Study of Parallelism In Regular Iterative Algorithms

V. P. Roychowdhury T. Kailath

Information Systems Laboratory
Stanford University

Abstract

The study of Regular Iterative Algorithms (RIAs), which was introduced in a seminal paper by Karp, Miller, and Winograd in 1967, forms the basis for systematic design and analysis of regular processor arrays, including the class of systolic arrays. The RIAs have also been studied under different contexts and different names (on the last count RIAs were reintroduced, as late as 1987, under the name of dynamic graphs). In spite of the interest such algorithms have received over the years, many important issues that were left unresolved in the original paper by Karp *et al.,* have remained unanswered. In this paper we answer many such questions, particularly those relating to parallel scheduling and implementation of RIAs. Based on the analysis of a simple graph that captures the dependence structure of a given RIA, we are able to 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 generalizes the so-called hyperplanar scheduling which was shown by Karp *et. al.* to work for only a subclass of RIAs. This geometric scheduling scheme is shown to be asymptotically optimal and is used to completely characterize the extent of parallelism in any RIA. Moreover, we develop procedures to determine explicit schedules (*i.e,* a closed form expression for the schedule of every computation in the algorithm) that correspond to the geometric schedules, and also show that every RIA can be automatically mapped onto regular processor arrays.

1 Introduction

In 1967 Karp, Miller and Winograd [4] introduced the study of Regular Iterative Algorithms (RIAs). They, however, referred to the RIAs as Uniform Recurrence Equations (UREs). In 1983, the term URE was redefined by Quinton [10] to identify a class of algorithms that is only a subset of UREs, as originally defined by Karp *et al.* [4] (see *e.g.* , [12], [13]). In order to avoid any confusion, Rao *et al.* [12] renamed UREs as RIAs. The RIAs (or UREs, as defined by Karp *et. al.*) were also reintroduced recently as dynamic graphs (see [1, 2, 5]) and the relationship of such more recent work will be pinted out later in this section.

RIAs have been shown to form an important class of algorithms and several algorithms from diverse fields (see [11], [7], [13]) such as signal and image processing, matrix linear algebra, combinatorial algorithms, numerical solutions for PDEs *etc.* have been reformulated as RIAs. Perhaps, the greatest appeal of RIAs (as far as parallel processing is concerned) originates from the fact that they are very closely related to algorithms that can be implemented on regular mesh-connected processor arrays (including *systolic arrays* as introduced in [6]), and that has lead to an extensive interest in such algorithms.

In spite of the extensive study and parallel implementation of RIAs, several unresolved issues related to the scheduling and parallel implementation of RIAs have remained unanswered. To be more specific about the open issues and the known results, let us briefly introduce RIAs. An RIA has a set of V variables, which are defined at every index point $I \in \mathbf{I}$, where $\mathbf{I}$ is a subset of the integral lattice points in space Z^S. At every index point I, the variable x_i (referred to as $x_i(I)$) is computed as follows:

$$x_i(I) = f_{i,I}(x_1(I - \mathbf{d_{1i}}),\ x_2(I - \mathbf{d_{2i}}), \cdots, x_V(I - \mathbf{d_{Vi}}))$$

where, the index displacement vectors $\mathbf{d_{ij}}$ are indepen-

dent of the index point I; hence, the data dependencies are regular and local. Furthermore, the index space, $\mathbf{I}$, can be bounded or semi-infinite in extent.

In [4], 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) [1]. 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 leading to the so-called hyperplanar schedules (see Section 3). *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.*

We might also note here that most early work on systolic arrays (such as [8]) concentrated solely on RIAs that admit hyperplanar schedules; it seems that the contribution of these researchers was to develop procedures for mapping RIAs that admit hyperplanar schedules to regular processor arrays.

More recently Rao *et al.* [3], [12] attempted to use the full generality of RIAs for analyzing algorithms and mapping them on regular arrays (more general than systolic arrays). For example they formally showed that algorithms implementable on systolic arrays (properly defined) form a precise subclass of RIAs (exactly the sub-class that admits hyperplanar schedules). They also gave instances of RIAs that do not admit hyperplanar schedules (*i.e*, the computability trees for such algorithms are of depth greater than 1); instances of such RIAs include Gaussian elimination with partial or full pivoting and certain 2-D digital filtering algorithms. They introduced the concept of the *regular iterative arrays* which retain most of the properties of systolic arrays (such as regularity, spatial locality), but allows implementation of a larger class of algorithms.

However, several important questions related to RIAs remained unanswered, *e.g.*, (1) How to schedule any RIA (as mentioned before, there are several algorithms that do not admit hyperplanar schedules; hence, this is an important question)? (2) How to determine explicit schedules for the computations (*i.e*, determine the schedule of a variable $x_i(I)$ as a function of i and I)? (3) How to systematically generate processor arrays for implementing any given RIA?

[1] This problem of determining the computability of an RIA was reintroduced in [2] as an open problem. In [5] the same algorithm given by Karp *et. al.* was reintroduced and improved. Finally, in [1] efficient polynomial-time and NC algorithms were presented. For scheduling purposes, however, we show in this paper that a formal analysis of the computability tree leads to optimal schedules for any RIA.

In this paper we present the following results:

1. We have developed a general scheduling procedure that can be used for complete characterization of available parallelism in any RIA. 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. In the process, we have formalized the derivation of the computability tree; this formalization, perhaps, plays an important role in getting the new results.

2. We have addressed the issue of determining explicit schedules (that is we want to determine an algebraic formula for the schedule of the variable $x_i(I)$ as a function of i and I). In the case of RIAs defined over *bounded index spaces*, we show how to always obtain such schedules using again the information in the computability tree. For the case of RIAs defined over semi-infinite index spaces, it turns out that the problem of determining explicit schedules is much more difficult. *However, we are able to define a precise sub-class for which the explicit schedules are polynomial in nature (i.e. , the schedule for $x_i(I)$ is a polynomial in the coordinates of I and the maximum degree of the polynomial is given by the depth of the computability tree).* It is also shown via examples, that the schedule of a variable $x_i(I)$ can be exponential functions of i, I.

3. We show how to use the subspace scheduling to design compatible processor arrays for *any given RIA*.

In this extended abstract, we concentrate only on the first result; i.e. , how to determine geometric schedules for any given RIA. In the process we briefly review the derivation of the computability tree: we use our formalization and introduce new results that are necessary for the general results. The details of the other results on RIAs are discussed in the concluding remarks.

2 Definitions and Terminologies

2.1 Regular Iterative Algorithms

Example 1: A simple RIA.

For all tuples (i, j) , $1 \leq i \leq N$, $1 \leq j \leq \infty$ **do**

$$x_1(i,\ j) = jx_1(i-1,\ j+1)x_2(i,\ j)$$

$$x_2(i,\ j)\ =\ ix_2(i+1,\ j-1)+x_1(i,\ j-1)$$

□

The 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 the above example) and an *index vector* ($I = [i\ j]^T$, in the example).

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 $x_2(I - \mathbf{d_{12}})$ then the *index displacement vector* $\mathbf{d_{12}}$, corresponding to this direct dependence, is the same regardless of the index point I.

Definition 1 *A Regular Iterative algorithm is defined by a triple* $\{\mathbf{I},\ \mathbf{X},\ \mathbf{F}\}$ *where*

1. $\mathbf{I}$ *is the Index Space which is the set of all integral points enclosed within a specified region in a S-dimensional Euclidean space.*
2. $\mathbf{X}$ *is a set of V variables that are defined at every point in the index space, where the variable x_j defined at the index-point I will be denoted as $x_j(I)$ and takes on an unique value in any particular instance of the algorithm, and*
3. $\mathbf{F}$ *is the set of functional relations among the variables, restricted to be such that if $x_i(I)$ is computed using $x_j(I - \mathbf{d_{ji}})$, then*
   a. $\mathbf{d_{ji}}$ *is a constant vector independent of I and the extent of the index space, and*
   b. *for every J contained in the index-space, $x_i(J)$ is directly dependent on $x_j(J - \mathbf{d_{ji}})$ (if $(J - \mathbf{d_{ji}})$ falls outside the index-space, then, $x_j(J - \mathbf{d_{ji}})$ is an external input to the algorithm).*

In general, the index space $\mathbf{I}$ will be semi-infinite along certain coordinates and bounded along others (*e.g.*, the index space of the RIA in example 1 is bounded in extent along the coordinate i and semi-infinite along j). In this paper, the bounds on the coordinates will be referred to as the *size parameters* of the RIA. Thus without loss of generality, the index vectors I and the index displacement vector $\mathbf{d_{ij}}$ can be partitioned as follows

$$I = \begin{bmatrix} I_1 \\ I_2 \end{bmatrix}; \qquad \mathbf{d_{ij}} = \begin{bmatrix} \mathbf{d_{ij}^1} \\ \mathbf{d_{ij}^2} \end{bmatrix}$$

where I_1 represents the components of the index point I along coordinates bounded in extent and I_2 represents the components along coordinates semi-infinite in extent. Similarly, $\mathbf{d_{ij}^1}$ ($\mathbf{d_{ij}^2}$) represents the displacement along I_1 (I_2).

2.2 Dependence Graphs (DGs) and Reduced Dependence Graphs (RDGs)

The dependence graph of an algorithm 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.

The regularity of the dependence graph of an RIA, however, suggests that (as first noted by Karp *et al.* [4] and by Waite [14]) it can be concisely expressed in terms of a simpler and smaller graph called the *Reduced Dependence Graph* (RDG) [2]

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 $x_i(I - \mathbf{d_{ij}})$ for some $\mathbf{d_{ij}}$; 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. The RDG for the RIA of example 1 is shown in Fig. 1(a).

The equivalence of the dependence graph and the RDG can be formally stated by the following lemma (the proof is quite obvious).

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 from the node x_i to the node x_j whose displacement vector is $I - J$. Similarly, 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, $I + \mathbf{d} \in \mathbf{I}$.*

The analysis and implementations of RIAs, as presented in the rest of this paper would be based on the analysis of their RDGs and it is convenient to develop an algebraic representation. An RDG is a directed weighted graph and thus can be equivalently represented by its *connection matrix* (or the *edge-vector incidence* matrix) and another matrix that represents the weights along the edges. Since, the weights are the index displacement vectors of the corresponding dependences in the RIA, the second matrix would be referred to as the *displacement matrix* of the given RDG.

The connection matrix, C, has E columns, one for each edge in the RDG, and V rows, one for every node

[2] In fact Karp *et al.* refers to the RDG as introduced here as the dependence graph. However, this may lead to confusion as the *real* dependence graph of an RIA is the one where all the variables are explicitly defined.

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. The displacement matrix D can be also partitioned as

$$D = \begin{bmatrix} D_1 \\ D_2 \end{bmatrix}$$

where D_1 (D_2) corresponds to index displacements along the coordinates with bounded (semi-infinite) extent.

Representations of paths in the RDG and their index displacements can also be obtained in terms of its connection and displacement matrices by observing 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

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

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 $m \equiv 0$; 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 $\mathbf{d} = D\mathbf{q}$.

2.3 Valid Schedules and Computability of an RIA

The idea behind scheduling an RIA is to assign a time index (which is a non-negative integer) to each variable $x_i(I)$ in the RIA such that if $x_i(I) \longrightarrow x_j(J)$ (to be read as '$x_j(J)$ depends on $x_i(I)$') then the time index for $x_j(J)$ is larger than that for $x_i(I)$. Thus, the schedule of a variable can be interpreted as the time at which it can be executed. Also, all the variables with the same schedule can be executed in parallel without violating any precedence constraints. If $\mathbf{X}$ is the set of V variables that are defined at every point in the index space $\mathbf{I}$, then $\mathbf{X} \times \mathbf{I}$ denotes all the variables in the RIA. Now a valid schedule S is a function of the form

$$S: \mathbf{X} \times \mathbf{I} \rightarrow Z^+$$

where Z^+ is the set of non-negative integers and such that $S(x_i(I)) < S(x_j(J))$ if $x_i(I) \longrightarrow x_j(J)$.

A given RIA is defined to be *computable* if it admits a valid schedule and for all $I < \infty \in \mathbf{I}$, the schedule $S(x_i(I)) < \infty$. Now, it is well established that a dependence graph has a valid schedule if and only if it is acyclic. The added condition that the schedule of every variable at a finite index point be finite is satisfied if and only if there is no directed path in the dependence graph that originates at ∞. Based on these observations the following lemma can be stated.

Lemma 2 *An RIA is non-computable if and only if any one of the following two statements is true:*

1. *There is a cycle in the dependence graph.*
2. *There is a directed path in the dependence graph coming from infinity.*

3 Computability Analysis and Asymptotically Optimal Scheduling of Regular Iterative Algorithms

In this section we shall first review the computability analysis of RIAs according to our framework; most of the results presented here about the properties of the computability tree are new. Next, based on the computability tree, we shall propose an asymptotically optimal scheduling strategy that is valid for RIAs defined over bounded and semi-infinite index spaces.

3.1 Computability Analysis

An equivalent characterization of non-computable RIAs in terms of the RDGs can be stated by the following theorem (see [4]):

Theorem 1 *An RIA is non-computable if and only if there is a directed cycle in the RDG whose index displacement vector has 0 entries corresponding to the indices that are bounded in extent and non-positive entries corresponding to the indices that are semi-infinite in extent.*

Proof: see [4]. □

Recalling our discussion about algebraic representation of paths in the RDG, we can observe that if an RIA is non-computable then there is a integral vector $\mathbf{q}$ such that

$$C\mathbf{q} = 0,\ D_1\mathbf{q} = 0,\ D_2\mathbf{q} \leq 0;\quad \mathbf{q} > 0 \qquad (1)$$

Notice that, the vector $\mathbf{q}$ has positive entries corresponding to the edges that appear in the directed cycle whose displacement vector satisfies the conditions for

non-computability stated in theorem 1. Correspondingly, an edge i would be said to participate in a solution to (1) only if there is an integral vector $\mathbf{q}$ which satisfies (1) and has a strictly positive i^{th} entry.

Before we proceed further, we shall introduce a definition and a simple result from graph theory, which will be useful for our analysis procedure.

Definition 2 *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 3 *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.*

Theorem 1 suggests that one needs to examine only directed cycles in the RDG for checking computability. Also, lemma 3 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 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 its strongly connected components. Important scheduling properties, however, are determined by the analysis of the strongly connected components [4], [11], [13]. 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 [13], and also in [12], [4].

It may seem that to check computability of a given RIA, one only needs to check whether (1) is satisfiable. However, if (1) has a feasible solution, even then 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 $C\mathbf{q_2} = 0$, which have index displacement vectors that satisfy $D_1\mathbf{q_1} + D_1\mathbf{q_2} = 0$ and $D_2\mathbf{q_1} + D_2\mathbf{q_2} \leq 0$. Now, 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 Fig. 1(a). Arcs 1 and 4 are loops whose displacement vectors add to 0. Thus 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 the 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 entry 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. The procedure can be described as construction of a tree structure with the RDG at its root node and can be described as follows:

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. Else, 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
   $$C^i\mathbf{q} = 0, \quad D_1^i\mathbf{q} = 0, \quad D_2^i\mathbf{q} \leq 0. \qquad (2)$$
   If none of the edges participate then GOTO step 3. Else, 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).
   c. If the sub-graph is a single strongly connected component then the RIA is non-computable.
   d. Else, mark each strongly connected component as a child of the parent node. Also, tag each one 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: Fig. 1(b) shows the computability tree that corresponds to the RDG shown in Fig. 1(a). 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 [4]. The basic idea is

to observe that if an RIA is non-computable 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.

1. 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 that 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.

2. 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

$$C\mathbf{q} = 0, \quad D_1\mathbf{q} = 0, \quad D_2\mathbf{q} \leq 0; \qquad \mathbf{q} \geq 0 \qquad (3)$$

where C and D are respectively, the connection and displacement matrices of the subgraph at the node. Notice that, since the above inequality is homogeneous, then from the theory of linear programming [9] one can show that if there is a rational solution to (3) then it would also have an integral solution. Hence, one can treat (3) as a simple linear program rather than an integer linear program.

Lemma 4 *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):*

Maximize $H^T\mathbf{q}$ such that

$$C\mathbf{q} = 0, \quad D_1\mathbf{q} = 0, \quad D_2\mathbf{q} \leq 0; \qquad \mathbf{q} \geq 0 \qquad (4)$$

where $H^T \geq [1\ 1\ \cdots\ 1]$. Moreover, the dual linear program given (the objective function of the dual is 0):

$$\Gamma^T C + \Lambda_1^T D_1 + \Lambda_2^T D_2 \geq H^T; \qquad \Lambda_2 \geq 0 \qquad (5)$$

is feasible.

Proof: Notice that 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 only solution to (3) is $\mathbf{q} = 0$. Since, the node under consideration is a leaf node we are assured that only feasible solution to (4) is $\mathbf{q} = 0$. Hence, the objective function of (4) is 0.

Now, by Farkas lemma [9], 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. □

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

3. 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 descendent nodes. The second type of edges correspond to 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 4 is possible be defining a vector $H_i^T \geq [h_1\ h_2 \cdots h_E]$ where h_i is defined as follows:
h_i = 1 if the i^{th} edge does not participate in non-zero solution to (2); else
$h_i = 0$.
Thus 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)

Maximize $H_i^T\mathbf{q}$ such that

$$C^i\mathbf{q} = 0, \quad D_1^i\mathbf{q} = 0, \quad D_2^i\mathbf{q} \leq 0; \qquad \mathbf{q} \geq 0 \qquad (6)$$

where H_i is defined as above. Also the dual of the above linear program given by

Minimize 0 such that

$$\Gamma^T C^i + \Lambda_1^T D_1^i + \Lambda_2^T D_2^i \geq H_i^T; \qquad \Lambda_2 \geq 0 \qquad (7)$$

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 (7) would be of the form

$$\gamma_{x_j} - \gamma_{x_i} + \Lambda_1^T\mathbf{d_1} + \Lambda_2^T\mathbf{d_2} \geq 0$$

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

Lemma 5 *All inequalities in (7) where the right hand side is 0 are binding i.e., the $\geq$ is satisfied with equality.*

Proof: The proof will appear in the full paper. □

This result again will be useful later when we consider scheduling strategies.

3.2 Optimal Linear Sub-space Scheduling of RIAs

We shall first deal with the case when the computability tree is just the root node (that is of unit depth) and then we shall generalize our results to the case when computability trees are of depth > 1.

3.2.1 Scheduling when The Computability Tree is of Unit Depth

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 non-zero positive solution to $C\mathbf{q} = 0$, $D_1\mathbf{q} = 0$ and $D_2\mathbf{q} \leq 0$. The root node (containing the RDG itself) becomes a leaf node and the results of lemma 4 apply to the RDG of the given RIA. Thus, by lemma 4 there always exist vectors $\Gamma = [\gamma_{x_1}\ \gamma_{x_2}\ \cdots\ \gamma_{x_V}]$ and $\Lambda^T = [\Lambda_1^T\ \Lambda_2^T]$ such that

$$\Gamma^T C + \Lambda^T D \geq [1\ 1\ \cdots 1]; \qquad \Lambda_2^T \geq 0. \tag{8}$$

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 lemma 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 (8) by the vector $\mathbf{q}$ we get, $\Gamma^T C\mathbf{q} + \Lambda^T DQ \geq [1\ 1 \cdots 1]\mathbf{q}$. Now, $C\mathbf{q} = 0$, $D\mathbf{q} = (J - I)$ and $[1\ 1\ \cdots 1]\mathbf{q} > 0$; 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 $\Lambda^T I + \delta_{x_i}$ (where δ_{x_i} is a constant to be determined) at the same time step. The δ_{x_i} 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

$$(\Lambda^T I + \delta_{x_i}) - (\Lambda^T J + \delta_{x_j}) > 0 \tag{9}$$

This can be assured by setting $\delta_{x_i} = \gamma_{x_i}$ where $\Gamma^T = [\gamma_{x_1}\ \gamma_{x_2}\ \cdots \gamma_{x_V}]$ is a vector satisfying (8). This can be easily shown as follows: if there is a dependence of the form $x_i(I) \longleftarrow x_j(I - \mathbf{d})$ (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 (8), 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 can be always scheduled by solving (8) and setting $S(x_i(I)) = \Lambda^T I + \gamma_{x_i}$. Since, by 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, such a scheduling strategy is often referred to as the *hyperplanar* scheduling strategy. Also, such RIAs that have computability trees of unit depth are said to admit *uniform affine* schedules. Next, we shall present a generalization of this result for RIAs whose computability tree is of depth greater than unity.

3.3 Scheduling RIAs when the Computability tree is of Depth > 1

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 $H_i^T \geq [1\ 1 \cdots 1]$ if i is a leaf node; and $H_i^T \geq [h_1\ h_2 \cdots h_E]$ for interior nodes where 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 appears 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 would 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 would be defined as *trace* of the variable x_i in the computability tree. Thus to 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 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* $\{< \Gamma_j, \Lambda_j, H_j >\}$ *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 $(I - J) \in \mathbf{N}(Y_i)$ *(where* $\mathbf{N}(A)$ *denotes the right null space of the matrix A), then there can be no directed path from* $x_i(I)$ *to* $x_i(I)$.

Proof: Recall that all the subgraphs in the trace of the variable x_i are strongly connected; also, if node j is an ancestor of node k then the graph at node k is a subgraph of the graph at node j.

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 $(J - I)$. This loop is obviously contained in the root node (since it contains the RDG). Now following the trace of x_i, we can identify a node, say k, such that the loop is contained in the graph at the k^{th} node but is not contained in any of its children. In other words, the loop which defines a directed path from $x_i(I)$ to $x_j(J)$ uses at least one edge that is present only in the k^{th} node and not present in any of its children.

Hence, $\exists\ \mathbf{q} > 0$ such that $C^k\mathbf{q} = 0$ and $D^k\mathbf{q} = (J - I)$ and $H_k^T\mathbf{q} > 0$ (because from the previous discussion and the definition of H_k, H_k and $\mathbf{q}$ have strictly positive entries at some corresponding locations). However, we know that $\Gamma_k^T C^k + \Lambda_k^T D^k \geq H_k^T$ and multiplying both sides of it by the vector $\mathbf{q}$ we get, $\Gamma_k^T C^k\mathbf{q} + \Lambda_k^T D^k\mathbf{q} \geq [1\ 1 \cdots 1]\mathbf{q}$. Now, $C\mathbf{q} = 0$, $D\mathbf{q} = (J - I)$ and $[1\ 1\ \cdots 1]\mathbf{q} > 0$; hence $\Lambda_k^T(J - I) > 0$.

Thus, if $x_i(J) \longleftarrow x_i(I)$ then there always exists a k such that $\Lambda_k^T(J - I) > 0$. □

For every variable x_i, one can compute its trace and the related scheduling matrix Y_i as defined in the above lemma. Then, two computations $x_i(I)$ and $x_i(J)$ can be assigned the same scheduling index if $(J - I) \in \mathbf{N}(Y_i)$. If the trace is of depth d, then it is shown in the Appendix that $rank(Y_i) = d$. Thus, 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 simple example that we have been considering so far; it has 2-dimensional index space and the depth of the computability tree is 2.

Example 5: The RIAs for certain classes of numerically stable 2-D filters all have RDGs of the same form given in Fig. 2(a). The computability tree of the RIA (as shown in Fig. 2(a)) is of depth 2; the arcs 7 and 8 (two disjoint self loops) are the only ones that appear in the second level. Also, let us assume that the index space is bounded along the first coordinate, and the bound is n. Thus, the computability analysis shows that the RIA is computable. We will determine the subspace scheduling for this problem. Let us first 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 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

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

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, the iso-temporal subspaces are the same as that of y. However, the variable x appears only in the root node; hence, the iso-temporal subspaces are the planes defined by the normal vector $[0\ 1\ 2]^T$. The iso-temporal subspaces are shown in Fig. 2(b). □

4 Concluding Remarks

Following are some of the results that we did not show in this extended abstract:

1. Explicit Schedules for RIAs with bounded index spaces: we can prove the following theorem (the proof can be found in the full paper):

   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*

   $$S(x_i(I)) = \sum_{j=1}^{d} \alpha_j(\Lambda^T I + \gamma_{jx_i}) \qquad (10)$$

2. Explicit Schedules for RIAs with semi-infinite index-spaces: The general problem turns out to

be very difficult. In fact we can construct examples of RIAs where the schedule of a variable $x_i(I)$ has to increase exponentially (or even super-exponentially) as a function of the co-ordinates of I. However, we restrict ourselves to RIAs that admit *strictly positive* scheduling vectors (*i.e.* , the scheduling vector Λ_i at every node of the computability tree is strictly positive) then we can show that the schedule is polynomially bounded. Following are some of the relevant theorems that we can prove.

Theorem 5 *A computability tree admits strictly positive scheduling vector Λ at every node of the computability tree if and only if it admits a strictly positive Λ at the root node (i.e.,) $\exists\ \Lambda = [\lambda_1 \cdots \lambda_S]$ such that $\lambda_j > 0\ \forall\ j = 1 \cdots S$ and*

$$\Gamma^T C + \Lambda^T D \geq H_1^T$$

where C and D are the incidence and displacement matrices of the RDG.

Theorem 6 *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 that admits strictly positive scheduling vector at every node. Then there exists a valid schedule S such that*

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

3. We can answer questions such as : Given any RIA, how many connected components are there in its dependence graph? The methodology involves computing the GCD of the integral weight matrix D of the given RIA.

Acknowledgements
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.

References

[1] Edith Cohen and Nimrod Megiddo. Strongly Polynomial-Time and NC Algorithms for Detecting Cycles in Dynamic Graphs. *Proceddings of the 21st ACM Annual Symposium on Theory of Computing*, pages 523–534, 1989.

[2] K. Iwano and K. Steiglitz. Testing for cycles in infinite graphs with periodic structure. *Proceddings of the 19th ACM Annual Symposium on Theory of Computing*, pages 46–53, 1987.

[3] H. V. Jagadish, S. K. Rao, and T. Kailath. Multiprocessor architectures for iterative algorithms. *Proceedings of the IEEE*, 75, No. 9:1304–1321, Sept. 1987.

[4] R. M. Karp, R. E. Miller, and S. Winograd. The Organization of Computations for Uniform Recurrence Equations. *Journal of the ACM*, 14:563–590, 1967.

[5] S. R. Kosaraju and G. F. Sullivan. Detecting Cycles in dynamic graphs in polynomial time. *Proceddings of the 20th ACM Annual Symposium on Theory of Computing*, pages 398–406, 1988.

[6] 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.

[7] S. Y. Kung. *VLSI Array Processors.* Prentice Hall Series, 1987.

[8] D. I. Moldovan. On the design of algorithms for VLSI systolic arrays. *Proceedings of the IEEE*, pages 113–120, Jan. 1983.

[9] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice Hall, 1982.

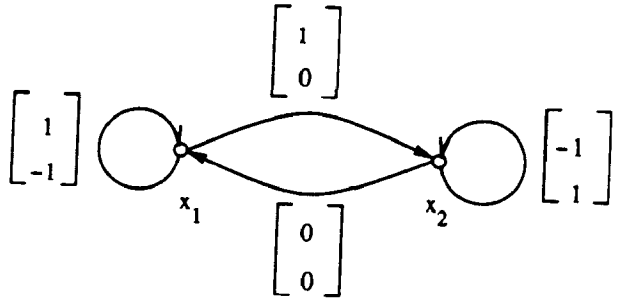
[10] P. Quinton. The systematic design of systolic arrays. Technical report, INRIA Report, Paris, 1983.

[11] S. K. Rao. *Regular Iterative Algorithms and their Implementation on Processor Arrays.* PhD thesis, Stanford University, Stanford, California, 1985.

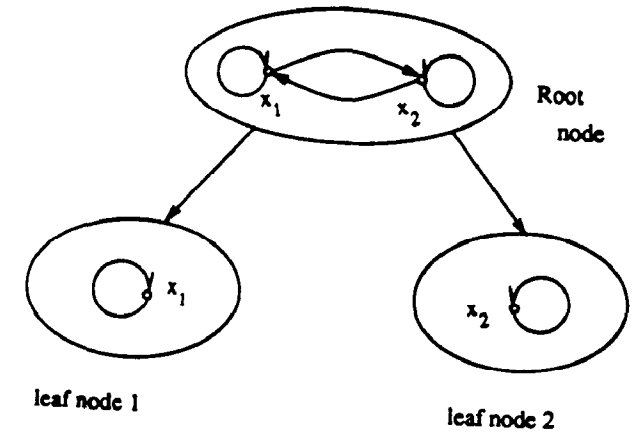
[12] S. K. Rao and T. Kailath. Regular iterative algorithms and their implementations on processor arrays. *Proceedings of the IEEE*, 76, No. 2:259–282, March 1988.

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

[14] W. M. Waite. Path detection in multi-dimensional iterative arrays. *Journal of the ACM*, 14, 1967.

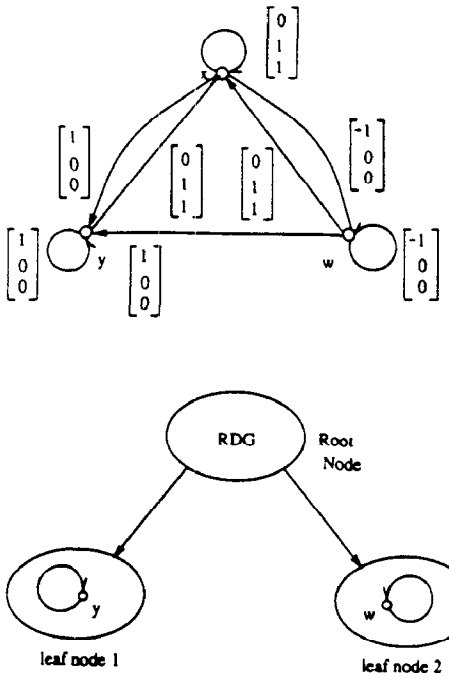


(a)

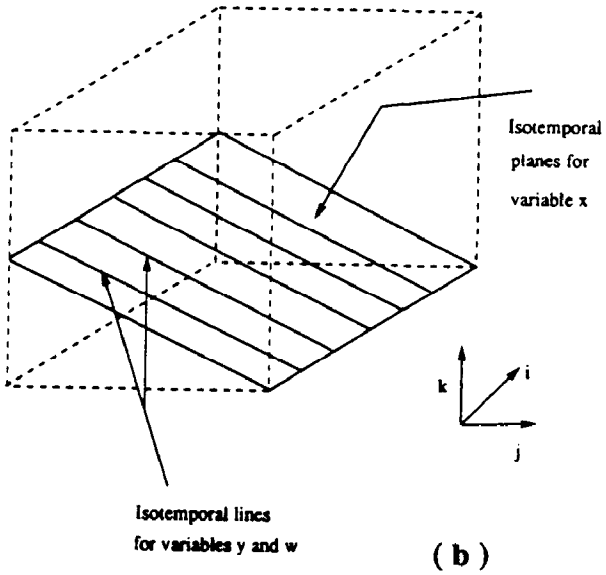


(b)

Figure 1: The Reduced Dependence Graph (RDG) of the RIA in Example 1 and its computability tree.



(a)



(b)

Figure 2: The RDG of an RIA that does not admit hyperplanar schedule, and its computability tree. (b) shows a subspace scheduling of the RIA.