Journal of VLSI Signal Processing, 1, 127–142 (1989)

Subspace Scheduling and Parallel Implementation of Non-Systolic Regular Iterative Algorithms*

V.P. ROYCHOWDHURY, T. KAILATH
Information Systems Laboratory, Stanford University

Received October 3, 1988

Abstract. The study of Regular Iterative Algorithms (RIAs) was introduced in a seminal paper by Karp, Miller, and Winograd in 1967. In more recent years, the study of systolic architectures has led to a renewed interest in this class of algorithms, and the class of algorithms implementable on systolic arrays (as commonly understood) has been identified as a precise subclass of RIAs. In this paper, we shall study the dependence structure of RIAs that are not systolic; examples of such RIAs include matrix pivoting algorithms and certain forms of numerically stable two-dimensional filtering algorithms. It has been shown that the so-called hyperplanar scheduling for systolic algorithms can no longer be used to schedule and implement non-systolic RIAs. Based on the analysis of a so-called computability tree we generalize the concept of hyperplanar scheduling and determine linear subspaces in the index space of a given RIA such that all variables lying on the same subspace can be scheduled at the same time. This subspace scheduling technique is shown to be asymptotically optimal, and formal procedures are developed for designing processor arrays that will be compatible with our scheduling schemes. Explicit formulas for the schedule of a given variable are determined whenever possible; subspace scheduling is also applied to obtain lower dimensional processor arrays for systolic algorithms.

1. Introduction

The study of Regular Iterative Algorithms (RIAs) was introduced in a seminal paper by Karp, Miller and Winograd [1]in 1967 who referred to RIAs (so named for certain reasons by Rao *et al.* [2]–[4] as Uniform Recurrence Equations (UREs). UREs were shown in [1] to arise naturally in finite difference approximations to systems of partial differential equations. Since then, it has been shown that RIAs appear in different areas [5], [6], and many algorithms in digital filtering (convolution, correlation, autoregressive, and moving-averaging filtering), numerical algebra (factorization, Gaussian elimination with and without pivoting, QR-factorization, SVD decomposition), discrete methods for PDEs and ODEs, graph theory (transitive closure, some coloring problems) can be reformulated as RIAs. Moreover, formal procedures for converting a class of algorithms that are specified as usual mathematical formulas to RIAs have recently been presented [7], [8].

The introduction of systolic arrays [9]–[11] in the late seventies, however, led to a renewed interest in the properties of RIAs and their parallel implementations on regular processor arrays. Early studies by a number of researchers including, Moldovan, *et al.*, [12]–[14], Quinton [15], Capello *et al.* [16], S.Y. Kung [5], showed that a special sub-class of RIAs (to be more precise, those RIAs that admit uniform linear schedules, see section 3) can be efficiently mapped on to systolic arrays. Though their several techniques for mapping the restricted class of RIAs to systolic arrays can be regarded as new, the basic analysis and scheduling techniques appear already in the work of Karp *et al.* [1].

More recently, Rao *et al.* [2]–[4], [6], studied the class of RIAs with bounded index spaces and showed that such RIAs can be efficiently implemented on a generalized class of architectures, *regular iterative arrays,* that retains most of the desirable properties of systolic arrays. Their work also led to a formal definition of systolic arrays that captures the generally accepted properties (see e.g., [5]): regularity (mostly identical processors), spatial locality (local interconnections), temporal locality (no delay-free operations, or more precisely, all combinational elements are latched, and pipelined operation (throughput independent of the order, suitably defined, of the system). Based on this definition, one

This work was supported in part by the SDIO/IST, managed by the Army Research Office under Contract DAAL03-87-K-0033 and by the Department of the Navy, Office of Naval Research under Contract N00014-86-K-0726.

can show that algorithms that do not admit *uniform affine* schedules can be termed as nonsystolic; pivoting algorithms in numerical linear algebra and certain classes of two-dimensional digital filters belong to this category.

To be more specific about the contributions of this paper, we must describe the work of Karp, *et al.* and of Rao, *et al.* in some more detail. In [1], Karp, *et al.* studied RIAs defined over *semi-infinite* index spaces and devised a procedure for determining whether a given RIA is *computable* (see Section 2). This computability analysis is based on an iterative decomposition of a graph that captures the dependences among the computations of an RIA and is referred to as the *Reduced Dependence Graph* (RDG). The decomposition of the RDG into its subgraphs leads to a tree structure that is called the *computability tree.* Karp, *et al.* showed that if the computability tree is of unit depth then the RIA admits *uniform affine* schedules (and therefore, is systolic) leading to the so-called hyperplanar schedules. If the computability tree is of depth greater than unity then they did not have any definite result about scheduling except some conjectures about asymptotic behavior. However, their analysis clearly demonstrated the existence of nonsystolic RIAs.

Rao, *et al.* [2], [4], [6] considered RIAs with only bounded index spaces, for which more explicit results can be obtained; for example, they were able to generate asymptotically optimal schedules for all such RIAs. Their scheduling procedure used the computability tree introduced by Karp, *et al.*; however, they introduced certain graph-theoretic representations that made the derivation and analyses of the computability trees more concise, and we shall adopt their representations in this paper. They showed that if the size parameter of an RIA (defined as the maximum bound on the extent of the index space) is given by N and the depth of the computability tree is l, then the I/O latency of the algorithm is $O(N^l)$; the schedules that they could determine were optimal in the sense that all the variables were scheduled in time $O(N^l)$. They also tackled the problem of implementing the RIAs when $l>1$ (i.e., nonsystolic RIAs) on regular processor arrays; however, their algebraic approach to scheduling did not permit an easy solution to the problem of determining a compatible processor array for implementing nonsystolic RIAs.

In the present paper, we present an alternative, more geometric, approach to the scheduling problem for nonsystolic RIAs that gives a simple and complete characterization of compatible processor arrays. It turns out to be a natural extension of the results for the case $l=1$, where hyperplanar scheduling is feasible. In particular, we show that if the computability tree is of depth l, then for every indexed variable x_i in the RIA one can define a linear sub-space of dimension $\geq S - l$ (where S is the dimension of the index space) such that if two variables lie on the same subspace then they can be scheduled at the same time. The explicit schedules corresponding to this subspace scheduling scheme turn out to be affine for all l; however, unlike as for systolic RIAs, the affine schedules will no longer be uniform (i.e., different indexed variables may have different coefficients), and the coefficients will be functions of the size-parameters (defined in Section 2) of the RIA.

We should note that the algebraic approach taken by Rao, *et al.* [3] may yield explicit schedules that are a constant factor better than those derived by the subspace scheduling approach. The subspace scheduling, however, more clearly brings out the combinatorial structure of the dependence graphs of RIAs and is much simpler to present. Moreover, the scheduling scheme works for RIAs defined over bounded as well as semi-infinite index space [17]; on the other hand, in the algebraic approach, the construction of the schedule is inherently dependent on the (finite) bounds of the index space. Also in our subspace scheduling scheme, the design procedure for *compatible* processor arrays can be formulated as a linear algebraic problem of determining a subspace that has no non-zero intersection with a finite number of given subspaces; this leads to a proof showing that there always exist compatible processor arrays.

An outline of the rest of the paper is as follows. Section 2 contains review of several basic concepts, such as RIAs, Reduced Dependence Graphs, valid schedules, computability of RIAs, etc.. In Section 3, we briefly describe the computability analysis procedure and the construction of the computability tree. This is followed by an exposition of the asymptotically optimal *linear subspace* scheduling strategy for scheduling RIAs defined over bounded index spaces; semi-infinite index spaces are treated in [17] and in the dissertation [18]. It turns out that, although the subspace scheduling strategy is valid for both cases, the other important details are quite different. For example, the explicit schedules determined in Section 4 of this paper for RIAs defined over bounded index spaces are not valid for RIAs defined over semi-infinite index spaces are no longer affine functions but are either polynomials or exponential functions of the index points. The sets of problems that can be handled by the two models are also different. For example the RIA for the 2-D filtering problem, discussed in detail in this paper, cannot be scheduled and implemented if it is treated as an RIA defined over semi-

infinite index space. Thus, the two cases are quite distinct and deserve separate analysis and understanding.

In Section 4, we show how to obtain explicit schedules that are compatible with the subspace scheduling scheme and also show how to design processor arrays. The issue is to demonstrate that one can always choose a processor array and devise an assignment of the computations that is compatible with the subspace scheduling. We also illustrate the relevance of the subspace scheduling procedure by applying it to schedule systolic algorithms and an RIA for performing Gaussian elimination algorithm with partial pivoting. Finally, Section 5 has some concluding remarks.

2. Basic Concepts and Terminologies

In this section some of the basic concepts in this area (see [5], [15]) are briefly reviewed and illustrated with examples, when necessary.

2.1. Regular Iterative Algorithms and Reduced Dependence Graphs

A formal definition can be found in [1], [4] and [18]. Here we shall introduce the concept via a simple example, that we shall call upon throughout the paper.

Example 1. A simple RIA.

For all tuples (i, j), $1 \leq i, j \leq N$ **do**
$x_1(i, j) = jx_1(i - 1, j + 1)x_2(i, j)$
$x_2(i, j) = ix_2(i + 1, j - 1) + x_1(i - 1, j)$

The above example displays the following (characteristic) features of an RIA:

Each variable in the RIA is identified by a label (e.g., x_1 or x_2 in example 1) and an *index vector* (e.g., $I = [i, j]^T$, in example 1).

The dependences among the variables are regular with respect to the index points. That is, if $x_1(I)$ is computed using the value of $x_2(I-d_{12})$ then the index displacement vector d_{12}, corresponding to this direct dependence, is the same regardless of the index point I.

Notice that although the direct dependences among the variables in an RIA are required to be independent of the index points, the actual computations carried out to evaluate these variables can depend on the index point. This is reflected in example 1 through the use of the values i and j in the instructions. In general, the index space $\mathbf{I}$ will be semi-infinite along certain coordinates and bounded along others. In this paper, we shall restrict ourselves to index spaces bounded in extent, and the bounds on the coordinates will be referred to as the *size parameters* of the RIA; in [19] we have dealt with the case of RIAs defined over semi-infinite index spaces.

The information regarding the parallelism in an algorithm can be captured by means of a so-called *dependence graph.* This graph has one node for each of the variables in the algorithm and an arc from node a to node b if and only if the variable b is computed using the value of a in the algorithm. The regular dependences of an RIA lead to a dependence graph with an iterative structure, which can be clearly demonstrated by embedding the dependence graph within the index space. That is, a set of V nodes is defined at every index point I and the i^{th} node represents the variable $x_i(I)$ in the RIA. As first noted by Karp, *et al.* [1] and by Waite [19], the regularity of the dependence graph of the RIA can be concisely expressed in terms of a simpler and smaller graph called the *Reduced Dependence Graph* (RDG).

The RDG of an RIA has one node for each of the indexed variables in the RIA; it has a directed arc from node x_i to node x_j, if $x_j(I)$ is computed using the value of $x_i(I - d_{ij})$ for some d_{ij}; finally, each directed arc is assigned a vector weight representing the displacement of the index point across the direct dependence. The index displacement of a directed path in the RDG can now be defined as the sum of the index displacements of the edges defining it (see figure 1 for the RDG of the RIA in example 1).

The equivalence of the DG and RDG can be formally stated by the following lemma, whose simple proof is omitted.

Lemma 1. If there is a directed path in the *DG* from $x_i(I)$ to $x_j(J)$, then there is a directed path in the RDG

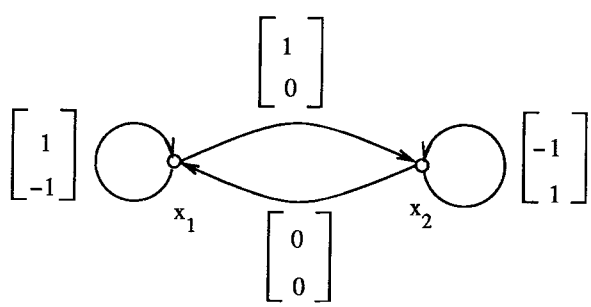

Fig. 1. The Reduced Dependence Graph (RDG) of the RIA in Example 1.

from the node x_i to the node x_j whose displacement vector is $I - J$. Conversely, if there is a directed path in the RDG from x_i to x_j with displacement vector $\mathbf{d}$, then there is a path in the dependence graph from $x_i(I)$ *to* $x_j(I + \mathbf{d})$ $\forall$ I, $I + \mathbf{d} \in \mathbf{I}$.

An RDG is a directed weighted graph and thus can be equivalently represented by its *connection matrix* (or the *node-edge incidence* matrix; see [20]) and another so-called *displacement matrix* that represents the weights along the edges. The connection matrix, C, has E columns, one for each edge in the RDG, and V rows, one for every node in the RDG. The $(i, j)^{th}$ element of C is $+1$ if edge j terminates in the node i, is -1 if edge j originates from node i, and is 0 otherwise (if edge j both originates at as well as terminates in node i, i.e., it is a self loop, then also the $(i, j)^{th}$ entry is 0). The displacement matrix D is an $S \times E$ (where S is the dimension of the index space) matrix, in which the j^{th} column is the vector weight along the j^{th} edge in the RDG.

We may note that any directed path in the RDG is a linear combination of the columns of its incidence matrix, C. For example, a path formed by the edges i, j and k can be expressed as

$$C\mathbf{q}=\mathbf{m} \text{ where } \mathbf{q}^{\mathrm{T}}=[0\ldots0\ 1\ldots1\ldots1\ldots0]$$

and the 1s in the vector $\mathbf{q}$ correspond to i^{th}, j^{th}, and k^{th} locations respectively. Since the rows of C correspond to the nodes, it is quite easy to verify that $\mathbf{m}$ will have a -1 $(+1)$ at the index of the initial (terminal) node. Notice that for a loop, the initial and terminal nodes are the same and $\mathbf{m} \equiv 0$; hence for every loop in the RDG there is a corresponding vector $\mathbf{q}$ such that $C\mathbf{q} = \mathbf{0}$. The net index displacement along a path in the RDG is just the sum of the displacements along its edges; hence for a path represented by $C\mathbf{q}=\mathbf{m}$, the displacement vector $\mathbf{d}=D\mathbf{q}$.

2.2. *Valid Schedules for an RIA*

The idea behind scheduling an RIA is to assign a time or sequencing index (which is a non-negative integer) to each variable $x_i(I)$ in the RIA such that if $x_i(I) \rightarrow x_j(J)$(to be read as '$x_j(J)$ depends on $x_i(I)$') then the sequencing index for $x_j(J)$ must be larger than that for $x_i(I)$. Thus, all the variables with the same schedule can be executed in parallel without violating any precedence constraints.

A given RIA is defined to be *computable* if it admits a valid schedule. Now, it is well established that a finite dependence graph has a valid schedule if and only if it is acyclic (see e.g., [20]). The following Theorem states the restrictions on the RDG of a computable RIA (defined over bounded index spaces).

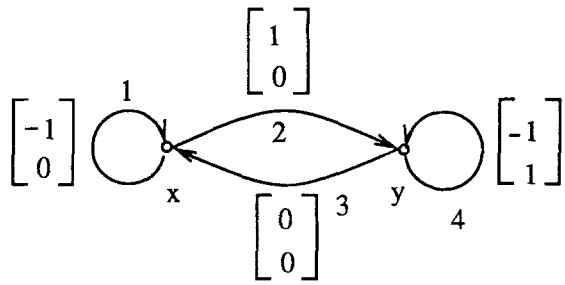

Fig. 2. The RDG of a non-computable RIA.

THEOREM 1. An RIA defined over a bounded index space is computable if and only if there is no directed cycle in the RDG whose index displacement vector is $\mathbf{0}$.

Proof. The proof follows directly from lemma 1: if there is a directed cycle in the RDG with index displacement vector $\mathbf{0}$, then there is path of the form $x_i(I) \rightarrow x_i(I)$ in the DG. Conversely, if the RIA is non-computable then there is cycle in the DG; hence by lemma 1, there is a cycle in the RDG whose displacement vector is $\mathbf{0}$. □

Example 2. Consider the RDG in figure 2. The edges 1, 2 and 3 form a cycle and their index displacement vectors add up to $\mathbf{0}$. Starting at a variable $x(I)$, one can construct a path comprised of the dependences corresponding to the arcs 1, 2, and 3 in the RDG to show that $x(I) \rightarrow x(I)$. Thus, there is a cycle in the dependence graph and the RIA is noncomputable. □

3. Computability Analysis and Sub-space Scheduling

In this section we shall briefly present the computability analysis and the subspace scheduling procedure for RIAs defined over bounded index spaces only. This will help explain the geometric scheduling procedure and will also set the foundations for the derivations of the explicit schedules as presented in Section 4.

3.1. *Computability Analysis*

Theorem 1 states that an RIA is noncomputable if and only if there is a directed cycle in the RDG with $\mathbf{0}$ index displacement vector. Hence, recalling our discussion about algebraic representation of paths in the RDG, we

can observe that if an RIA is noncomputable then there is an integral vector $\mathbf{q}$ such that

$$C\mathbf{q} = 0,\ D\mathbf{q} = 0;\ \mathbf{q} > 0 \qquad (1)$$

where C and D are the incidence and the displacement matrices of the given RDG. The vector $\mathbf{q}$ has positive entries corresponding to edges that appear in the directed cycle whose displacement vector satisifies the conditions for noncomputability stated in theorem 1. Correspondingly, an edge i will be said to *participate in a solution to (1)* if there is an integral vector $\mathbf{q}$ which satisfies (1) and has a strictly positive i^{th} entry.

We next introduce a definition and a simple result from graph theory, which will be useful for our analysis procedure.

Definition 1. A strongly connected component of a directed graph is a subgraph such that every node in the subgraph is connected to every other node in the subgraph by at least one directed path. A maximally strongly connected component of a directed graph is a strongly connected subgraph such that if any other node is included in the subgraph then the augmented subgraph is no longer strongly connected.

Lemma 2. If a directed graph is decomposed into its maximally strongly connected components then there cannot exist a directed cycle that connects two or more of these components.

Proof. Follows directly from definition 1; see e.g., [21]. Theorem 1 suggests that one needs to examine only directed cycles in the RDG for checking computability. Also, lemma 2 shows that all cycles of a graph are restricted inside its maximally strongly connected components. Thus the RDG, can be first decomposed into its maximally strongly connected components and then the computability analysis can be applied to each one of the components separately. In fact, in the rest of this paper we shall assume that the given RDG is itself a strongly connected directed graph and hence needs no further decomposition. We should note that, since the computability analysis leads to scheduling schemes, it is important to study RDGs that are not strongly connected and need to be decomposed into their strongly connected components. Important scheduling properties, however, are determined by the analysis of the strongly connected components [1], [3], [18]. Moreover, one can use schedules that are obtained from the analysis of strongly connected components to construct schedules that are valid for the whole RIA, and these procedures are discussed in more detail in [1], [4], and [18].

It may seem that to check computability of a given RIA, one only needs to check whether (1) is satisfiable. However, even if (1) has a feasible solution, the RIA may still be computable. Consider the case when there are two disjoint loops in the RDG, represented by $C\mathbf{q}_1 = 0$ and $C\mathbf{q}_2 = 0$. Let the index displacement vectors of the disjoint loops (given by $D\mathbf{q}_1$ and $D\mathbf{q}_2$, respectively) satisfy $D\mathbf{q}_1 + D\mathbf{q}_2 = 0$. If we set $\mathbf{q} = \mathbf{q}_1 + \mathbf{q}_2$ then $\mathbf{q}$ is a feasible solution to (1); however $\mathbf{q}$ does not correspond to a single directed cycle in the RDG that satisfies the conditions of Theorem 1.

Example 3. Consider the RDG in figure 1. The displacement vectors of the self loops around x_1 and x_2 add to 0. Hence, there is a solution to (1); however, the corresponding RIA is computable because the solution to (1) corresponds to *disjoint* loops. □

Thus, it is not sufficient to just check the satisfiability of (1). One solution is to consider all edges that participate in non-zero integral solutions to (1) (an edge participates in a solution only if there is a solution vector $\mathbf{q}$ whose entry corresponding to the given edge is strictly positive) and verify whether they form a single connected loop that satisfies the conditions of theorem 1. This leads to a procedure that iteratively decomposes the RDG into its subgraphs and can be described as the construction of a tree structure with the RDG at its root node.

The Computability Tree:

1. Start with the RDG as the root node and attach a tag to it, which implies that the node needs to be processed.
2. If there are no tagged nodes in the tree then STOP; the RIA is computable. Otherwise, choose any one of the tagged nodes as the node under consideration and execute the following steps:
   a. Let C^i and D^i be the connection and the displacement matrices of the graph at the node under consideration. Then, determine the edges that participate in all non-zero integral solutions to
   $$C^i\mathbf{q} = 0,\ D^i\mathbf{q} = 0. \qquad (2)$$
   If none of the edges participates then GOTO step 3. Otherwise, perform steps b through d.
   b. Form the sub-graph comprised of only those edges that participate in non-zero solutions to (2) (this sub-graph will be a collection of strongly connected components).

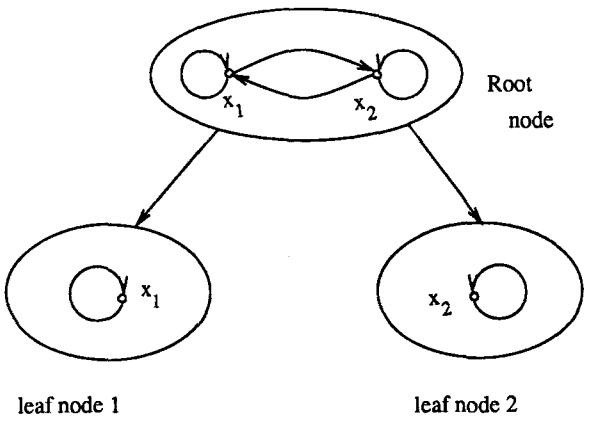

Fig. 3. The computability tree of the RIA in Example 1.

c. If the sub-graph is a single strongly connected component then STOP; the RIA is noncomputable.
d. Otherwise, mark each strongly connected component as a child of the parent node. Also, tag each of the new nodes (so that they get processed too).

3. Remove the tag on the node under consideration and GOTO step 2.

Example 4. Figure 3 shows the computability tree that corresponds to the RDG shown in Fig. 1. In the first step, the self loops 1 and 4 are the only edges that participate in solutions to (2) and get marked as the children of the root node. The self loops do not satisfy (2) and the procedure is completed. □

A proof of the above procedure can be easily constructed based on the previous discussion and for a formal proof the reader is referred to [6]. The basic idea is to observe that if an RIA is noncomputable then the directed cycle satisfying the conditions in theorem 1 would lead to a subgraph that would satisfy the test in step 2c of the computability procedure. Conversely, if in the computability procedure step 2c is never satisfied, then it is easy to see that theorem 1 would not be satisfied and the RIA would be computable.

Before we proceed to discuss the scheduling and assignment of RIAs, let us discuss some key properties of the computability tree. First, we should note that the subgraph of the RDG at each node of the computability tree is strongly connected. This is trivially true for the root node since by assumption the RDG is strongly connected. For any other node, the subgraph is comprised of edges that participate in a solution to (2). Any solution $\mathbf{q}$ to (2) satisfies $C\mathbf{q} = 0$; it can be shown [18] that such a $\mathbf{q}$ corresponds to a connected loop or a collection of disjoint loops. Hence, each of the sub-graphs generated in step 2b. of the procedure corresponds to a bunch of connected loops; in other words each subgraph is a strongly connected component.

3.1.1 Properties of the Nodes in the Computability Tree
If a node of the computability tree is a leaf node (i.e., it does not have any descendent nodes), then the corresponding subgraph has no edges that participate in a non-zero solution to (2). In other words, there is no non-zero integral solution to

$$C\mathbf{q} = 0,\ D\mathbf{q} = 0;\ \mathbf{q} \geq 0 \tag{3}$$

where C and D are, respectively, the connection and displacement matrices of the subgraph at the node. Since the above inequality is homogeneous, one can show that if there is a rational solution to (3) then it will also have an integral solution. Hence, one can treat (3) as a simple linear program rather than an integer linear program.

Lemma 3. Let C and D be the connection and displacement matrices of the subgraph corresponding to a leaf node in the computability tree. Then, the objective function of the following linear program is bounded (in fact = 0):

$$\textbf{Maximize } H^T\mathbf{q} \textbf{ such that}$$
$$C\mathbf{q} = 0,\ D\mathbf{q} = 0;\ \mathbf{q} \geq 0 \tag{4}$$

where $H^T \geq [1\ 1\ \ldots\ 1]$. Moreover, the dual linear program given by

$$\textbf{Minimize } 0 \textbf{ such that}$$
$$\Gamma^T C + \Lambda^T D \geq H^T \tag{5}$$

has a feasible solution.

Proof. If $\mathbf{q}$ is a solution to (3), then $\alpha\mathbf{q}$ is also a solution $\forall\alpha \geq 0$. Therefore, $H^T\mathbf{q}$ is bounded if and only if the only solution to (3) is $\mathbf{q} = 0$. Since the node under consideration is a leaf node, we are assured that the only feasible solution to (4) is $\mathbf{q} = 0$. Hence, the objective function of (4) is 0. □

Now, by Farkas lemma [20], the dual linear program is feasible if and only if the objective function of the primal is bounded. Hence, the dual program of (4) as given by (5) is feasible.

The satisfiability of the inequalities of the form (5) will play a major role in scheduling a given RIA.

The above lemma can be generalized for interior nodes by appropriately redefining the vector H^T. If a node of the computability tree is not a leaf node then the subgraph of the RDG at the node has two types of edges. The first type of edges do not participate in non-zero solutions to (2) and do not appear in the subgraphs of the children nodes. The second type of edges correspond to those that participate in non-zero solutions to (2); that is there is $\mathbf{q} > 0$ that satisfies (2) and its entries corresponding to these edges are strictly positive. A generalization of lemma 3 is possible by defining a vector $H_i^T = [h_1\ h_2\ \ldots\ h_E]$ where h_i is defined as follows:

$$h_i = \begin{cases} 1 \text{ if the } i^{th} \text{ edge does not participate in non-zero solution to (2)} \\ 0 \text{ if the } i^{th} \text{ edge participates in non-zero solution to (2)} \end{cases} \quad (6)$$

By construction, $H_i^T\mathbf{q} = 0 \forall\ \mathbf{q}$ satisfying (2). Hence, the objective function of the following linear program is bounded (in fact = 0)

$$\textbf{Maximize } H_i^T\ \mathbf{q} \textbf{ such that } C^i\mathbf{q} = 0,\ D^i\mathbf{q} = 0;\ \mathbf{q} \geq 0 \quad (7)$$

where H_i is defined as in (6). Also the dual of the above linear program, given by

$$\textbf{Minimize } 0 \textbf{ such that } \Gamma_i^T C^i + \Lambda_i^T D^i \geq H_i^T, \quad (8)$$

always has a feasible solution. If the i^{th} edge (say from x_i to x_j and with index displacement vector=$\mathbf{d}$) appears in one of the children nodes of the node under consideration, then by definition, the corresponding entry in H_i^T is 0. Hence, the inequality corresponding to the i^{th} edge in (8) would be of the form

$$\gamma_{xj} - \gamma_{xi} + \Lambda^T\mathbf{d} \geq 0.$$

The following lemma shows that the above inequality must necessarily be a strict equality.

Lemma 4. All inequalities in (8) where the right hand side is 0 are binding, i.e., the $\geq$ relationship is satisfied with equality.

Proof. We shall refer to [17] and [18]. □

3.2 Optimal Linear Subspace Scheduling of RIAs

We shall first deal with the case when the computability tree is just the root node (that is of unit depth, $l = 1$), and then we shall generalize our results to the case when computability trees have $l > 1$. The results for $l = 1$ are due to Karp, et al.; however, our presentation is slightly different and becomes more useful in the context of subspace scheduling for $l > 1$.

3.2.1 Computability Trees of Unit Dept (Systolic RIAs) Let C and D be the connection matrix and the displacement matrix of the RDG. Then it follows from the computability tree construction procedure that the tree would be of unit depth if and only if there is no positive solution to $C\mathbf{q} = 0$ and $D\mathbf{q} = 0$. The root node (containing the RDG itself) becomes a leaf node and the results of lemma 3 apply to the RDG of the given RIA. Thus, by lemma 3 there always exist vectors $\Gamma = [\gamma_{x_1}\ \gamma_{x_2} \ldots \gamma_{x_V}]$ and $\Lambda^T = [\lambda_1\ \lambda_2 \ldots \lambda_S)$, such that

$$\Gamma^T C + \Lambda^T D \geq [1\ 1 \ldots 1]. \quad (9)$$

Now, the condition for a schedule to be valid is that if two computations are assigned the same schedule then there should be no precedence relation between the two. The following result shows that indexed variables with the same label (say x_i) that are on the null space of the vector Λ do not have dependences among themselves.

THEOREM 2. If $\Lambda^T(I - J) = 0$, then there can be no directed path from a variable $x_i(I)$ to $x_i(J)$.

Proof. If there is a directed path from $x_i(I)$ to $x_i(J)$, then by lemma 1, there must be a loop in the RDG whose displacement is $(J - I)$. That is, $\exists\ \mathbf{q} > 0$ such that $C\mathbf{q} = 0$ and $D\mathbf{q} = (J - I)$. However, multiplying both sides of (9) by the vector $\mathbf{q}$ we get, $\Gamma^T C\mathbf{q} + \Lambda^T D\mathbf{q} \geq [1\ 1 \ldots 1]\ \mathbf{q}$. Now, $C\mathbf{q} = 0$, $D\mathbf{q} = (J - I)$ and $[1\ 1 \ldots 1]\mathbf{q} > 0$; hence $\Lambda^T(J - I) > 0$. □

Therefore, for every variable x_i, we can schedule all $x_i(I)$ lying on the same hyperplane defined by $\Lambda^T I + \delta_{xi}$ (where δ_{xi} is a constant to be determined) at the same time step. δ_{xi} should be such that the precedence relations are satisfied, i.e., if $x_i(I)$ depends on $x_j(I - \mathbf{d})$ then we should have

$$(\Lambda^T I + \delta_{x_i}) - (\Lambda^T J + \delta_{x_j}) > 0 \quad (10)$$

This can be assured by setting $\delta_{x_i} = \gamma_{x_i}$ where $\Gamma^T = [\gamma_{x_1}\gamma_{x_2} \ldots \gamma_{x_V}]$ is a vector satisfying (9); a simple verification is as follows: if there is a dependence of the form $x_i(I) \leftarrow x_j(I-\mathbf{d})$ (corresponding to an edge in the RDG) then

$$S(x_i(I)) - S(x_j(I - \mathbf{d})) = \gamma_{x_i} - \gamma_{x_j} + \Lambda^T\mathbf{d}.$$

From (9), we know that $\gamma_{x_i} - \gamma_{x_j} + \Lambda^T\mathbf{d} > 0$; hence, $S(x_i(I)) = \Lambda^T I + \gamma_{x_i}$ is a valid schedule.

Thus, an RIA whose computability tree is of unit depth always can be scheduled by solving (9) and setting $S(_{x_i}(I)) = \Lambda^T I + \gamma_{x_i}$. According to Theorem 2, the iso-temporal surfaces (defined as the loci of all index variables that have the same schedule) form hyperplanes defined by the normal vector Λ; hence, such a scheduling strategy is often referred to as the *hyperplanar* scheduling strategy. Also, because all the variables, when $l = 1$, can be scheduled by affine functions of the form Λ^TI+ γ_{x_i} (note that, all variables have the same scheduling vector Λ), such RIAs are said to admit *uniform affine* schedules. Next, we shall present a generalization of the hyperplanar scheduling procedure that can be used to schedule RIAs whose computability tree is of depth greater than unity.

3.2.2. Computability Trees of Depth >1 (Nonsystolic RIAs) Every node of the computability tree has a subgraph of the RDG associated with it. If the computability tree is of depth >1, then we showed that at every node i (with C^i and D^i as the connection and displacement matrices of the subgraph at the i^{th} node) in the tree one can find a feasible solution to the following equation:

$$\Gamma_i^T C^i + \Lambda_i^T D^i \geq H_i^T$$

where $H_i^T = [h_1 h_2 \ldots h_E]$ (see (6); h_i is 0 if the i^{th} edge appears in one of its children and $h_i = 1$ if the i^{th} edge never apears in any of its children). Thus, to every node i in the tree we can associate a set of three vectors $<\Gamma_i, \Lambda_i, H_i>$, and we shall define this triple as the set of *scheduling parameters* for the i^{th} node.

In the computability procedure, the subgraphs at the children nodes of the same parent node are disjoint; hence a node in the graph of the parent node either does not appear in any of its children nodes or it appears in exactly one of its child. This implies that if we trace a particular variable, say x_i, starting at the root node (which contains the RDG), then we will define a unique path in the tree that starts at the root node and ends in a node at some depth d. Such a path is defined as *trace of the variable x_i in the computability tree.* Hence, with every variable x_i we can associate a trace and a set of scheduling parameters $\{<\Gamma_j, \Lambda_j, H_j>\}$, which is the set of scheduling parameters of the nodes of the computability tree appearing in the trace. One can now generalize Theorem 2 and obtain a result that allows one to define iso-temporal subspaces (possibly of lower dimensions than $S-1$) in the index space.

THEOREM 3. Let $\{<\Gamma_j, \Lambda_j, H_j>\}$, $j = 1 \ldots d$ be the set of scheduling parameters associated with the trace of a variable x_i in the computability tree of a given RIA and let

$$Y_i = \begin{bmatrix} \Lambda_1^T \\ \Lambda_2^T \\ \vdots \\ \Lambda_d^T \end{bmatrix}.$$

If $(I - J) \in \mathrm{N}(Y_i)$ (where $\mathrm{N}(A)$ denotes the right null space of the matrix A), *then there can be no directed path from $x_i(I)$ to $x_i(J)$ and vice-versa.*

Proof. The proof is a generalization of the proof for Theorem 2 and can be found in [17] and [18]. □

Thus according to the above theorem, if $x_i(J) \leftarrow x_i(I)$ then there always exists a k such that $\Lambda_k^T(J - I) > 0$. In other words, if $\Lambda_k^T(I - J) = 0$, $\forall k = 1, \ldots, d$, then $x_i(I)$ and $x_i(J)$ can be assigned the same schedule and hence can be executed in parallel.

Lemma 5. For every variable x_i, the scheduling matrix Y_i (as defined in theorem 3) is of full row rank, i.e., rank(Y_i)=d.

Proof. We shall refer to [17] and [18]. □

For every variable x_i, one can compute its trace and the related scheduling matrix Y_i as defined in theorem 3. Then, two computations $x_i(I)$ and $x_i(J)$ can be assigned the same schedule if $(J - I) \in \mathrm{N}(Y_i)$. If the trace of a variable is of depth d, then it is shown in lemma 5 that *rank*(Y_i) = d. Hence, the above scheduling scheme defines iso-temporal subspaces of dimension $S - d$ in the index space. In general, if the depth of the computability tree is l, then there is a variable x_k whose iso-temporal subspaces are of dimension $S - l$. Hence, the extent of parallelism in the RIA is determined by the depth of the tree. For example, if the depth $l = S$, then the RIA has almost no parallelism and is basically sequential in nature. This is true for the RIA in example 1; it has a two-dimensional index space and the depth of the computability tree is 2.

Example 5. *(Subspace Scheduling of 2-D Filters)* It has been shown (see [3], [4]) that certain numerically stable 2-D filtering algorithms can be written in the form:

For all (i, j, k), **where** $0 \leq i < n$ and $0 \leq j, k \leq N$, **do**

$$x(i, j + 1, k + 1) = f_{x,i}(x(i, j, k), y(i, j, k), w(i, j, k))$$

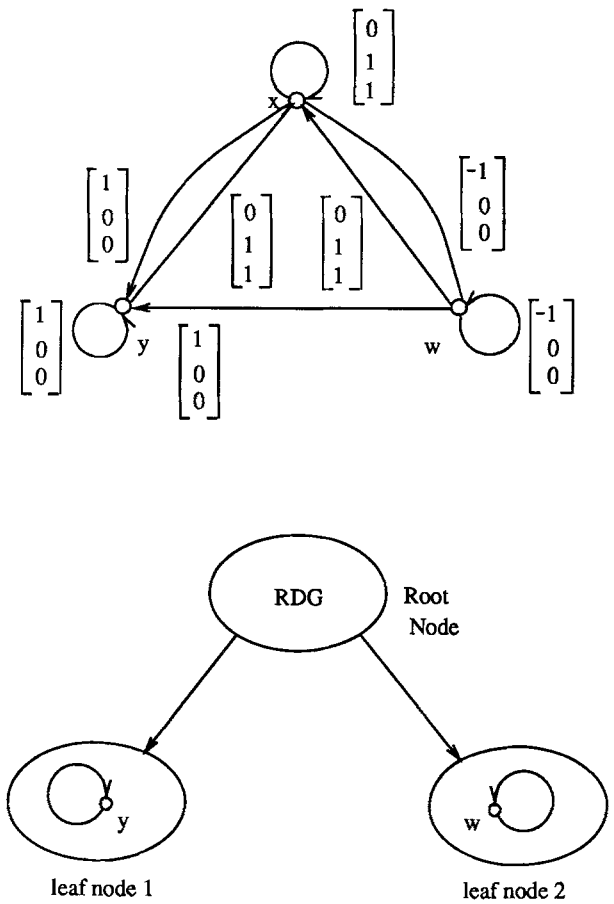

Fig. 4. The RDG for a class of numerically stable 2-D filtering algorithms and its computability tree.

$$y(i+1, j, k) = f_{y,i}(x(i, j, k), y(i, j, k), w(i, j, k))$$
$$w(i-1, j, k) = f_{w,i}(x(i, j, k), w(i, j, k))$$

where $f_{x,i}$, $f_{y,i}$, $f_{w,i}$ are linear functions that are determined by a synthesis procedure (see figure 4 for the RDG of the above RIA). The computability tree of the RIA (also shown in figure 4) is of depth 2; the arcs 7 and 8 (two disjoint self loops) are the only ones that appear in the second level. Thus, the computability analysis shows that the RIA is computable, and we shall determine the subspace scheduling for this problem. First, however, let us determine the scheduling parameters of the nodes of the computability tree.

For the root node let $H^T = [1\ 1\ 1\ 1\ 1\ 1\ 0\ 0]$. It has 0 entries for the two edges that appear in its children. Now one can solve for

$$\Gamma^T C + \Lambda^T D \geq H^T$$

and one possible solution is $\Lambda^T = [0\ 1\ 2]$ and $\Gamma^T = [0\ 2\ 1]$. Each of the other nodes in the computability tree is just a self loop with displacement vectors $[1\ 0\ 0]^T$ and $[-1\ 0\ 0]^T$. One can easily verify that the corresponding scheduling parameters are given by $\Gamma_1 = \Gamma_2 = 0$ and $\Lambda_1^T = -\Lambda_2^T = [1\ 0\ 0]$. Thus for the variable y the scheduling matrix is given by

$$Y_y = \begin{bmatrix} \Lambda \\ \Lambda_1 \end{bmatrix} = \begin{bmatrix} 0\ 1\ 2 \\ 1\ 0\ 0 \end{bmatrix}$$

and its null-space is spanned by the vector $[0{-}2\ 1]$. Hence, $y(I)$ and $y(J)$ will be scheduled at the same time if $I - J = \alpha[0\ -2\ 1]^T$. For the variable w, $N(Y_w) = N(Y_y)$; hence, the iso-temporal subspaces of the variables y and w are parallel lines.

The variable x appears only in the root node; hence, the iso-temporal subspaces are planes defined by the normal vector $[0\ 1\ 2]^T$ (see figure 5 for iso-temporal subspaces of the different variables). □

In the above example, the iso-temporal subspaces for the variable x are planes, whereas the iso-temporal subspaces for variables y and w are lines. In general, such a situation would occur whenever the depth of the trace of a variable is less than the depth of the computability tree. Often (as we shall discuss in Section 4), one requires all the iso-temporal substances to be of the same dimension. One procedure for doing so can be outlined as follows. If the trace of a variable x_i is of depth $d < l$ (where l is the depth of the computability tree), then one can append $l - d$ new rows to the scheduling matrix Y_{xi}

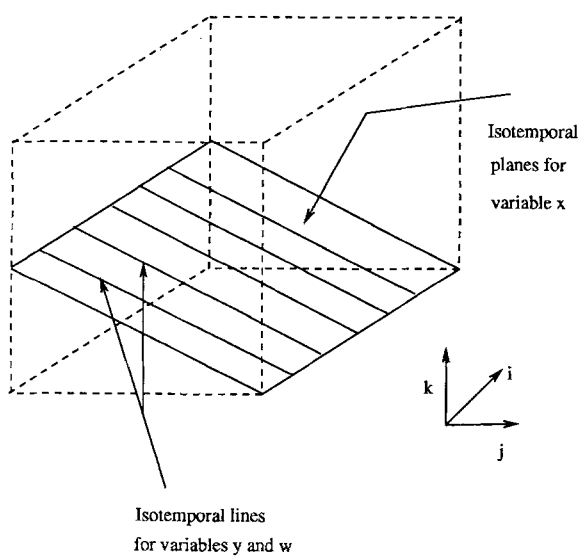

Fig. 5. Iso-temporal subspaces corresponding to different variables of the RIA in Example 5.

and form a new scheduling matrix Y'_{x_i}, such that *rank* $(Y'_{x_i}) = l$ and $N(Y'_{x_i}) \subset N(Y_{x_i})$. The last condition (i.e., $N(Y'_{x_i}) \subset N(Y_{x_i})$) ensures that the precedence constraints are not violated. That is, the new iso-temporal subspaces are subsets of the ones previously determine.

Example 6. In Example 5, iso-temporal subspaces of the variable y are subsets of those of the variable x. Hence, the iso-temporal subspaces of x can be reduced in dimension by augmenting its scheduling matrix with the vector Λ_1^T, i.e., the new scheduling matrix for x is given by

$$Y'_x = \begin{bmatrix} \Lambda \\ \Lambda_1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 0 & 0 \end{bmatrix}$$

It can be shown that the rows appended to obtain the new scheduling matrix can always be chosen to be the scheduling vectors of other variables in the RDG of the given RIA. □

Though we have been able to give a geometrical description of our scheduling procedure, we have not yet determined a formula for the sequencing index of $x_i(I)$ (represented as $S(x_i(I))$, as a function of i and the index point I. In the next section we shall discuss how to come up with such formulas and also show how to determine processor arrays that are compatible with the sub-space scheduling presented in this section.

4. Explicit Schedules and Parallel Implementations

In this section we shall first determine explicit formulas for scheduling any indexed variable $x_i(I)$ as a function of I, i and then we shall outline procedures to determine compatible processor arrays.

4.1. Explicit Schedules for RIAs

THEOREM 4. Let $\{<\Gamma_j, \Lambda_j, H_j>\}$ be the set of scheduling parameters associated with the trace of a variable x_i in the computability tree of a given RIA and let d be its depth. Then there exists a valid schedule S such that

$$S(x_i(I)) = \sum_{j=1}^{d} \alpha_j(\Lambda^T I + \gamma_{jx_i}) \tag{11}$$

Proof. We shall provide a constructive proof. The basic idea is to ensure that for every dependence of the form $x_i(I) \leftarrow x_j(I-\mathbf{d})$, one must have

$$S(x_i(I)) > S(x_j(I - \mathbf{d})) \ \forall\ I \in \mathbf{I}. \tag{12}$$

This implies that given the regularity of an RIA one needs to check (12) for every edge in the RDG of the RIA.

The construction procedure starts with leaf nodes of the computability tree and works its way up. If a node is a leaf node, then all the discussions in Section 3.2.1 are valid, i.e., if $<\Lambda_d, \Gamma_d, H_d>$ are the scheduling parameters of the leaf node, then a scheduling function that satisfies the precedence constraints imposed by the edges of the subgraph at the leaf node is given by $S_d(x_i(I)) = \Lambda_d^T I + \gamma_{dx_i}$.

Let us now consider a node whose children are leaf nodes of the tree. Let the graph at this node be represented by G and the scheduling parameters of the node be $<\Lambda, \Gamma, H>$. Recall that if C and D are the connection and displacement matrices of the graph G then we have a feasible solution to the inequality

$$\Gamma^T C + \Lambda^T D \geq H^T \tag{13}$$

where H^T has 1s corresponding to the edges that do not appear in the children nodes and 0s otherwise. Now, if a node $x_i(I)$ in G appears in one of its children, then define the new scheduling function to be

$$S(x_i(I)) = \alpha(\Lambda^T I + \gamma_{x_i}) + L_{x_i}(I)$$

where $L_{x_i}(I)$ is the schedule of $x_i(I)$ in the descendent node and α is yet to be determined; from the previous discussion, we know that $L_{x_i}(I)$ is of the form $\Lambda_d^T I + \gamma_{dx_i}$). If the node $x_i(I)$ does not appear in any of the children nodes, then define its scheduling function as $S(x_i(I)) = \alpha(\Lambda^T I + \gamma_{xi})$. Now we have to determine α such that the precedence constraint is met for every edge in G. G has two kinds of edges, and let us consider them separately as follows:

1. Let there be an edge in G of the form $x_i \rightarrow x_j$ with index displacement vector $\mathbf{d}$ that appears in one of the children nodes. Thus, the corresponding entry in H is 0, and by lemma 4, we know that the corresponding inequality in (13) is satisified with an equality, i.e., $\gamma_{x_j} - \gamma_{x_i} + \Lambda^T\mathbf{d} = 0$. Hence,

$$S(x_j(I)) - S(x_i(I - \mathbf{d})) = (\gamma_{x_j} - \gamma_{x_i} + \Lambda^T\mathbf{d}) + L_{x_j}(I) - L_{x_i}(I - \mathbf{d}).$$

Now, $\gamma_{x_j} - \gamma_{x_i} + \Lambda^T\mathbf{d} = 0$, and since the edge $(x_i \rightarrow x_j)$ appears in one of the children nodes that is already scheduled, we are assured that $L_{x_j}(I) - L_{x_i}(I - \mathbf{d}) > 0$. Thus, $S(x_j(I)) - S(x_i(I - \mathbf{d})) > 0$.

2. Let the edge $x_i \rightarrow x_j$ not appear in any of the children nodes. Then,

$$S(x_j(I)) - S(x_i(I - \mathbf{d})) = \alpha(\gamma_{x_j} - \gamma_{x_i} + \Lambda^T\mathbf{d}) + L_{x_j}(I) - L_{x_i}(I - \mathbf{d}).$$

Now, from (13) we know that $\gamma_{x_j} - \gamma_{x_i} + \Lambda^T\mathbf{d} = c > 0$; hence for the precedence constraint to be met one should have α such that

$$\alpha c > \text{Maximum } \forall I \in \mathbf{I} \text{ of } (L_{x_i}(I - \mathbf{d}) - L_{xj}(I)). \tag{14}$$

Such an α can be always found since the index space of the RIA under consideration is bounded in extent. Thus, each edge in G that does not appear in any of the children nodes (i.e., those edges whose corresponding entries in H are strictly positive) imposes a constraint on α as shown in (14). One can choose α which satisfies all these constraints by choosing the maximum.

The above procedure can be continued for any node all of whose children have been scheduled. Thus for a particular variable in the RDG, the schedule picks up a term of the form $\alpha_j(\Lambda_j^T + \gamma_{jx_i})$ at every node j in its trace. Hence, the complete scheduling function is given by

$$S(x_i(I)) = \sum_{j=1}^{d} \alpha_j(\Lambda^T I + \gamma_{jx_i}) \qquad \square$$

Example 7. *(Explicit Schedules for 2-D Filters)* Let us again consider the RDG shown in figure 4. We have already determined the scheduling parameters for this problem and they can be listed as follows:

node 1 : $<\Lambda = [0\ 1\ 2]^T;\ \gamma_x = 0,\ \gamma_y = 2,\ \gamma_w = 1$
node 2 : $<\Lambda_1 = [1\ 0\ 0]^T;\ \gamma_{2y} = 0>$
node 3 : $<\Lambda_2 = [-1\ 0\ 0]^T;\ \gamma_{3w} = 0>$

Let us schedule the leaf nodes first. The only variable in the leaf node 1 is y and hence its schedule is $\Lambda_1^T I + \gamma_2 y = i$. In the leaf node 2 the variable is w and its schedule turns out to be $-i$. However, we want schedules to be positive integers and this can be done by choosing $\gamma_{3_w} = n$, where n is the bound on the coordinate axis i. Thus the schedule of the variable w in node 3 is $n - i$. Now, we can schedule the root node; following the constructive procedure in the above theorem we get

$$\begin{aligned} S(x(I)) &= \alpha(\Lambda^T I + \gamma_x) \\ S(y(I)) &= \alpha(\Lambda^T I + \gamma_y) + (\Lambda_1^T i) \\ S(w(I)) &= \alpha(\Lambda^T I + \gamma_w) + (n + \Lambda_2^T i) \end{aligned}$$

A possible choice of α that satisfies the constraints of the type shown in (14) is $\alpha = n$. Hence, the valid schedules of the variables in the RIA are

$$\begin{aligned} S(x(I)) &= nj + 2nk \\ S(y(I)) &= i + nj + 2nk + 2n \\ S(w(I)) &= 2n - i + nj + 2nk \end{aligned}$$

One can now easily verify that the scheduling formula given in theorem 4 indeed conforms with the subspace scheduling scheme discussed in the previous section, i.e., if $(J - I) \in \mathrm{N}(Y_i)$ (as defined in theorem 3 in section 3) then $x_i(I)$ and $x_i(J)$ have the same schedule. Let $J = I + \Delta I$ where $\Delta I \in \mathrm{N}(Y_i)$. Then,

$$\begin{aligned} S(x_i(J)) &= S(x_i(I + \Delta I)) = \\ &\sum_{j=1}^{d} \alpha_j(\Lambda_j^T(I + \Delta I) + \gamma_{jx_i}) = \\ &\sum_{j=1}^{d} \alpha_j(\Lambda_j^T(I) + \gamma_{jx_i}) = S(x_i(I)). \end{aligned} \qquad \square$$

We can now make the following important observations about the scheduling formulas discussed in this section:

1. In [3], Rao, *et al.* showed that if the depth of the computability tree is l, then there is a path in the DG of the RIA of length $O(N^l)$, where N is the maximum among the size parameters of the RIA. We shall define a schedule to be an *asymptotically optimal schedule* if the maximum schedule of any variable in the RIA is $O(N^l)$. Let us consider a variable x_i that appears at depth l in one of the leaf nodes of the computability tree. Thus, in (11) $d = l$ and according to the constructive procedure outlined in Theorem 4, we observe that $\alpha_l = 1$, i.e., the edges of the leaf nodes can be scheduled by a simple affine function. α_{l-1} is determined next by an equation of the form 14; it involves determining the maximum of a linear function over a bounded index space, where the maximum bound is N. Hence α_{l-1} is $O(N)$. One can now show by induction that α_{l-i} is $O(N^i)$; thus α_1 is $O(N^{l-1})$. Now if one evaluates the maximum of $S(x_i(I))$, then it will be $O(N^l)$. One can also show that the same result is true for any variable whose trace is of depth $d < l$. Thus the latest schedule of any variable in the RIA is of $O(N^l)$; hence, the subspace scheduling strategy is asymptotically optimal.
2. We showed that the scheduling formula given in Theorem 4 corresponds to the subspace scheduling discussed in Section 3. Thus if the trace of a variable is of depth d, then the isotemporal subspaces of the variables are of dimension $S - d$. However, at the end of Section 3 we discussed possible ways of reducing the dimension of the isotemporal subspaces of a variable from $S - d$ to $S - l (d < l)$. A scheduling formula that corresponds to such a reduction can be easily obtained by slightly modifying the constructive procedure

outlined in Theorem 4. For example, suppose a variable x_i belongs to the graph at an interior node j (with scheduling parameters $<\Lambda_j, \Gamma_j, H_j>$) of the computability tree, but does not appear in any of its children nodes. Obviously then, $d < l$ and if one follows the procedure in Theorem 4 then one would get the iso-temporal subspaces to be of dimension $S - d$. An alternate strategy that reduces the dimension of the isotemporal subspaces can be outlined as follows. At node j of the computability tree, determine the variable with the maximum depth; let the schedule of the variable at one of the children nodes of the node j be $L(I)$. Then the schedule of the variable x_i at the node j can be defined as $\alpha_j(\Lambda^T + \gamma_{x_{ij}}) + L(I)$ instead of just $\alpha_j(\Lambda^T + \gamma_{x_{ij}})$. One can show that adding the extra term is equivalent to appending linearly independent rows to the scheduling matrix of x_i. The rest of the procedure for evaluating α_j can be carried out as in the proof of Theorem 4. A similar procedure can be carried out when the depths of the children nodes of any node in the computability tree are different.

Example 8. In the previous example, we determined schedules that correspond to isotemporal lines for variables y and w, but isotemporal planes for the variable x. One possible way of ensuring isotemporal lines for the variable x is to define $S(x(I)) = \alpha(\Lambda^T + \gamma_x) + \Lambda_1^T$. The extra term ensures that the new scheduling matrix for the variable x is given by

$$Y_x = \begin{bmatrix} \Lambda^T \\ \Lambda_1^T \end{bmatrix}$$

as discussed in the example at the end of Section 3. One can now carry out the same steps as in example 9; however, one can show that the choice α remains the same as before and hence the new scheduling function for the variable x is $S'(x(I)) = i + nj + 2nk$. □

4.2. An Example

We shall illustrate here how to determine valid schedules for an RIA that performs Gaussian elimination with partial pivoting. The RIA was derived in [22] and [23]; however, for scheduling purposes it is enough to present the RDG, which is shown in figure 6 along with the computability tree. The computability tree is of depth 2; hence, the isotemporal subspaces should be lines in the index space. The leaf nodes are to be scheduled first and a valid set of schedules can be given as

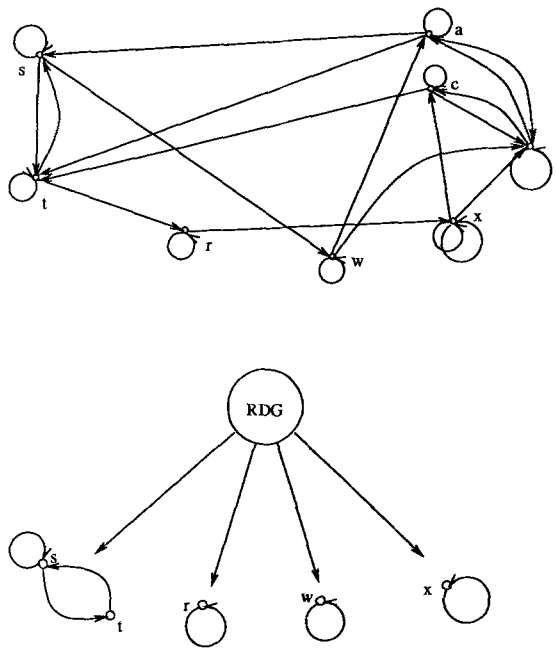

Fig. 6. The RDG and the computability tree of the RIA for performing Gaussian elimination with partial pivoting.

$$L_s(I) = i + 1,\ L_t(I) = i,\ L_x(I) = N - i,\ L_w(I) = N - i,\ L_r(I) = i$$

At the root node, we have to determine Λ and Γ that satisfies an equation of the form:

$$\Gamma^T C + \Lambda^T D \geq H^T$$

where H has 0 entries corresponding to the edges that appear in the subgraphs at the leaf nodes of the computability tree. One can solve the above inequality and a valid set of Λ and Γ can be given as:

$$\lambda_1 = 0,\ \lambda_2 = 1,\ \lambda_3 = 4.$$
$$\gamma_s = \gamma_t = 2,\ \gamma_a = \gamma_c = \gamma_r = 1,\ \gamma_x = 0,$$
$$\gamma_w = 3,\ \gamma_\rho = 4.$$

Thus, following the proof of theorem 4, the scheduling functions for a, c, ρ are of the form $\alpha(\Lambda^T + \gamma_i)$, whereas $S(s(I)) = \alpha(\Lambda^T I + \gamma_s) + i - 1$, $S(t(I)) = \alpha(\Lambda^T I + \gamma_t) + i$, $S(r(I)) = \alpha(\Lambda^T I + \gamma_r) + i$, $S(x(I)) = \alpha(\Lambda^T I + \gamma_x) + N - i$ and $S(w(I)) = \alpha(\Lambda^T I + \gamma_w) + N - i$. α can be determined as in (14) and a valid choice is given by $\alpha = N + 1$. The I/O latency of this explicit schedule is $O(N^2)$, and as discussed in [22], this is asymptotically optimal.

4.3. Parallel Implementations On Regular Processor Arrays

We shall use the linear projection scheme discussed by several authors [4], [3], [16] to determine regular processor arrays that are compatible with the subspace scheduling scheme. One of our results presented in this section shows that there always exist compatible processor arrays for any given subspace scheduling.

In the linear projection scheme, computations at index points I and J are mapped to the same processor if and only if $(I - J) \in \mathbf{R}(U)$, where U is an $S \times p$ integral matrix and $\mathbf{R}(A)$ denotes the column space of the matrix A. U is defined as the *iteration matrix*, and its column space $\mathbf{R}(U)$ is defined as the *iteration space.* Thus, computations at all index points that belong to the range space of the matrix U are computed at the same processor; hence, the processor array can be obtained by projecting the index space along $\mathbf{R}(U)$. An algebraic way of obtaining the processor array is to define a $(S - p) \times S$ matrix P^T which is orthogonal to U, i.e., $P^TU = 0$. Now, the processor array is obtained by assigning the computation at the index point I to the processor at the location P^TI. Thus the processor array is of dimension $S - p$.

Because of the regularity of the dependence graph of the RIA, the processor array generated by the above scheme also can be shown to be regular. Consider a dependence in the RIA of the form $x_i(I) \leftarrow x_j(I - \mathbf{d})$ $\forall\, I \in \mathbf{I}$. Now, by the assignment procedure, $x_j(I - \mathbf{d})$ is available at location $P^T(I - \mathbf{d})$ and the variable $x_i(I)$ is computed at location P^TI. Thus, one needs a communicating link from processor $P^T(I - \mathbf{d})$ to processor P^TI for all I; also, the displacement of the link is given by $P^TI - P^T(I - \mathbf{d}) = P^T\mathbf{d}$.

We showed in the previous section that if the computability tree is of depth l then the isotemporal subspaces of some of the variables are of dimension $S - l$; other variables may have isotemporal subspaces of higher dimensions. Hence, the dimensionality of the computations that can be done in parallel is $S - l$, and choosing a processor array of dimension more than that would be inefficient, since all processors would not be utilized. Therefore, for a computability tree of depth l, a good choice of the dimensions of U and P^T would be $S \times L$ and $(S - l) \times S$ respectively. The next important thing is to ensure that the assignment procedure is compatible with the scheduling procedure, and the following lemma provides a necessary and sufficient condition for compatibility.

Lemma 6. Let U be the iteration matrix used for obtaining a processor array and let the given RIA be scheduled according to the subspace scheduling scheme outlined in theorem 3. Then, U is a compatible iteration matrix if and only if

$$\mathbf{R}(U) \cap \mathrm{N}(Y_i) = \{0\}$$

for all $i = 1, 2, \ldots V$; where Y_i is the scheduling matrix of the variable x_i as defined in theorem 3

Proof. In order for the scheduling and assignment to be compatible, two computations scheduled at the same time should be assigned to different processors. Now, for a variable x_i we know that two variables $x_i(I)$ and $x_i(J)$ are to be scheduled at the same time if $(I - J) \in \mathrm{N}(Y_i)$ where Y_i is the matrix comprised of the scheduling parameters and is defined in theorem 3. Also, $x_i(I)$ and $x_i(J)$ are assigned on the same processor if $(I - J) \in \mathbf{R}(U)$. Hence, for the assignment and scheduling procedure to be compatible one should have no nonzero vector in the intersection of $\mathrm{N}(Y_i)$ and $\mathbf{R}(U)$; in other words $\mathbf{R}(U) \cap \mathrm{N}(Y_i) = \{0\}$. □

We noted that if the depth of the computability tree is l then for extraction of maximum parallelism, $\mathbf{R}(U)$ should be of dimension l; however, we showed in section 3 that the isotemporal subspace of some variable may be of dimension $S - d$, where $d - l$. In that case, one can show that one cannot find a U that would satisfy the conditions of the above lemma. Hence, in the rest of this section we shall assume that the isotemporal subspaces have been constrained to be of the same dimension for every variable by applying the techniques in section in 4.1. Let us now consider the example whose RDG is shown in Fig. 4 and show how to determine processor arrays for this example.

Example 9. *(Linear Arrays for 2-D Filters)* The height of the computability tree is 2 and the dimension S of the index space is 3. Thus for efficient implementation one should choose a linear array. Also, the isotemporal subspaces are given by the span of the vector $[0\ -2\ 1]^T$. Hence, for a compatible iteration matrix U one should have $[0\ -2\ 1]^T \notin \mathbf{R}(U)$. A particular choice of the iteration matrix is

$$U = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$$

The resultant processor array is shown in figure 7. A compatible processor array matrix is $P^T = [0\ 0\ 1]$, and

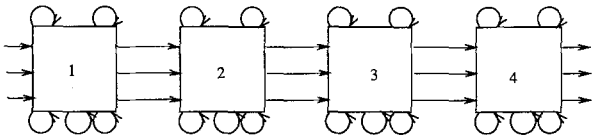

Fig. 7. A processor array obtained for the RIA in Example 7.

the processor interconnections can be obtained by considering every edge in the RDG and determining the link to which it is mapped. For example, $x(I) \leftarrow x(I - [0\ 1\ 1]^T)$; this translates into a link whose displacement vector is given by $P^T[0\ 1\ 1]^T = [1]$. Thus every processor j would have link connecting it to processor $j + 1$.

The delays along each of the links can also be determined from the scheduling function that we have determined earlier. Let us consider the same dependence again, i.e., $x(I) \leftarrow x(I - [0\ 1\ 1]^T$. The delay along the corresponding link would be $S(x(I)) - S(x(I - [0\ 1\ 1]^T))$, i.e., it is the time the variable $x(I - [0\ 1\ 1]^T)$ would have to wait before it is used by the variable $x(I)$. From the scheduling functions calculated previously, $S(x(I)) - S(x(I - [0\ 1\ 1]^T)) = 3n$. □

According to lemma 6, for the assignment and the scheduling to be compatible, the iteration matrix $\mathbf{R}(U)$ should have no non-zero intersection with V linear subspaces (each representing the isotemporal subspace associated with one of the indexed variables in the RIA). The following theorem shows that for any given RIA one can always find an iteration matrix, U, that would satisfy the conditions in lemma 6.

THEOREM 5. There always exists a matrix U such that

$$\mathbf{R}(U) \cap \mathrm{N}(y_i) = \{0\}$$

for all i=1, 2,. . .V, where Y_i is the scheduling matrix of the variable x_i as defined in theorem 3.

Proof. See Appendix. □

4.4. Subspace Scheduling of Systolic Algorithms

Systolic algorithms, defined over S-dimensional index space, have isotemporal subspaces that are hyperplanes of dimension $S - 1$, and the compatible choice of the dimension of iteration space is 1. This results in processor arrays of dimension $S - 1$. Often, however, the designer needs to map the given systolic algorithm to processor arrays of lower dimension than $S - 1$; obviously, hyperplanar scheduling can no longer be compatible, and one has to appropriately reduce the dimension of the isotemporal subspaces in the index space of the given RIA. Several researchers have proposed different methodologies for mapping systolic algorithms to lower dimensional arrays; see e.g., [5], [24]. In this section, however, we shall show how our subspace scheduling procedure can be used to do the mapping in a rather elegant manner.

Consider a systolic algorithm defined over a three-dimensional index space. A systolic implementation taking $O(N)$ time requires a two-dimensional array; however, let us assume that only a linear array is available. Let the isotemporal planes be defined by the scheduling vector Λ. From our discussion so far, we know that to design a linear array, one needs isotemporal subspaces that are lines instead of planes in the index space. This can be achieved by augmenting the scheduling matrix Y with a linearly independent row vector Λ_1^T. Thus the new scheduling matrix is

$$Y' = \begin{bmatrix} \Lambda^T \\ \Lambda_1^T \end{bmatrix}$$

and N(Y) defines the isotemporal lines in the index space. Note that, by construction, the new isotemporal lines lie on the isotemporal planes defined by the vector Λ.

In general, if the desired processor array is of dimension $S - p$ then one can obtain a compatible scheduling matrix $Y(p \times S)$ by augmenting the scheduling vector Λ with $p - 1$ linearly independent row vectors. Then N(Y) would define the new isotemporal subspaces. The following example should illustrate the basic ideas.

Example 10. *(Subspace Scheduling for Matrix Multiplication)* Consider the RIA for matrix multiplication given below [23]:

For all triples (i, j, k), $1 \le i, j, k \le n$ **do**
$a(i, j + 1, k) = a(i, j, k)$
$b(i + 1, j, k) = b(i, j, k)$
$c(i, j, k + 1) = c(i, j, k) + a(i, j, k)b(i, j, k)$.

One can verify that a valid scheduling vector for the RIA is $\Lambda^T = [1\ 1\ 1]$ and $\Gamma = \mathbf{0}$. Thus, for implementation on a two-dimensional array, a valid scheduling function is $S(a(I)) = S(b(I)) = S(c(I)) = i + j + k$. However, if one wants to implement this on a linear array, then one has to reduce the isotemporal planes to isotemporal lines. This can be done by augmenting the scheduling matrix with a vector $\Lambda_1^T = [0\ 0\ -1]$; thus the new scheduling matrix

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

The N(Y) is spanned by the vector $[1\ -1\ 0]^T$, and these lines define the new isotemporal subspaces. Next, we can apply the constructive procedure in the proof of theorem 4 to obtain an explicit scheduling function. The candidate scheduling function is of the form $S(a(I)) = S(b(I)) = S(c(I)) = \alpha(\Lambda^T I)+(\gamma + \Lambda_1^T I) = \alpha(i + j + k) + (n - k)$, where n is the bound along the coordinate axis k of the index space. The constant a can now be determined by inequalities of the form (14), and a valid choice that satisfies the precedence constraints, is $\alpha = n$. Hence, the explicit scheduling function is $S(a(I)) = S(b(I)) = S(c(I)) = ni + nj + (n - 1)k + n$. A linear processor array compatible with the subspace scheduling can be determined by a procedure similar to that in example 9. □

5. Concluding Remarks

In this paper we have analyzed the combinatorial properties of RIAs defined over bounded index spaces. We have generalized several results from Karp, *et al.* [1] and Rao, *et al.* [2], [4] and obtained asymptotically optimal scheduling and implementation strategies for any RIA defined over bounded index spaces. Our results indicate that if the depth of the computability tree is less than the dimension of the index space, then the RIA will always have unbounded parallelism. In particular one can define isotemporal subspaces in the index space. Assignment procedures that are compatible with the scheduling schemes are also presented. Moreover, explicit formulas for the schedules are generated.

6. Acknowledgements

The authors would like to thank Dr. S.K. Rao for his initial comments and numerous discussions.

Appendix

THEOREM 5. There always exists a matrix U such that

$$\mathbf{R}(U) \cap \mathrm{N}(Y_i) = \{0\}$$

for all $i = 1, 2, \ldots V$; where Y_i is the scheduling matrix of the variable x_i as defined in theorem 3.

Proof. Let N_i be a matrix whose column space is the same as N(Y_i). Then we have to find a matrix U such that $\mathbf{R}(U) \cap \mathbf{R}(N_i) = \{0\}$ for all $i = 1 \ldots V$. If the depth of the computability tree is l then without loss of generality we can assume that the scheduling matrices Y_i have rank l and hence the rank of N_i is $S - l$. Thus, we want an l-dimensional subspace which has only zero intersection with a given set of V, $S - l$ dimensional subspaces.

The basic idea behind the proof is rather an obvious one. That is, if we are given V (i.e., any finite number) p-dimensional subspaces in an S-dimensional space ($p < S$), then there always exists at least one vector which is not in the range of any of the given subspaces. Suppose the previous statement is not true then it would imply that an S-dimensional subspace can be completely spanned by the set-theoretic union of a finite number of p-dimensional subspaces, which is a contradiction. As an example, consider $p = 2$ and $S = 3$; then, we are given a finite number of planes in a three-dimensional vector space. One can easily verify that there is an infinite number of vectors that do not lie on any of the given planes.

Thus, we are given V matrices $N_1 \ldots N_V$ of dimensions $S \times (S - l)$ and we want to construct a $S \times l$ dimensional matrix whose range space intersects with the range space of $N_i, i = 1 \ldots V$, at the origin. We can construct such a matrix iteratively. In the first step, determine a vector u_1 such that $u_1 \notin \mathbf{R}(N_i)\ \forall\ i = 1 \ldots V$. From the previous discussion we know that this can be always done. Next, let us define V augmented matrices as follows

$$N_1^1 = [N_1|u_1],\ N_2^1 = [\mathbf{N}_2|u_1] \ldots N_V^1 = [N_V|u_1].$$

We can repeat the step with the augmented matrices and determine a second vector u_2 such that $u_2 \notin \mathbf{R}(N_i^1)\ \forall\ i = 1 \ldots V$. Thus vectors u_1 and u_2 are linearly independent and do not belong to the range of $N_i\ \forall\ i = 1 \ldots V$. We can again augment N_i^1 with u_2 and determine a third vector u_3. This can continue until the augmented matrices are $S \times S$ and we can then define, $U = [u_1 \ldots U_l]$. □

References

1. R.M. Karp, R.E. Miller, and S. Winograd. The organization of computations for uniform recurrence equations. *JACM*, 14:563–590, 1967.
2. H.V. Jagadish, S.K. Rao, and T. Kailath. Multi-processor architectures for iterative algorithms. *Proceedings of the IEEE*, 75, No. 9:1304–1321, Sept. 1987.

3. S.K. Rao, *Regular Iterative Algorithms and their Implementation on Processor Arrays.* PhD thesis, Stanford University, Stanford, Calafornia, 1985.
4. S.K. Rao and T. Kailath. Regular iterative algorithms and their implementations on processor arrays. *Proc. IEEE,* 76, No. 2:259–282, March 1988.
5. S.Y. Kung. *VLSI Array Processors.* Prentice Hall Series, 1987.
6. S.K. Rao. *Systolic Arrays and their Extensions.* Prentice Hall Series, to appear, 1988.
7. V.P. Roychowdhury, S.K. Rao, L. Thiele, and T. Kailath. On the localization of algorithms for VLSI processor arrays. *1988 Workshop On VLSI Signal Processing,* pages 459–470, Nov. 1988.
8. V.P. Roychowdhury, L. Thiele, S.K. Rao, and T. Kailath. On the localization of algorithms for VLSI processor arrays. Submitted to *IEEE Trans. Computers,* October, 1988.
9. H.T. Kung. Let's design algorithms for VLSI systems. In *Proc. Caltech Conf. on VLSI,* pages 65–90, Jan. 1979.
10. H.T. Kung. Why systolic architectures? *IEEE Computer Magazine,* 25:37–46, Jan. 1980.
11. H.T. Kung and C,E, Leiserson, Systolic arrays for VLSI. In *Sparse Matrix Proceedings,* pages 245–282. Philadelphia Society of Industrial and Applied Mathematicians, 1978.
12. D.I. Moldovan. On the analysis and synthesis of VLSI algorithms. *IEEE Trans. Computers,* C-31:1121–1126, Nov. 1982.
13. D.I. Moldovan. On the design of algorithms for VLSI systolic arrays. *Proc. IEEE,* pages 113–120, Jan. 1983.
14. J.A.B. Fortes. *Algorithm transformations for parallel processing and VLSI architectures.* PhD thesis, University of Southern California, Los Angeles, Dec. 1983.
15. P. Quinton. The systematic design of systolic arrays. Technical report, INRIA Report, Paris, 1983.
16. P.R. Capello and K. Steiglitz, Unifying VLSI array design with linear transformations of space-time. *Advances in Computing Research,* 2:23-65, 1984.
17. V.P. Roychowdhury and T. Kailath. Study of parallelism in regular iterative algorithms. Submitted to SIAM Journal of Computing, Dec. 1988.
18. Vwani P. Roychowdhury. *Derivation, Extensions and Parallel Implementation of Regular Iterative Algorithms.* PhD thesis, Department of Electrical Engineering, Stanford University, Stanford, California, December 1988.
19. W.M. Waite. Path detection in multi-dimensional iterative arrays. *JACM,* 14, 1967.
20. C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice Hall. 1982.
21. M. Behzad, G. Chartrand, and L. Lesniak-Foster. *Graphs and Digraphs.* Prindle, Weber and Schmidt International Series, 1979.
22. V.P. Roychowdhury and T. Kailath. Regular processor arrays for matrix algorithms with pivoting. Submitted to *CACM,* Feb. 1988.
23. V.P. Roychowdhury and T. Kailath. Regular processor arrays for matrix algorithms with pivoting, *Int. Conference on Systolic Arrays,* pages 237–246, May 1988.
24. D.I. Moldovan and J.A.B. Fortes. Partitioning and mapping of algorithms into fixed size systolic arrays. *IEEE Trans. Computers,* No. 1:1-12, January 1986.

Vwani P. Roychowdhury was born in Asansol, India, on April 16, 1961. He received the B. Tech degree from the Indian Institute of Technology, Kanpur, India, the M.S. degree from University of Rochester, Rochester, NY, and the Ph.D. degree from Stanford University, Stanford, CA, in 1982, 1983, and 1988 respectively, all in electrical engineering.

He is currently associated with the Information Systems Laboratory at Stanford University as a Research Associate. His research interests include parallel algorithms and architectures, special purpose computing arrays and VLSI design, faul-tolerant design and the theory of neural networks.

Thomas Kailath was educated in Poona, India, and at the Massachusetts Institute of Technology (S.M., 1959; Sc.D., 1961). After a year at the Jet Propulsion Laboratories, Pasadena, California, he joined Stanford University as an Associate Professor of Electrical Engineering in 1963. He was promoted to Full Professor in 1968, served as Director of the Information Systems Laboratory from 1971 through 1980, as Associate Department Chairman from 1981 to 1987, and currently holds the Hitachi America Professorship in Engineering.

Dr. Kailath has worked in a number of areas including information and communication theory, signal processing, linear systems, linear algebra, operator theory and control theory; his recent research interests include array processing, fast algorithms for nonstationary signal processing, and the design of special purpose computing arrays. He is the author of *Linear Systems,* Prentice Hall, 1980, and *Lectures on Wiener and Kalman Filtering,* Springer-Verlag, 1981. He has held Guggenheim and Churchill fellowships, among others, and received awards from the IEEE Information Theory Group, the IEEE Signal Processing Society, and the Education Award of the American Control Council. He is a Fellow of the IEEE and of the Institute of Mathematical Statistics and is a member of the National Academy of Engineering.