Reasoning About Permutations in Regular Arrays

Björn Lisper, Royal Institute of Technology
Sanjay Rajopadhye, University of Oregon

CIS-TR-91-17
August 15, 1991

Department of Computer and Information Science
University of Oregon

Reasoning about permutations in regular arrays

Björn Lisper
Telecomm. and Computer Systems
Royal Institute of Technology
P.O. Box 70043
S-100 44 Stockholm
SWEDEN
bjornl@tds.kth.se

Sanjay Rajopadhye*
Computer Science Department
120 Deschutes Hall
University of Oregon
Eugene, OR 97403-1202
USA
sanjay@cs.uoregon.edu

Abstract

We present a general theoretical model for reasoning about prescheduled data transfers in processor arrays, using a notation called *transfer relations*. We show how our model can be used for the verification of such transfer operations, and also present necessary and sufficient conditions under which they can be implemented with *purely local* control. We indicate how it is possible to automate the verification and synthesis. We illustrate our model by describing a class of data permutation networks, designed as interfaces between different systolic arrays. The permutations performed by such networks include many standard operations such as skewing, rotation, reflection, transposition, etc.

1 Introduction

Parallel computing invariably includes the transmission and reaarrangement of large quantities of data. Solutions to the communication problem range from global but not very scalable solutions, like buses, through somewhat more scalable solutions like interconnection networks [Sie85], to local but scalable point-to-point connection schemes. There is a corresponding range in the type of parallellism exploited, from large-grain computation on a few, powerful processors, to fine-grain computation on many simple units. The development of VLSI technology has stimulated research in the latter extreme end. The following properties are desirable for on-chip integration of parallel systems: *regular layout*, *local communication*, *local control*, and *I/O restricted to the boundaries*.

Together, these properties define the concepts of synchronous *systolic arrays* [KL80] and asynchronous *wavefront array processors* [Kun88]. Systolic (wavefront) array implementations have been proposed for many algorithms, especially in areas like linear algebra and image and signal processing (see for instance [MMJ89,MMU87]). There has also been a great interest in formal methods for synthesizing such arrays, especially by space-time mapping of algorithms that have a regular structure (Fortes et al. [FFW85] and Rajopadhye [RF90] present surveys of many of these methods).

*Supported by NSF grant MIP-8802454.

One question that naturally arises, is how different systolic algorithms should be interfaced. Such interfacing could take place in space, i.e. connecting several systolic arrays into one system, or in time, i.e. combining different operations taking place in the same array at different times, or a combination of both. Some pipelined interfacing operations, like transposition of streams, were studied by O'Leary [O'L87]. Optimization of a class of pipelined buffers for the interfacing of systolic arrays has been described by Wah, Aboelaze and Shang [WAS88]. Much of this work is based on informal reasoning about pipelined data flows. This paper is an attempt to provide a theoretical model for such operations, in order to address questions commonly asked in hardware verification: is a given implementation correct (w.r.t. a specification)?; are two different hardware structures equivalent?; etc. We present a formalism for reasoning about ensembles of data storage elements that can route data locally amongst themselves. The theory is general enough to describe any prescheduled data transfers, and we prove some general theorems that are of interest when designing data transfer operations in distributed systems in general. We are particularly interested in permutations that can be implemented by regular interconnections using localized control only. These include the common transformations required for interfacing systolic arrays, and a number of such arrays are presented elsewhere [RL90]. We also develop a characterization within our formalism that enables us to determine whether the permutation can be implemented with purely local control—an important consideration in obtaining VLSI implementations.

The remainder of this paper is organized as follows. In Secs 2 and 3 we develop the formalism and a characterization of when the expressions in the formalism define implementable hardware structures. Then, in Sec 4 we describe a mesh-connected array of cells that can reflect a matrix stored in it (about the central column). This array consists of independent rows of cells, each one responsible for a single row of the matrix. We show how such a row of cells is described in our formalism, and how one can formally verify that it indeed implements the desired permutation. In section 5 we develop a characterization of the "transfer profiles" that enable us to implement the arrays using only localized control. We use this characterization to determine the details of the control organization of the arrays. Finally, we present our conclusions. Because of space constraints, all proofs are omitted in this paper.

1.1 Preliminaries

In this paper we will use *binary relations* and operations on them. A binary relation R over a set A is a subset of $A \times A$. If $\langle a, b\rangle \in R$ we write aRb. All ordinary set operations are defined on relations. Here we will use union, intersection and set difference, the latter defined by $A \setminus B = \{\, a \mid a \in A, a \notin B \,\}$. The identity relation $E = \{\, \langle a, a\rangle \mid a \in A \,\}$. Powers of a relation R are recursively defined by $R^0 = E$ and $R^n = \{\, \langle a, c\rangle \mid \exists b : aR^{n-1}b \wedge bRc \,\}$ when $n > 0$. The *inverse* of R is $R^{-1} = \{\, \langle b, a\rangle \mid aRb \,\}$. R is *symmetric* if aRb always implies bRa. The *transitive closure* of R is $R^+ = \bigcup_{i=1}^{\infty} R^i$ and the *transitive and reflexive* closure of R is $R^* = \bigcup_{i=0}^{\infty} R^i = R^+ \cup E$. It is easily seen that $(R^*)^{-1} = (R^{-1})^*$ always. We say that a is a *predecessor* of b if aR^+b and an *immediate* predecessor if aRb.

For every binary relation R over A there is a corresponding directed graph $\langle A, R\rangle$, where there is an edge from a to b exactly when aRb. If aR^+b, then there is a path of length ≥ 1 from a to b in the graph. If aR^*b, then there is a path of length ≥ 0.

A binary relation R is *acyclic* iff aR^+b implies that $b \not R^+ a$ for all a, b. It is *well-founded* (noethe-

rian) if there are no infinite sequences $a_0, a_1, a_2, \ldots, a_i, \ldots$ such that $\ldots Ra_iR \ldots Ra_2Ra_1Ra_0$. Well-foundedness implies acyclicity, and if A is finite, acyclicity implies well-foundedness. R is a *tree* if it is well-founded and if there is exactly one element with no immediate predecessor and every other element has exactly one immediate predecessor. A *forest* is a union of disjoint trees. Forests can also be defined as well-founded relations where each element has at most one immediate predecessor. A path from a to b in a forest (tree) is unique.

A relation R, for which each element a has at most one immediate predecessor and at at most one immediate successor (i.e. element b such that aRb), is *multilinear*. A multilinear relation is a collection of disjoint paths. The inverse of a multilinear relation is also multilinear. A relation that is multilinear and well-founded is a forest.

For any relation R on A and elements a, b in A we define $n(R,a,b) = \{\, c \mid aR^*cR^*b \,\}$, i.e., the set of intermediate nodes in all paths from a to b. It follows that $n(R,a,b) = n(R^{-1},b,a)$. The *set of paths* in R from a to b, $p(R,a,b)$, is defined as $R|_{n(R,a,b)}$. It is immediately clear that $p(R,a,b) \subseteq R$ and that $p(R,a,b)^{-1} = p(R^{-1},b,a)$. If R is a forest then, for any a, b, $p(R,a,b)$ is multilinear (since the path from a to b, if it exists, is unique).

2 Prescheduled space-time permutations of data

We are interested in permutations of indexed data fields $a(i)$, where i belongs to some finite index set I. The permutations are then permutations of the index set. In this section we will develop the theory to describe how given permutations can be implemented by moving data in memory in a *pre-scheduled* manner. The framework can be used to verify that a certain implementation performs a given permutation correctly. We view data as being stored in memory and permutations taking place there. Thus, we assume an *address space* M. An address in M could possibly consist of several fields: in a distributed system, for instance, one field can give the processor location and another field a local memory address. Furthermore, we consider the system to be synchronous. Events in such a system can be identified with an address and a discrete time. If we take the times to be the nonnegative* integers $\{0,1,\ldots\} = N$, then the set of all events is the *address space-time* $N \times M$.

Definition 1 A function from an index set to an address space-time is a *space-time mapping* of the index set.

A space-time mapping ϕ can be seen as two functions: ϕ_m to address space and ϕ_t to time. Space-time mappings, in comparison with static mappings of indices to memory addresses, add the capability to describe dynamic behaviour, like data flowing over a communication link or in a systolic array. If i is mapped to $\phi(i) = \langle t, m \rangle$ by a space-time mapping ϕ, the interpretation is that a data item indexed by i, say $a(i)$, is to occur at memory address m at time t. Space-time mappings serve two purposes: first, to specify where and when data is input, and second where and when data is to be picked up.

Let us introduce some notation: we write $\tau + \langle t, m \rangle$ for $\langle \tau + t, m \rangle$ whenever $\tau + t \in N$ and $\tau - \langle t, m \rangle$ for $\langle \tau - t, m \rangle$ when $\tau - t \in N$. For any space-time mapping ϕ and integer τ, $\tau + \phi$ is defined

*Of course, any subset $\{t_0, t_0 + 1, \ldots\}$ of the integers will do.

by $(\tau + \phi)(i) = \tau + \phi(i)$ for all i in I. $\tau + \phi$ is well-defined whenever $\tau + \phi_t(i) \in N$ for all i. In the same way, $\tau - \phi$ is defined by $(\tau - \phi)(i) = \tau - \phi(i)$, for all i in I, when all $\tau - \phi_t(i) \in N$. For subsets S of $N \times M$, $\tau + S = \{ \tau + s \mid s \in S, \tau + s \in N \times M \}$ and $\tau - S = \{ \tau - s \mid s \in S, \tau + s \in N \times M \}$.

Next, we need a way to specify how data is propagated in (a possibly distributed) memory with time.

Definition 2 A relation $\rightarrow$ on the address space-time $N \times M$ is a *transfer relation* if it fulfils the following:

1. For all $\langle t, m \rangle$, $\langle t', m' \rangle$ in $N \times M$, $\langle t, m \rangle \rightarrow \langle t', m' \rangle \Rightarrow t < t'$. (causality)
2. The graph of $\rightarrow$ is a forest. (uniqueness of source)

The transfer relation $\rightarrow$ *connects* s to s' iff $s \rightarrow^* s'$. Transfer relations are used to model prescheduled movements of data. Intuitively, if $\langle t, m \rangle \rightarrow \langle t', m' \rangle$, then the data item stored in address m at time t becomes stored in m' at time t'. The causality property in Def. 2 rules out the transfer of data backwards in time. Since elements in forests have unique immediate predecessors, property 2 ensures that two different data items never are written to the same address at the same time. Because of this, the set of paths between any two space time events is unique, i.e. $p(\rightarrow, a, b)$ is either empty or a singleton set. When an address space-time event is connected to another event there is a transfer relation path between them, which means that data will be transferred from the first to the second in a number of steps.

Definition 3 [Translation in time] Let R be a relation on the address space-time $N \times M$. Then, for all integers τ, the relations $\tau + R$ and $\tau - R$ are defined by:

- $\langle \tau + t, m \rangle \ (\tau + R) \ \langle \tau + t', m' \rangle$ exactly when $\langle t, m \rangle R \langle t', m' \rangle$ and $\tau + t, \tau + t' \in N$.
- $\langle \tau - t, m \rangle \ (\tau - R) \ \langle \tau - t', m' \rangle$ exactly when $\langle t, m \rangle R \langle t', m' \rangle$ and $\tau - t, \tau - t' \in N$.

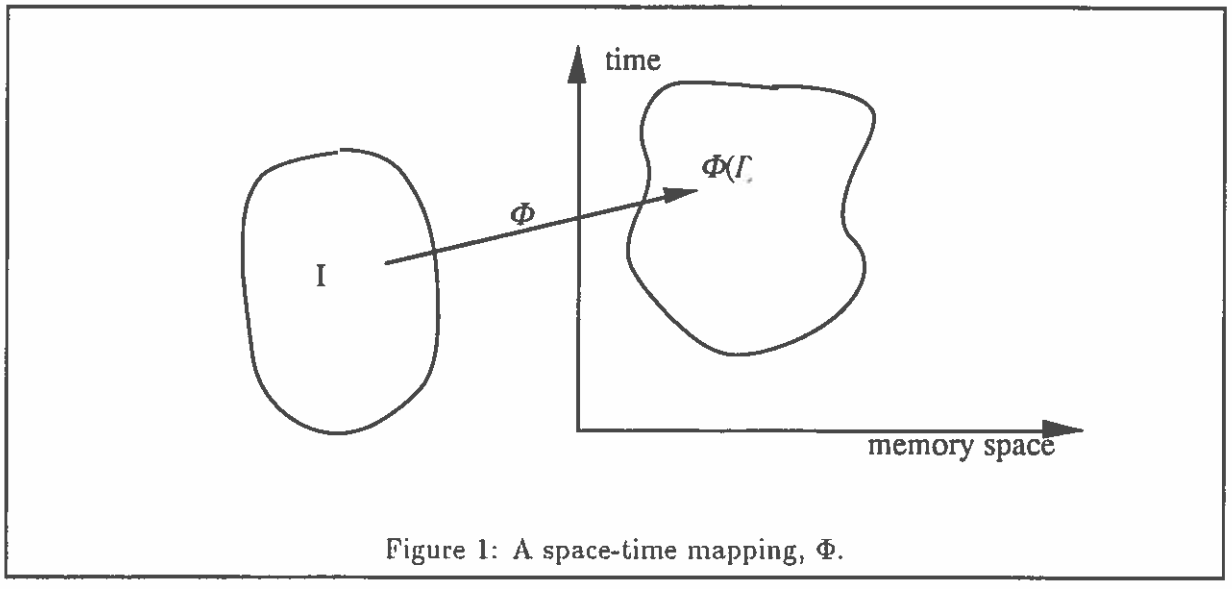

Figure 1: A space-time mapping, Φ.

Definition 4 Let π be a permutation of the index set I. Let ϕ and ψ be space-time mappings from I to an address space-time. Let $\rightarrow$ be a transfer relation on the address space-time. Let furthermore the following hold:

1. For all $i \in I$, $\phi(i) \rightarrow^* \psi(\pi(i))$ (Connectivity).
2. For any $i, j \in I$, $i \neq j$, $n(\rightarrow, \phi(i), \psi(\pi(i))) \cap n(\rightarrow, \phi(j), \psi(\pi(j))) = \emptyset$. (Disjoint paths)

Then $\rightarrow$ *implements* π with respect to ϕ and ψ.

Property 1 says that there is a (unique) path in $\rightarrow$ from $\phi(i)$ to $\psi(\pi(i))$, i.e. the value, say $a(i)$, at $\phi(i)$ will be copied to $\psi(\pi(i))$. Property 2 states that the paths between different input and outputs never overlap. Thus, a value that is input will never overwrite a value being transferred to an output through the transfer relation. Def. 4 can be visualized by a "commuting diagram" (see Fig. 2. We refer to ϕ and ψ as the input and output space-time maps, respectively, and say that ϕ, ψ and $\rightarrow$ constitute a "space-time permutation".

Property 2 allows the possibility that $s \rightarrow \phi(i)$ for some s. This may seem somewhat awkward, since a strictly operational interpretation of $\rightarrow$ would imply the value from s being transferred to $\phi(i)$ with a resulting write conflict. The problem is, however, only in the interpretation. The correct interpretation is that the data input at $\phi(i)$ overwrites whatever should have been transferred to there. We have chosen the definition here because of its mathematical simplicity. Note also, that an equivalent "strictly write conflict-free" transfer relation always can be found by simply removing all arcs of the form $s \rightarrow \phi(i)$ from $\rightarrow$. The resulting relation still implements π w.r.t. ϕ and ψ.

Proposition 1 *If a transfer relation $\rightarrow$ implements a permutation π w.r.t. ϕ and ψ, then ϕ and ψ are injective.*

Definition 5 For any two transfer relations $\rightarrow_1$, $\rightarrow_2$ on the address space-time S, $\rightarrow_1 \parallel \rightarrow_2$ is defined as $\rightarrow_1 \cup \rightarrow_2$ exactly when $s' \rightarrow_1 s$ and $s'' \rightarrow_2 s$ imply $s' = s''$ for all s, s', s'' in S.

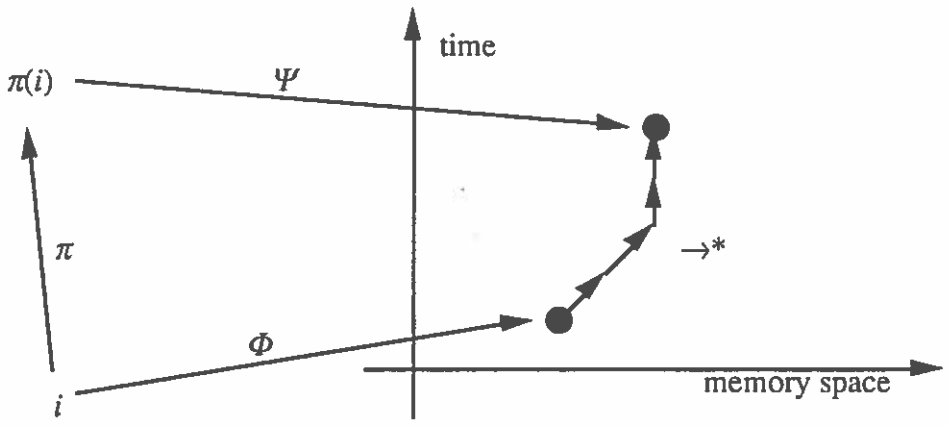

Figure 2: "Commuting diagram" for an implementation of a permutation.

It is easy to verify that $\rightarrow_1 \parallel \rightarrow_2$ really is a transfer relation whenever defined: causality implies well-foundedness, and the uniqueness of immediate predecessors follows directly from the definition. This partial operation provides a simple means for successively forming complex transfer relations from simpler ones.

We conclude this section with three theorems regarding transformations and composition of space-time permutations.

Theorem 1 *(Translation) Let $\rightarrow$ be an implementation of π w.r.t. ϕ and ψ. Assume that $\tau + \phi$ is well-defined. Then $\tau + \rightarrow$ implements π w.r.t. $\tau + \phi$ and $\tau + \psi$.*

Theorem 2 *(Inversion) Let $\rightarrow$ be a multilinear transfer relation that implements π w.r.t. ϕ and ψ. Assume that $\tau - \psi$ is well-defined. Then $\tau - \rightarrow^{-1}$ implements π^{-1} w.r.t. $\tau - \psi$ and $\tau - \phi$.*

Here, it is crucial that $\rightarrow$ is multilinear. If not, $\tau - \rightarrow^{-1}$ is not necessarily a forest. Permutations can always be implemented by multilinear transfer relations.

Theorem 3 *(Composition) Let $\rightarrow_1$ and $\rightarrow_2$ be transfer relations such that $\rightarrow_1$ implements π_1 w.r.t. ϕ and ψ, $\rightarrow_2$ implements π_2 w.r.t. ψ and ω and $\rightarrow_1 \parallel \rightarrow_2$ is defined. Assume furthermore, for all indices i, j, that $n(\rightarrow_1, \phi(i), \psi(\pi_1(i))) \cap n(\rightarrow_2, \psi(\pi_1(j))), \omega(\pi_2 \circ \pi_1(j))) = \emptyset$ when $i \neq j$. Then, $\rightarrow_1 \parallel \rightarrow_2$ implements $\pi_2 \circ \pi_1$ w.r.t. ϕ and ω.*

Theorem 3 gives general conditions for when space-time permutations can be composed. A special case, where the condition on disjointness in the theorem is fulfilled, is composition of in-place permutations that are initiated and completed at certain times for all data. Then, the transfer relations are temporally disjoint. Another example is composition of two distinct interconnection networks. In this case the transfer relations are spatially disjoint.

3 Implementable space-time permutations

Transfer relations can be used both for specifying the desired behaviour of a synchronous memory system and to describe the actual behaviour of a given system. In the latter case, there are some additional restrictions that have to be met. First, in a synchronous memory system the contents of a memory cell is copied and re-stored every clock cycle. Thus, if $\langle t, m\rangle \rightarrow \langle t', m'\rangle$ it must hold that $t' = t + 1$ when $\rightarrow$ describes such a system. Second, a memory cell could possibly not be connected to all other memory cells. Especially, we are interested in systems where there is a spatial locality and a memory cell is connected to nearest neighbours only. Spatial locality constraints often arise in distributed systems, where an address is divided into a processor address and a local address. More formally, we have the following definition:

Definition 6 An address space is *processor-decomposable w.r.t. P* if it can be written as a cartesian product $P \times L$. If $(p, l) \in P \times L$, then p is called a *processor address* and l a *local address*. Let $\leftrightarrow$ be a relation on P. The transfer relation $\rightarrow$ on $N \times (P \times L)$ is *spatially local* w.r.t. $\rightarrow$ iff it always holds that $\langle t, (p, l)\rangle \rightarrow \langle t', (p', l')\rangle$ implies that $p \leftrightarrow p'$. $\rightarrow$ is *temporally local* iff it always holds that $\langle t, m\rangle \rightarrow \langle t', m'\rangle$ implies $t' = t + 1$. $\rightarrow$ is *implementable* w.r.t. $\rightarrow$ iff it is temporally and spatially local.

Processor-decomposable address spaces model memory in distributed systems. We may call $\leftrightarrow$ a *neighbourhood* relation on P. It describes how processors (and thus addresses) may be connected. Note, that $\leftrightarrow$ may be asymmetric: then, a system with unidirectional links is modelled. It may also well be the case that $m \not\leftrightarrow m$: this models for instance a cell in a systolic array with no local storage, that must always pass its data on. We define the *distance* $d(p,p')$ (w.r.t. $\leftrightarrow$) from p to p' to be $\min(\, k \mid p \leftrightarrow^k p' \,)$, i.e. the length of a shortest path in $\leftrightarrow$ between p and p'. It follows immediately that a distance is nonnegative and that $d(p,p') = 0$ implies that $p = p'$ (i.e. $p \leftrightarrow^0 p'$). Note, that possibly $d(p,p') \neq d(p',p)$, since $\leftrightarrow$ may be asymmetric. The distance from a set of processor addresses Q to a single address p' is defined as $\min(\, d(p,p') \mid p \in Q \,)$. Also, for any processor addresses p, p' and local addresses l, l', we define $d((p,l),(p',l')) = d(p,p')$.

Note that any address space M is isomorphic to the processor-decomposable address space $M \times \{l\}$ (i.e. every address is seen as a processor with one local address l). Thus, neighbourhood relations can model locality constraints also at the lowest register level.

A locality constraint can also be imposed on the *control* of a memory system. We want to model systems where the only global control signal is the clock, and the behaviour of a processor (or cell) is completely controlled by locally propagated signals. Certain cells are *boundary* cells; they can recieve external control signals and pass them on. Denote the set of boundary cells by B. Then p can clearly not be affected by an external control signal, arriving at time t, before $t + d(B,p)$. This gives a restriction on the transfer relations that such a system can implement, since a cell in such systems must recieve a control signal to change its behaviour.

Proposition 2 *If $\rightarrow$ is implementable w.r.t. $\leftrightarrow$, then $\tau+\rightarrow$ is implementable w.r.t. $\leftrightarrow$ and $\tau-\rightarrow^{-1}$ is implementable w.r.t. $\rightarrow^{-1}$. If $\rightarrow_1$ and $\rightarrow_2$ are implementable w.r.t. $\rightarrow$, then $\rightarrow_1 \| \rightarrow_2$, whenever defined, is implementable w.r.t. $\rightarrow$.*

Corollary 1 *If $\rightarrow$ is symmetric and if $\rightarrow$ is implementable w.r.t. $\rightarrow$, then $\tau-\rightarrow^{-1}$ is implementable w.r.t. $\rightarrow$.*

Definition 7 A transfer relation $\rightarrow_1$ is *implemented* by the transfer relation $\rightarrow_2$ iff $s \rightarrow_2^* s' \Rightarrow s \rightarrow_1^* s'$ for all s, s'.

The formal model that we have developed provides a framework in which we can address the synthesis problem. We start with a transfer relation describing the desired space-time permutation and then find successively more and more "refined" transfer relations, that implement the previous ones, until finally an implementable transfer relation is reached.

Proposition 3 *Let s_1, s_2 be given. Let $\rightarrow_1$ and $\rightarrow_2$ be transfer relations, where $s_1 \rightarrow_2^* s_2$. Assume furthermore that if $s \rightarrow_2 s'$ and $s' \neq s_2$, then s' has no immediate predecessor w.r.t. $\rightarrow_1 \setminus p(\rightarrow_1, s_1, s_2)$. Then $\rightarrow = (\rightarrow_1 \setminus p(\rightarrow_1, s_1, s_2)) \| \rightarrow_2$ is defined, and $s_1 \rightarrow^* s_2$.*

In proposition 3, $\rightarrow_1 \setminus p(\rightarrow_1, s_1, s_2)$ is what remains of $\rightarrow_1$ when the path from s_1 to s_2 is removed. The proposition gives a condition for when this path can be replaced by adding a transfer relation $\rightarrow_2$ that connects s_1 to s_2.

Proposition 4 *Let s_1, s_2 be given. Let $\rightarrow_1$ be a transfer relation on the address space-time $N \times M$. Let $m \notin M$. Let $\rightarrow_2$ be a transfer relation on $N \times M \cup \{m\}$ such that:*

- $s_1 \rightarrow_2^* s_2$.
- *If* $s \rightarrow_2 s'$, *then*
  1. *either* $s = s_1$ *or* $s = \langle t, m \rangle$ *for some* t.
  2. *either* $s' = s_2$ *or* $s' = \langle t, m \rangle$ *for some* t.

Then $(\rightarrow_1 \setminus p(\rightarrow_1, s_1, s_2)) \| \rightarrow_2$ *is a transfer relation for which holds that* $s_1 \rightarrow^* s_2$.

Theorem 4 *Let the distance* d *be based on the neighbourhood relation* $\leftrightarrow$. *Consider two* p, p' *such that* $d(p,p') > 0$. *If* $d(p,p') = t' - t$, *then there exists an implementable transfer relation* $\rightarrow$ *such that* $\langle t, (p,l) \rangle \rightarrow^* \langle t', (p',l') \rangle$ *for any local addresses* l *and* l'. *If* $d(p,p') \leq t' - t$, *and if* $p \leftrightarrow p$ *or* $p' \leftrightarrow p'$, *then there also exists such a transfer relation. If* $d(p,p') > t' - t$, *then such a transfer relation does not exist.*

Theorem 4 states formally what is intuitively evident: namely that communication cannot take place in time less than n between processors that are more than n "hops" apart. It provides a fast means to check the implementability of a given space-time permutation. Since the proof of existence is constructive, it also provides a support for synthesis: once a chain of neighbour processors, short enough, is found, an implementable relation that connects the desired events can be derived.

Theorem 4 considers the communication between two events only. When several data items are to be transferred, care has to be taken so that they do not interfere. The following process ensures this. Let a finite index set $I = \{i_1, \ldots, i_n\}$, a permutation π and space-time mappings ϕ and ψ be given such that there are implementable transfer paths, for each single i, from $\phi(i)$ to $\psi(\pi(i))$. Start with the transfer relation $\rightarrow_0$ connecting $\phi(i)$ directly to $\psi(\pi(i))$ for all $i \in I$. Then, for $k = 1, \ldots, n$, replace the direct connection in $\rightarrow_{k-1}$ from $\phi(i_k)$ to $\psi(\pi(i_k))$ with an implementable path, according to proposition 3. This gives $\rightarrow_k$. The final transfer relation $\rightarrow_n$ will then be implementable and will furthermore implement π w.r.t. ϕ and ψ.

The procedure sketched above may include the addition of new memory cells. If there is no implementable path to be found between $\phi(i_k)$ and $\psi(\pi(i_k))$, a new local address can be added. By proposition 4, a path can then always be found. The new local address will typically act as a data transfer register.

Theorem 5 *Let* $M = P \times L$ *be a processor-decomposable address space. Let* I *be a finite index space with* n *elements. Let* π *be a permutation of* I *and let* ϕ, ψ *be space-time mappings such that for all* $i \in I$, $d(\phi_m(i), \psi_m(\pi(i))) \leq \psi_t(\pi(i)) - \phi_t(i)$. *If there are at least* n *local addresses in* L, *then there is an implementable transfer relation on* $N \times M$ *that implements* π *w.r.t.* ϕ *and* ψ.

An interesting issue is *optimization* with respect to time. Instead of one single output space-time mapping ψ, one could consider a whole *family* ($\tau + \psi \mid \tau + \psi$ well-defined) of mappings and minimize τ under maintained implementability and possibly other constraints, like for instance bounds on the number of local adresses. Theorem 4 thus gives a lower bound for the minimum.

In the following sections, we use the concepts defined here to describe and verify a permutation operation of an two-dimensional $n \times n$ mesh-connected processor array with locally propagated control signals. In such an array, an address consists of a local address and a two-dimensional processor address (x, y): $0 \leq x, y \leq n-1$. Every local address corresponds to a local register. Two processors (cells) (x, y), (x', y') will be adjacent (i.e. $(x, y) \rightharpoonup (x', y')$) whenever $|x - x'| + |y - y'| \leq$

1. The distance between two cells (x, y), (x', y') is simply $|x - x'| + |y - y'|$. (x, y) is a boundary cell if $x = 0$, $x = n - 1$, $y = 0$ or $y = n - 1$. A cell (x, y) then has the distance x, $n - 1 - x$, y and $n - 1 - y$ to the respective boundaries, with corresponding restrictions on the control.

4 An array for reflecting a matrix and its formal proof

We now address the *verification* problem, i.e. showing that a memory system really implements a desired permutation. This can be carried out in two steps as follows:

1. Show that the system implements a transfer relation $\rightarrow$, i.e. if $\langle t, m\rangle \rightarrow^* \langle t', m'\rangle$, then the system ensures that the data stored at m at time t is present at m' at time t'.

2. Show that $\rightarrow$ implements the desired permutation w.r.t. the given space-time mappings of the index set.

Here, we will not treat step 1 formally. Such a formalism would include a mathematical model of memory systems. Possible choices are Chen's space-time recursion equations [Che83] or Chandy's and Misra's UNITY [CM88]. The formalism should describe how transfer relations are obtained from formal system descriptions. We will instead rely on informal descriptions of the memory systems and make it viable that they implement the desired transfer relations.

Step 2 can be verified in several steps. One may start with a (probably implementable) transfer relation that describes the transfers of the memory system closely and then find successively more and more "abstract" transfer relations, that are "closer" to the desired space-time permutation. If finally a transfer relation is reached, that connects the respective inputs and outputs directly and is implemented by the original one, then the verification has succeeded (see Def. 7).

As an example, we now describe an array that performs a useful permutation in interfacing systolic arrays (see [RL90]). Given an $n \times n$ array of cells that initially has an $n \times n$ matrix stored in it, we desire to perform a "reflection" of this matrix about the central column. This is specified by the following transfer relation.

$$\langle 0, i, j, \mathrm{Acc}\rangle \rightarrow \langle T, i, n - j - 1, \mathrm{Acc}\rangle \tag{1}$$

where the input and output maps are $\phi(i, j) = \langle 0, i, j, \mathrm{Acc}\rangle$ and $\psi(i, j) = \langle T, i, j, \mathrm{Acc}\rangle$ (for some, as yet unspecified time, $T > 0$). It is easy to see that this is a transfer relation: causality is obvious, because $T > 0$; and since $[i, n - j - 1]$ is a permutation of $[i, j]$, $\rightarrow$ is a forest where each edge is a distinct tree.

The implementation is a refinement of this relation which is spatially and temporally local. All the trees in the relation can be partitioned into planes parallel to the i axis. Hence, all transfers could be restricted to the same *row* of memory cells, and therefore, we will design a *one-dimensional* array of cells which can independently perform the horizontal reflection of one row of the matrix. This is specified as follows.

$$\langle 0, i, \mathrm{Acc}\rangle \rightarrow \langle T, n - i - 1, \mathrm{Acc}\rangle \tag{2}$$

where $\phi(i) = \langle 0, i, \mathrm{Acc}\rangle$ and $\psi(i) = \langle T, i, \mathrm{Acc}\rangle$. We now informally describe the operation of a linear array that achieves this, express its behavior as a transfer relation, and then prove that this

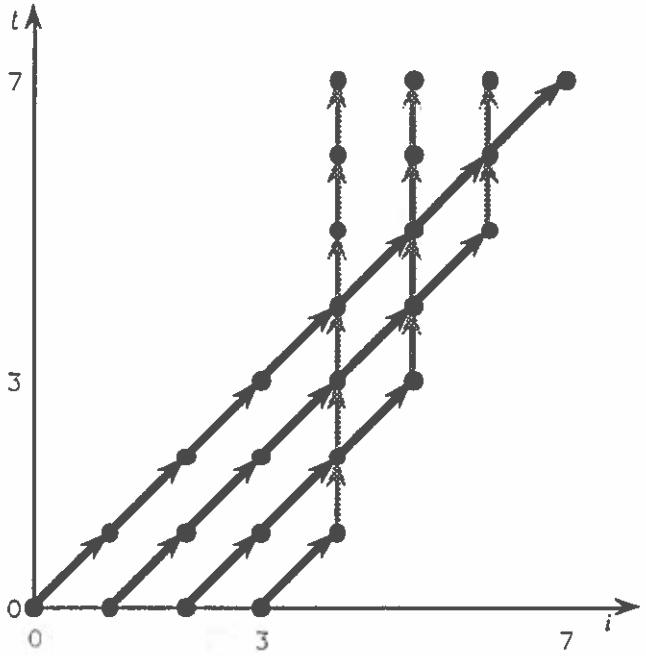

Figure 3: Transfer relation for reflecting the contents of a row of cells

transfer relation is a correct implementation of Eqn 2. Each processor has three storage elements, *Acc*, *DL* and *DR* (for Accumulator, Data-Left and Data-Right, respectively). A cell, y can read DR of its left neighbor, $y-1$, and DL of its right neighbor, $y+1$. At $t=1$, all cells in the *left* half of the row ($2y < n$) load their DR from Acc of $y-1$ (the leftmost cell does an undefined, dont-care load), and the *right* half cells load their DL from Acc of $y+1$. From then on, the left half cells copy DR of cell $y-1$ to their own DR, until $t=y$, do nothing (undefined, dont-care) until $t=n-2y-1$, at which time they copy DL of cell $y+1$ into their Acc, and then until $t=n-y-1$ they copy DL of $y+1$ to their DL. From $t=n-2y$ to $t=n$ they must also ensure that their accumulator remains unchanged. The right half cells are similar, but move data to the *left*. This yields the transfer relations shown in Fig. 3 (for the sake of clarity, only the relations transferring data from the left half to the right half are shown). It is easy to see that the data values Acc of cell y at $t=n-1$ is the value that was in Acc of $n-y-1$ at $t=0$. The transfer relation is formally defined below (Eqns 3–9).

$$(t=1 \wedge 2y < n) \Rightarrow \langle t-1, y-1, \mathrm{Acc}\rangle \text{—} \langle t, y, \mathrm{DR}\rangle \quad (3)$$

$$(t>1 \wedge t > 2y-n-2 \wedge t \leq y) \Rightarrow \langle t-1, y-1, \mathrm{DR}\rangle \text{—} \langle t, y, \mathrm{DR}\rangle \quad (4)$$

$$(t=1 \wedge 2y \geq n) \Rightarrow \langle t-1, y+1, \mathrm{Acc}\rangle \text{—} \langle t, y, \mathrm{DL}\rangle \quad (5)$$

$$(t>1 \wedge t > n-1-2y \wedge t \leq n-y-1) \Rightarrow \langle t-1, y+1, \mathrm{DL}\rangle \rightarrow \langle t, y, \mathrm{DL}\rangle \quad (6)$$

$$t = n-2y-1 \Rightarrow \langle t-1, y+1, \mathrm{DL}\rangle \rightarrow \langle t, y, \mathrm{Acc}\rangle \quad (7)$$

$$t = 2y - n + 1 \;\Rightarrow\; \langle t-1, y-1, \mathrm{DR}\rangle \rightarrow \langle t, y, \mathrm{Acc}\rangle \tag{8}$$

$$(t > 2y - n - 2 \wedge t > n - 1 - 2y \wedge t < n) \;\Rightarrow\; \langle t-1, y, \mathrm{Acc}\rangle \rightarrow \langle t, y, \mathrm{Acc}\rangle \tag{9}$$

To show that the above definition constitutes a transfer relation, we must show that it is causal and forms a forest. Causality is obvious from the definition, since the RHS of each of the equations above are of the form $\langle t-1, m\rangle \rightarrow \langle t, m'\rangle$. Moreover, the above definition is an implementation, since each of the instances are temporally and spatially local (spatial locality also follows from the fact that the relations are of the form $\langle t, y, L\rangle \rightarrow \langle t', y \pm 1, L'\rangle$). Hence $\rightarrow$ is a transfer relation if we can show that it is a forest, i.e., there do not exist two distinct points in space-time, $\langle t', y', L'\rangle$ and $\langle t'', y'', L''\rangle$, which are both related to the same point $\langle t, y, L\rangle$. To do this, we need to simply ensure that for all different equations defining a variable (i.e., equations whose RHS is of the form $\langle t, y, \mathrm{Var}\rangle$), the guards are disjoint. This can be verified by straightforward inspection of the *regions* defined by the (in)equalities: for example, the intersection of $(t = 1 \wedge 2y < n)$ and $(t > 1 \wedge t > 2y - n - 2 \wedge t \leq y)$ can be seen to be null.

We will now prove that Eqns 3–9 implement Eqn 2. We need to show that $\langle 0, i, \mathrm{Acc}\rangle \rightarrow^* \langle n-1, n-1-i, \mathrm{Acc}\rangle$. The RHS of this consists of points on the $t = n-1$ line, and since it involves Acc, this can only arise from one of Eqns 7, 8 or 9. Of these, the guard for Eqn 7 is true for $t = n-1$ only at $y = 0$, and that of Eqn 8 can be true only at $y = n-1$. At all other points, (i.e., $\langle t, y\rangle$ such that $0 < y < n-1, t = n-1$), the value of Acc must be as defined by Eqn 9. Now, within the region defined by the guard of this equation, we may draw a family of straight lines parallel to $\langle -1, 0\rangle$ (corresponding to $\langle t, y\rangle - \langle t-1, y\rangle$) and these lines correspond to transitive closure of $\rightarrow$ within the region. Because of causality, these lines *must* intersect another boundary of the region, and by some algebraic manipulation, we see that the line from $\langle n-1, j\rangle$ meets $\langle n-2j-1, j\rangle$ (if $2j < n$), or $\langle 2j-n-1, j\rangle$ (if $2j \geq n$). Moreover, the two additional points, $\langle n-1, 0\rangle$ and $\langle n-1, n-1\rangle$ also belong to these two line segments, respectively. Hence we have shown that

$$2j < n \;\Rightarrow\; \langle n-2j-1, j, \mathrm{Acc}\rangle \rightarrow^* \langle n-1, j, \mathrm{Acc}\rangle \tag{10}$$

$$2j \geq n \;\Rightarrow\; \langle 2j-n+1, j, \mathrm{Acc}\rangle \rightarrow^* \langle n-1, j, \mathrm{Acc}\rangle \tag{11}$$

Since the points $\langle n-2j-1, j\rangle$ and $\langle 2j-n+1, j\rangle$ are precisely the regions defined by the guards of Eqns 7 and 8, we may easily conclude that

$$2j < n \;\Rightarrow\; \langle n-2j-2, j+1, \mathrm{DL}\rangle \rightarrow^* \langle n-1, j, \mathrm{Acc}\rangle \tag{12}$$

$$2j \geq n \;\Rightarrow\; \langle 2j-n, j-1, \mathrm{DR}\rangle \rightarrow^* \langle n-1, j, \mathrm{Acc}\rangle \tag{13}$$

Proceeding in this manner, and reasoning about the domains of Eqns 4 and 6 (using lines of slope $\langle -1, -1\rangle$ and $\langle -1, 1\rangle$ respectively) we can show that

$$2j < n \;\Rightarrow\; \langle 1, n-j-2, \mathrm{DL}\rangle \rightarrow^* \langle n-1, j, \mathrm{Acc}\rangle \tag{14}$$

$$2j \geq n \;\Rightarrow\; \langle 1, n-j, \mathrm{DR}\rangle \rightarrow^* \langle n-1, j, \mathrm{Acc}\rangle \tag{15}$$

Once again, the regions defined by the guards of these equations make the guards of Eqn 5 and 3 true, and we therefore have

$$2j < n \;\Rightarrow\; \langle 0, n-j-1, \mathrm{Acc}\rangle \rightarrow^* \langle n-1, j, \mathrm{Acc}\rangle \tag{16}$$

$$2j \geq n \;\Rightarrow\; \langle 1, n-j-1, \mathrm{Acc}\rangle \rightarrow^* \langle n-1, j, \mathrm{Acc}\rangle \tag{17}$$

Since the union of the implicants of both these equations is a tautology, and the implied formulas are identical, this reduces to

$$\langle 0, n-j-1, \mathrm{Acc}\rangle \rightarrow^* \langle n-1, j, \mathrm{Acc}\rangle \tag{18}$$

which is precisely what is required.

5 Implementation of the arrays with local control

The example presented in the preceeding section illustrates an important aspect of our formalism. In general, we are interested in not merely arrays whose address space is processor decomposable, but in *regular*, processor decomposable address spaces. For such arrays, we seek a finite, parameterized representation of a *family* of arrays*, and the notation of Eqns 3–9 provides precisely such a representation.

Definition 8 A processor-decomposable address space $P \times L$ is said to be *regular* if the neighborhood relation, $\leftrightarrow$ on P is of the form $p \leftrightarrow p + \omega$ for *all* $p \in P$ and some *constant* vector ω.

For our discussion, we restrict ω to be either 0 or ± 1 (for a singly indexed processor space) or one of $[0,0]$, $[0,\pm 1]$, or $[\pm 1, 0]$ (for a two-dimensional processor space). Furthermore, we say that at a transfer relation is *regular implementable* if it is implementable on a regular processor-decomposable address space.

Definition 9 A transfer relation, $\rightarrow$ is said to be *homogeneous* if its address space is regular processor-decomposable, it is implementable and $\langle t, \langle p, l\rangle\rangle \rightarrow \langle t+1, \langle p+\omega, l'\rangle\rangle \Rightarrow \langle t', \langle p', l\rangle\rangle \rightarrow \langle t'+1, \langle p'+\omega, l'\rangle\rangle$, $\forall p' \in P$ and $t' \in N$. It is *piecewise homogeneous* if P can be partitioned into finitely many disjoint convex regions within which it is homogeneous.

In a homogeneous transfer relation, all the memory cells are identical, and do not change their behavior over time. Therefore the atomic transfers that they are supposed to perform can be hardwired, and this set of operations does not change. This is formalized as follows.

Lemma 1 *A homogeneous transfer relation describes an array of memory cells which operate without any control signals.*

However, there is no truly homogeneous transfer relation (other than the trivial identity permutation). This is so, because we are dealing with synchronous memory arrays which must store

*Indeed, as shown by Rao [RK86] and Quinton [Qui87], the very notion of regularity can be defined only in the context of a family of parameterized computations.

the data unchanged before performing any permutation, and continue to store the results after the final step. Hence there are at least two instants during the lifetime of any memory cell when it changes its mode of operation—once when it begins transferring data, and once when it stops.

In a piecewise homogeneous transfer relation, the operation of a memory cell does not change within any homogeneous subregion of space-time. However, at any boundary of such a region, the cell behavior must change. Such "points of inflection" in the lifetime of any cell, p are precisely the intersection of the line through p parallel to the time axis, and the boundaries of the convex regions of P. Since there are only finitely many regions, the different atomic transfers corresponding to *each* region can be stored in the cell. The critical problem is to inform the cells the precise time instants when it is to switch between the different modes of operation.

In order to achieve this, we use the fact that the regions are convex, and hence their boundaries are the conjunction of a finite number of linear guard expressions. Each such expression corresponds to a control signal, whose speed and direction can be automatically determined as follows. Consider a vector, $\sigma = \langle \sigma_P, \sigma_t \rangle$ in space-time which is parallel to the region boundary. Then, if a cell p is supposed to switch its mode of operation at time t, the inflection of cell $p + \sigma_P$ *must* occur at $t + \sigma_t$. Hence if p is somehow informed of this fact, by means of a control signal that arrives at time t, this same signal could be forwarded to cell $p + \sigma_P$ with a delay of σ_t, and serve to switch that processor too. Since σ is a constant, independent of the space-time index, the flow of the control signal corresponds to a regular transfer relation. The following theorem provides a constructive characterization of when such a control signal can be found.

Theorem 6 *A piecewise homogeneous transfer relation can be implemented using only local control iff the slope of each region boundary is at least* 45°

We see that in the reflection array of Sec 4 the transfer relation is piecewise homogeneous, with the two regions separated by the boundary $t = 2i - n + 1$. This line has a slope greater than 45° so we should find a localized control scheme. Indeed, the vector $[1, 2]$ is parallel to this line and hence a signal that travels to the right at a rate of one cell every two cycles will achieve the desired control. However, there is a problem with the $t = 0$ boundary. This has a slope zero, and hence the control cannot be achieved without a broadcast signal. A different array that achieves this permutation with purely local control has been presented elsewhere [RL90], as well as number of other arrays for common data permutations.

6 Conclusions

Traditionally, there have been two separate approaches to resoning about and deriving systolic algorithms. One is the hardware verification view as used by Chen [Che83] and others, where a formal model is set up and verification is viewed as proving the equivalence of two expressions in the formalism, much as we have done in this paper. In this view, synthesis corresponds to a refinement of the algorithm into a form that is eventually implementable as a regular localized processor array. The other approach, due to Quinton [Qui87], Rao [Rao85], Lisper [Lis89] and others is a constructive one where affine transformations are applied to algorithms that are expressed in the form of systems of recurrence equations. This paper was motivated by a desire to combine the clean formalism of the first approach with the elegant, constructive techniques used in the latter.

Note that our definition of homogeneous transfer relations is isomorphic to a subset of the Uniform Recurrence Equations (UREs) of Karp et al. [KMW67]. Moreover, the piecewise homogeneous transfer relations correspond to a subset of Conditional UREs, defined by Rajopadhye [Raj89]. Thus, we have a framework in which formal verification can be carried out. Moreover, we are able to develop constructive methods as in systems of recurrences which enable us to characterize the local control properties of the implementation.

In conclusion, we have developed a theoretical framework for describing and reasoning about prescheduled data transfers. The framework is general and can describe topologies other than meshes and operations other than permutations as well. The introduction of address space-time adds the power to describe both temporal and spatial distributions of data, which is crucial when reasoning about systolic and wavefront array architectures which operate in a pipelined fashion. We have presented a number of general results regarding transformations of transfer relations, including translation, reflection, composition of space-time permutations and about stepwise transformation of transfer relations for synthesis and verification purposes. We have also outlined a general synthesis procedure for architectures implementing a given space-time permutation.

The theoretical framework is by no means restricted to systems with local control: the actions of other types of systems like SIMD machines can be also be described. Verification of the correctness of SIMD data distribution operations could, for instance, be done by giving a transfer relation as formal semantics to each data-moving instruction, combining transfer relations arising from sequences of such instructions and then verifying that the resulting transfer relation connects the desired events.

References

[Che83] Marina C. Chen. *Space-Time Algorithms: Semantics and Methodology.* PhD thesis, California Institute of Technology, Pasadena, Ca, May 1983.

[CM88] K. Mani Chandy and Jayadev Misra. *Parallel Program Design: A Foundation.* Addison-Wesley, Reading, MA, 1988.

[FFW85] Jose A. B. Fortes, K. S. Fu, and B. S. Wah. Approaches to the design of algorithmically specified systolic arrays. In *Proceedings of ICASSP*, pages 8.9.1–4, 1985.

[KL80] H. T. Kung and C. E. Leiserson. *Algorithms for VLSI Processor Arrays*, chapter 8.3 (in 'Introduction to VLSI Systems,' Mead, C. and Conway, L.), pages 271–292. Addison-Wesley, Reading, Ma, 1980.

[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.

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

[Lis89] Björn Lisper. *Synthesis of Synchronous Systems by Static Scheduling in Space-time*, volume 362 of *Lecture Notes in Computer Science.* Springer-Verlag, Heidelberg, May 1989.

[MMJ89] John McCanny, John McWirther, and Earl Schwartzlander Jr., editors. *Systolic Array Processors.* Prentice Hall, Hertfordshire, UK, 1989.

[MMU87] Will Moore, Andrew McCabe, and Roddy Urquhart, editors. *Systolic Arrays.* Adam Hilger, Bristol, UK, 1987.

[MW85] Zohar Manna and Richard Waldinger. *The Logical Basis for Computer Programming, Volume I: Deductive Reasoning.* Addison-Wesley, Reading, MA, 1985.

[O'L87] Dianne P. O'Leary. Systolic arrays for matrix transpose and other reorderings. *IEEE Transactions on Computers*, C-36(1):117–122, January 1987.

[Qui87] Patrice Quinton. *in Automata Networks in Computer Science*, chapter 9: The Systematic Design of Systolic Arrays, 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.

[RF90] Sanjay V. Rajopadhye and Richard M. Fujimoto. Synthesizing systolic arrays from recurrence equations. *Parallel Computing*, 14:163–189, June 1990.

[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.

[RL90] Sanjay V. Rajopadhye and Björn O. Lisper. Matrix permutations on mesh-connected arrays. In *International Symposium on Circuits and Systems*, pages 2626–2629, New Orleans, LA, May 1990. IEEE CaS Society. An extended version is to appear in "Parallel Algorithms and Architectures for DSP Applications" Ed. M. A. Bayoumi, Kluwer Academic Publishers.

[Sho81] Robert Shostak. Deciding linear equalities by computing loop residues. *Journal of the Association for Computer Machinery*, 28(4):769–779, October 1981.

[Sie85] Howard Jay Siegel. *Interconnection Networks for Large-Scale Parallel Processing.* Lexington Books, Lexington, MA/Toronto, 1985.

[WAS88] Benjamin W. Wah, Mokhtar Aboelaze, and Weijia Shang. Systematic design of buffers in macropipelines of systolic arrays. *Journal of Parallel and Distributed Computing*, 5(1):1–25, February 1988.