_1) could impose additional constraints on the space of candidate Γ's. By investigating these constraints, we may be able to further reduce time and space complexities of the enumerating procedure. This is currently under investigation.

Our results also raise another question in systolic array research, namely what are permissible interconnection structures for systolic arrays? One of the possible criteria is distance. However, even for this simple criterion, it is still worthwhile to investigate different notions of the distance (say, physical distance or the minimum number of integral points). We expect that under different criteria, there will be different upper bounds on the number of distinct valid allocation functions (and of course, different enumerating procedures).

References

[CR91] Peter R. Cappello and Sanjay V. Rajopadhye. Cost measures in systolic array design. In *IEEE Pacific Rim Conference on Circuits and Systems*, Victoria, BC, Canada, May 1991.

[CS83] Peter R. Cappello and Kenneth Steiglitz. Unifying VLSI array designs with geometric transformations. In H. J. Siegel and L. Siegel, editors, *Proc. IEEE Parallel Processing Conference*, August 1983.

[KMW67] R. M. Karp, R. E. Miller, and S. Winograd. The organization of computations for uniform recurrence equations. *JACM*, 14(3):563–590, July 1967.

[KOG89] S. C. Kothari, H. Oh, and E. Gannett. Optimal designs of linear flow systolic architectures. In *International Conference on Parallel Processing*, St. Charles, Il, 1989. IEEE.

[Kun88] S. Y. Kung. *VLSI Array Processors.* Prentice Hall, 1988.

[Mol83] D. I. Moldovan. On the design of algorithms for VLSI systolic arrays. *Proceedings of the IEEE*, 71(1):113–120, January 1983.

[Qui87] Patrice Quinton. *The Systematic Design of Systolic Arrays*, chapter 9, Automata Networks in Computer Science, pages 229–260. Princeton University Press, 1987. Preliminary versions appear as IRISA Tech Reports 193 and 216, 1983.

[Raj89] Sanjay V. Rajopadhye. Synthesizing systolic arrays with control signals from recurrence equations. *Distributed Computing*, pages 88–105, May 1989.

[Rao85] Sailesh Rao. *Regular Iterative Algorithms and their Implementations on Processor Arrays.* PhD thesis, Stanford University, Information Systems Lab., Stanford, Ca, October 1985.

[RK86] Sailesh Rao and Thomas Kailath. What is a systolic algorithm. In *Proceedings, Highly Parallel Signal Processing Architectures*, pages 34–48, Los Angeles, Ca, Jan 1986. SPIE.

[Roy88] Vwani P. Roychowdhury. *Derivation, Extensions and Parallel Implementation of Regular Iterative Algorithms.* PhD thesis, Stanford University, Department of Electrical Engineering, Stanford, CA, December 1988.

[Sch88] A. Schrijver. *Theory of Integer and Linear Programming.* John Wiley and Sons, 1988.

[SF89] W. Shang and J. A. B. Fortes. On the optimality of linear schedules. *Journal of VLSI Signal Processing*, 1:209–220, 1989.

[WD89] Jiwan Wong and Jean-Marc Delosme. Optimization of the processor count for systolic arrays. Technical Report YALEU-DCS-RR-697, Computer Science Dept. Yale University, May 1989.

[ZR91] Xiaoxiong Zhong and Sanjay V. Rajopadhye. Deriving fully efficient systolic arrays by quasi-linear allocation functions. In *Parallel Architectures and Languages, Europe*, Eindhoven, the Netherlands, June 1991. Springer Verlag. An extended version is submitted to J. VLSI Signal Processing.

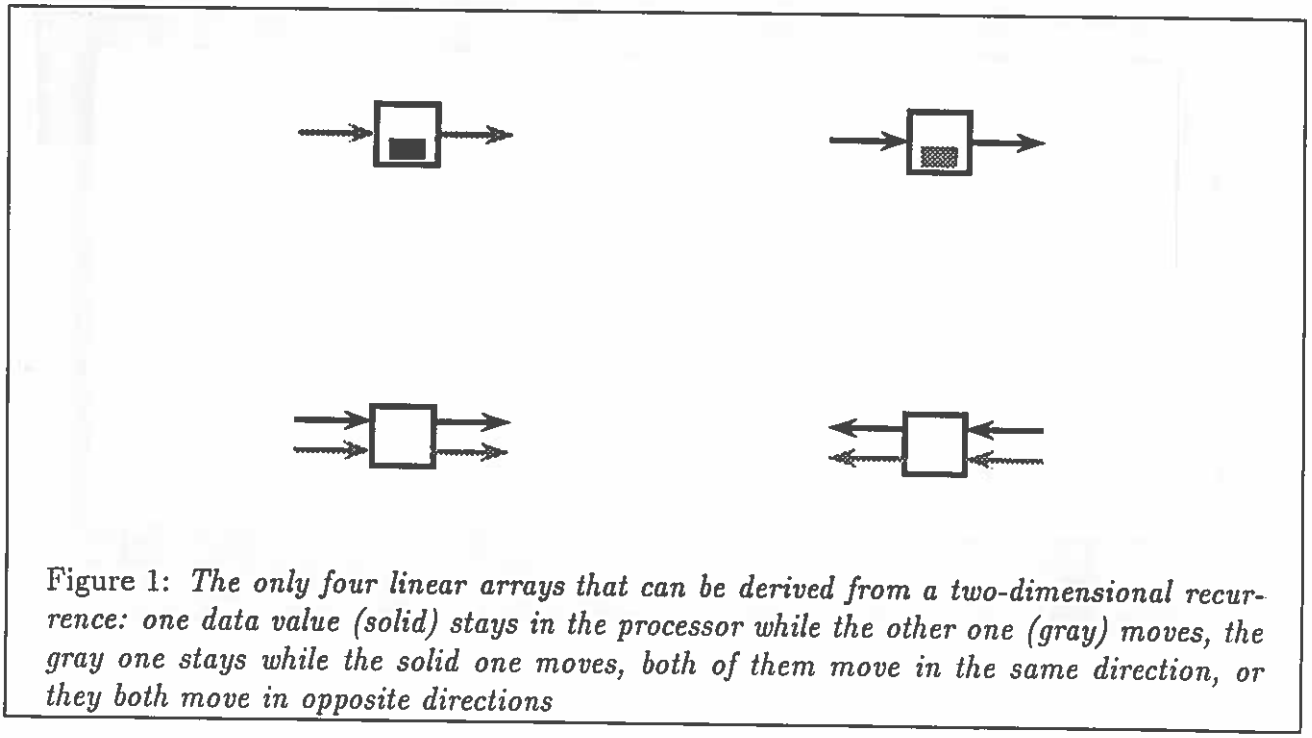

Figure 1: *The only four linear arrays that can be derived from a two-dimensional recurrence: one data value (solid) stays in the processor while the other one (gray) moves, the gray one stays while the solid one moves, both of them move in the same direction, or they both move in opposite directions*

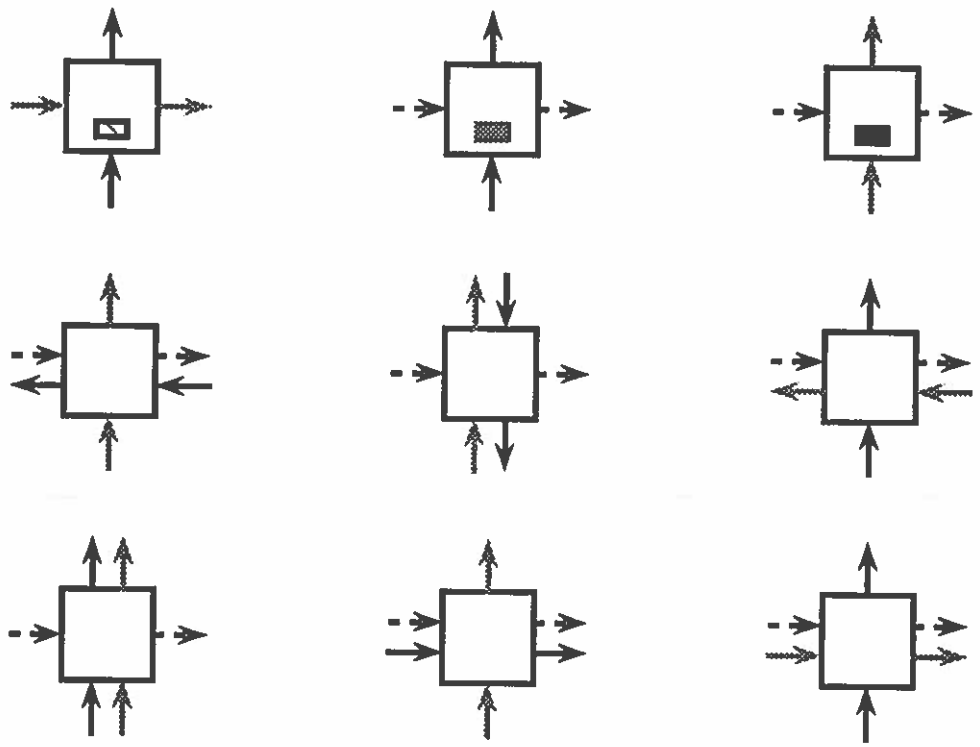

Figure 2: *All distinct two dimensional arrays with pure mesh connections*

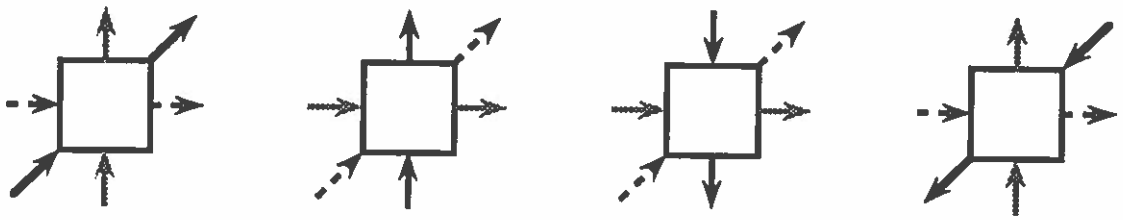

Figure 3: *Additional two-dimensional arrays if one set of diagonals are permitted ($\mathcal{P}_3$)*

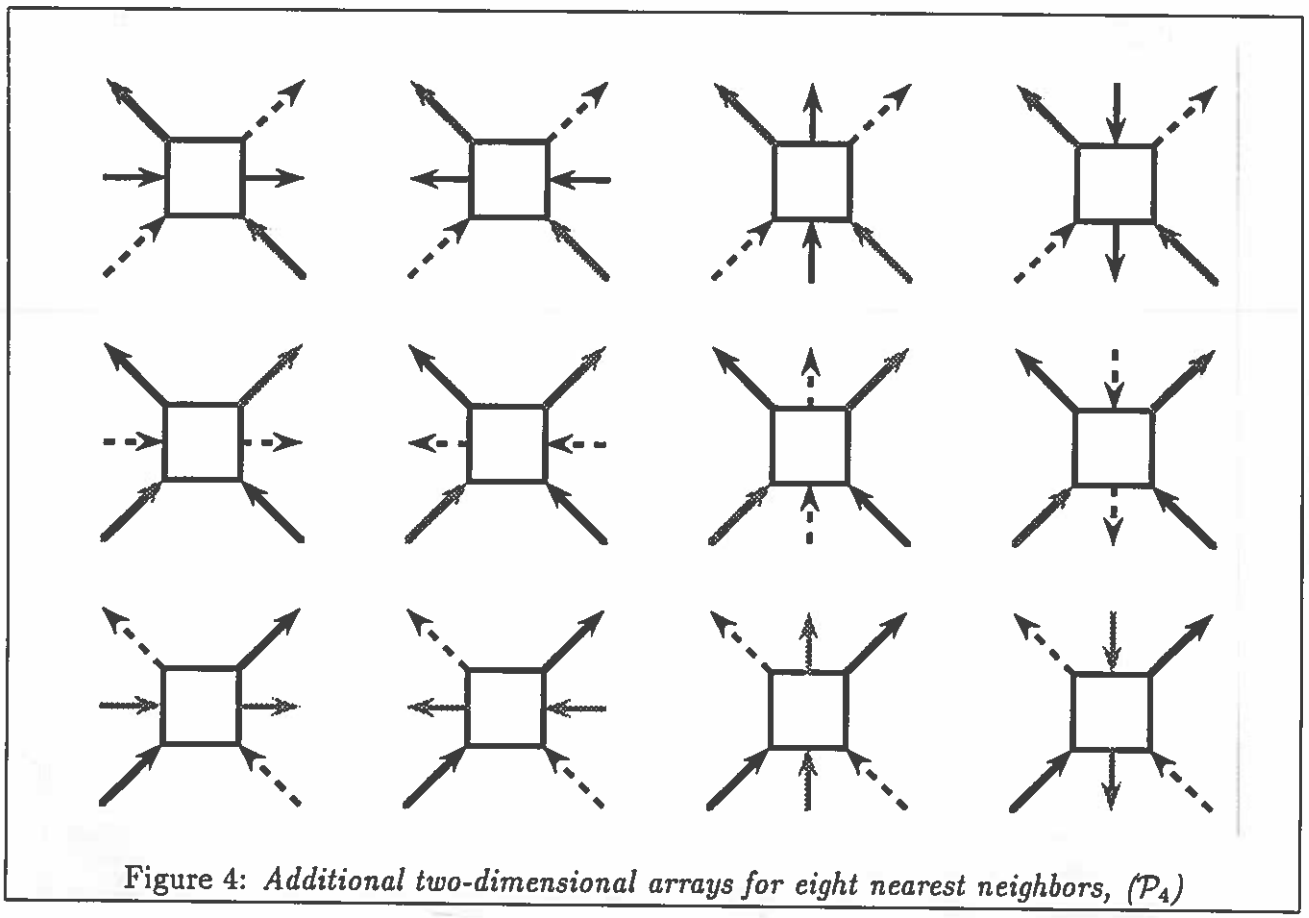

Figure 4: *Additional two-dimensional arrays for eight nearest neighbors, ($\mathcal{P}_4$)*