Scheduling Linearly Indexed Assignment Codes *

T. Kailath V. P. Roychowdhury

Information Systems Laboratory
Stanford University

Abstract

It has been recently shown that linearly indexed Assignment Codes can be efficiently used for coding several problems especially in signal processing and matrix algebra. In fact, mathematical expressions for many algorithms are directly in the form of linearly indexed codes, and examples include the formulas for matrix multiplication, any m-dimensional convolution/correlation, matrix transposition, and solving matrix Lyapunov's equation. Systematic procedures for converting linearly indexed Assignment Codes to *localized algorithms* that are closely related to Regular Iterative Algorithms (RIAs) have also been developed. These localized algorithms can be often efficiently scheduled by modeling them as RIAs; however, it is not always efficient to do so. In this paper we shall analyze and develop systematic procedures for determining efficient schedules directly for the linearly indexed ACs and the localized algorithms. We shall also illustrate our procedures by determining schedules for examples such as matrix transposition and Gauss-Jordan elimination algorithm.

1 Introduction

It is well known by now that algorithms expressed as Regular Iterative Algorithms (RIAs) can be optimally scheduled and implemented on systolic arrays or on a generalized class of architectures that retain most of the desirable properties of systolic arrays and are termed as *regular iterative arrays* [7], [11], [13], [12], [6], [9], [18], [15], [8]. Most algorithms, however, are not available to the designer in the form of an RIA. The available representations are mostly in the form of sequential programs or mathematical expressions. In order to use the powerful and elegant theory of RIAs, several researchers have proposed methodologies to convert various intermediate representations of algorithms into RIAs or into forms that are closely related to RIAs.

A useful class of initial representations has been termed as *linearly indexed* Assignment Codes (ACs) [18], [16], [17]. Such codes have been shown to be very close to mathematical expressions for a number of problems, especially in signal processing and matrix algebra. A linearly indexed AC has statements of the form

$$x(PI + \mathbf{d}) \text{ depends on } y(QI + \mathbf{e}) \text{ for all } I \in \mathbf{I} \subset \mathbf{Z}^S \tag{1}$$

*This work was supported in part by the Department of the Navy, Office of Naval Research under Contract N00014-86-K-0726, the SDIO/IST, managed by the Army Research Office under Contract DAAL03-87-K-0033, and the U. S. Army Research Office under Contract DAAL03-86-K-0045.

where P and Q are integral matrices independent of I, $\mathbf{I}$ is an index space which is the set of all lattice points enclosed within a specified region in a S-dimensional Euclidean space and $\mathbf{d}$, $\mathbf{e}$ are constant displacement vectors. P and Q are often referred to as the *indexing matrices.* Many algorithms are actually directly available as (1), and examples include the formulas for matrix multiplication, any m-dimensional convolution/correlation, matrix transposition, and solving matrix Lyapunov's equation (see *e.g.* , [18]). Algorithms that are not directly in the form of (1) can often be easily put in that form by analyzing their sequential representations (see *e.g.* , [11], [18]).

Several researchers [4], [10], [2], [19], [1], [3] have tried to develop procedures for converting linearly indexed ACs to codes that are close to RIAs . However, it is only recently that a complete and a systematic procedure for transforming such codes has been proposed in [18], [16], [17]. The systematic procedure for conversion has been termed as the localization procedure and the derived algorithms have been termed as *localized algorithms.* The localized algorithms have *regular* dependence graphs that are not as homogeneous as the dependence graphs of RIAs. In particular, the localized algorithms may have dependences and variables that are defined only in specific parts of the dependence graphs, instead of being defined over the whole graph. Such algorithms can be modeled as those that have statements of the form

$$x_i(I) = f_i(x_1(I - d_{i1}), \cdots, x_V(I - d_{iV})) \quad \forall\, I \in \mathbf{I_i} \tag{2}$$

where the index space $\mathbf{I_i}$ may be different for each such statement.

The goal of this paper can now be motivated as follows. Procedures for exploiting the regularity of the dependence graphs of RIAs to systematically derive asymptotically optimal schedules are well known by now. However, the question is: Can one develop similar procedures for the localized algorithms that have statements of the form (2)? The answer does not turn out to be easy, and in fact a general scheduling theory for such algorithms has not been developed. In this paper we shall analyze and develop systematic procedures for scheduling certain classes of linearly indexed ACs and localized algorithms. In the process, we shall also generalize several results that are known for RIAs to the case of linearly indexed codes.

An easy but often effective way of scheduling localized algorithms is to ignore the inhomogeneties and treat the algorithm as an RIA. In fact, such a strategy is commonly adopted while scheduling algorithms such as the Gaussian elimination algorithm without pivoting. In this approach, however, one introduces additional dependences and hence the analysis and implementations may turn out to be suboptimal and inefficient. As an example, it can be shown [18], [14] that if the localized algorithm for Gauss-Jordan elimination algorithm is treated as a single RIA then the resulting algorithm admits no affine schedule and cannot be implemented on systolic arrays. We should mention here that one of the results in this paper shows how to determine affine schedules for the Gauss-Jordan algorithm by breaking up the localized algorithm into two RIAs.

In this paper, we shall first analyze how to determine affine schedules for the linearly indexed algorithms from which the localized algorithms are derived. If we are able to analyze and schedule the linearly indexed codes then of course the task of analyzing and scheduling the localized algorithms will become easier. For example, if the underlying linearly indexed code does not admit affine schedules, surely then one cannot find an affine schedule for the algorithms obtained after localization. In Section 2, we shall study the necessary and sufficient conditions for the existence of affine schedules for linearly indexed codes. We shall show that under certain assumptions, the existence of affine schedules can be determined by solving a set of linear inequalities, whose size is independent of the size of the index space. In the process, we are

able to find affine schedules for problems that could not be so scheduled by results reported earlier in [20], [21].

In Section 3, we shall deal with the issue of scheduling localized algorithms which do not admit affine schedules if one wants to schedule all variables using the same affine function. We show that such algorithms may still have affine schedules if the dependence graphs are properly partitioned. In other words, the whole algorithm may not admit a single affine schedule; however, when properly partitioned, each partition can have a separate schedule which is then consistent with the schedules of variables in other partitions. We will further apply this idea to examples such as matrix transposition and Gauss-Jordan elimination algorithm.

2 Affine Schedules for Linearly Indexed Codes

In this section we shall study the necessary and sufficient conditions for the existence of affine schedules of linearly indexed codes. In a linearly indexed code, the statements are of the form

$$x_i(P_iI + d_i) = f(x_1(Q_{i1}I + e_{i1}), \ldots, x_j(Q_{ij}I + e_{ij}), \ldots, x_V(Q_{iV}I + e_{iV})).$$

$\forall\ I \in \mathbf{I_i}$. If an affine schedule exists then one must have vectors $\Lambda = [\lambda_1 \cdots \lambda_S]$ and $\Gamma = [\gamma_1 \ \cdots \ \gamma_V]^T$ such that

$$\Lambda^T(P_iI + d_i) + \gamma_i - \Lambda^T(Q_{ij}I + e_{ij}) - \gamma_j \ > \ 0 \qquad \forall\, j = 1, \cdots, V$$

and $\forall\, I \in \mathbf{I_i}$. We can rewrite the above inequality, and for every j we will get a constraint of the form

$$\Lambda^T(P_i - Q_{ij})I + \Lambda^T(d_i - e_{ij}) + \gamma_i - \gamma_j \ > \ 0 \tag{3}$$

$\forall\, I \in \mathbf{I_i}$.

If $P_i = Q_{ij} = I_{S\times S}$ (*i.e.* , the identity matrix), then of course (3) reduces to a single linear inequality and solutions to such systems of linear inequalities have been studied in great detail in the context of RIAs [11], [18].

For arbitrary indexing matrices, $P_i\ Q_{ij}$, one can derive a system of linear inequalities by evaluating (3) for every $I \in \mathbf{I_i}$. This, however, will lead to a linear programming problem whose size will depend on the extent of the range space of $(P_i - Q_{ij})$ as determined by the index space $\mathbf{I_i}$. In order to avoid evaluating (3) for every I, the underlying range space should have a structure *i.e.* , one should be able to describe it without enumerating all possible index points. One such description that we shall study here is called a *cone.*

Definition 1 *Given a set of vectors $a_i \in R^n$, $i = 1, \cdots, m$; the cone generated by the set $\{a_i\}$, denoted by $C(a_i)$, is*

$$C(a_i) = \{x \in R^n : x = \sum_{i=1}^{m} \pi_i a_i, \ \pi_i \geq 0\}.$$

Our assumption in the rest of this section will be the following.

Assumption 1 For every $j = 1, \cdots, V$, the range space of $(P_i - Q_{ij})$, as determined by the index space $\mathbf{I - i}$ can be described by a cone $C(a_{ij})$.

Thus we shall assume that there are vectors a_{ij} such that if $I = (P_i - Q_{ij})J, \quad J \in \mathbf{I_i}$, then I can be expressed as

$$I = \sum_{i=1}^{m} \pi_i a_{ij} \qquad \pi_i \geq 0$$

. We shall see later that such an assumption is not too restrictive and in many examples, the range space of $(P_i - Q_{ij})$ can be described as a cone or its subset. Before we proceed further, another necessary definition will be introduced.

Definition 2 *Let $C(a_i)$ be the cone generated by the vectors a_i, $i = 1, \cdots m$, also let a_j be the vector with the maximum length, i.e. , $||a_j|| \geq ||a_i||$, for all $i = 1, \cdots, m$. Then, the neighborhood $N(a_i)$ of the cone $C(a_i)$ is defined as*

$$N(a_i) = \{x \in C(a_i) : x \text{ is integral, and } ||x|| \leq ||a_j||\}.$$

Note that, by definition, the vectors defining the cone (*i.e.* , a_i) belong to $N(a_i)$. Also, since the vectors in $N(a_i)$ are restricted to be integral, the set $N(a_i)$ has only finite number of vectors. Next, we present a lemma which will prove to be useful in proving a necessary and sufficient condition for the existence of affine schedules.

Lemma 1 *Let $C(a_i)$ be the cone generated by a set of vectors a_i, $i = 1 \cdots m$, and also let $N(a_i)$ be its neighborhood. If $\Lambda \in R^n$ such that $\Lambda^T a_i \geq 0$ for all $i = 1, \cdots, m$, then for every vector $v \in C(a_i)$ and $v \notin N(a_i)$, there exists p such that $\Lambda^T v \geq \Lambda^T a_p$.*

Proof: Since $v \in C(a_i)$, we have

$$v = \sum_{i=1}^{m} \pi_i a_i. \tag{4}$$

where, $\pi_i \geq$. Also, $v \notin N(a_i)$; hence, we have $||v|| \geq ||a_i||$, for all $i = 1, \cdots, m$. This implies that $\sum_{i=1}^{m} \pi_i \geq 1$. This is because

$$||v|| \leq \sum_{i=1}^{m} \pi_i ||a_i|| \leq (\sum_{i=1}^{m} \pi_i)||a_j||$$

where a_j is the vector with maximum length. Since, $v \notin N(a_i)$ we have $||v|| \geq ||a_j||$, which implies that $\sum \pi_i \geq 1$.

Now, let a_p be such that $0 \leq \Lambda^T a_p \leq \Lambda^T a_i$, for all $i = 1, \cdots, m$; then

$$\Lambda^T v = \sum_{i=1}^{m} \pi_i \Lambda^T a_i \geq (\sum_{i=1}^{m} \pi_i)\Lambda^T a_p.$$

Since $\sum \pi_i \geq 1$, we have $\Lambda^T v \geq \Lambda^T a_p$. □

Now, we can present the following theorem, which basically shows that if we make Assumption 1, then one can determine the existence of affine schedules for linearly indexed codes by dealing with a set of inequalities whose size is independent of the size of the index space $\mathbf{I_i}$.

Theorem 1 *Let the conditions in Assumption 1 be true. That is, for every dependence of the form*

$$x_i(P_i + d_i) = f(x_j(Q_{ij} + e_{ij})) \quad \forall I \in \mathbf{I_i}$$

we have a cone $C(a_{ij})$ that defines the range space of $(P_i - Q_{ij})$ as determined by the index space $\mathbf{I_i}$. Then, the given linearly indexed code will have an affine schedule if and only if there exist Λ and Γ such that

1. *Every* $x \in N(a_{ij})$ *satisfies*

$$\Lambda^T x + \Lambda^T(d_i - e_{ij}) + \gamma_i - \gamma_j > 0$$

2. *Every vector* $\{a_{ij}\}$, *defining the cone* $C(a_{ij})$ *satisfies* $\Lambda^T a_{ij} \geq 0$.

Proof: If the given linearly indexed code admits an affine schedule then (3) is satisfied, that is for every $x = (P_i - Q_{ij})I$ and $I \in \mathbf{I_i}$, we have

$$\Lambda^T x + \Lambda^T(d_i - e_{ij}) + \gamma_i - \gamma_j > 0. \tag{5}$$

By Assumption 1 and the definition of the neighborhood set $N(a_{ij})$ we know that every $x \in N(a_{ij})$ can be expressed as $x = (P_i - Q_{ij})I$ for some $I \in \mathbf{I_i}$; hence, every $x \in N(a_{ij})$ satisfies

$$\Lambda^T x + \Lambda^T(d_i - e_{ij}) + \gamma_i - \gamma_j > 0$$

Moreover, for every a_{ij} and $\alpha > 0$, $\alpha a_{ij} \in C(a_{ij})$. Hence, (5), is satisfied when $x = \alpha a_{ij}$. However, if $\Lambda^T a_{ij} < 0$, then one can always choose α large enough such that (5) is not satisfied. Thus, $\Lambda^T a_{ij} \geq 0$.

Next let us assume that conditions 1 and 2 in the statement of the theorem are satisfied. By Assumption 1, every x such that $x = (P_i - Q_{ij})I$ for some $I \in \mathbf{I_i}$, belongs to the cone $C(a_{ij})$ and can be expressed as $x = \sum \pi_k a_{ij}^k$. We have to show that (3) is satisfied. If $x = (P_i - Q_{ij})I \in N(a_{ij})$, then obviously by condition 1, (3) is satisfied.

Next, let us consider $x = (P_i - Q_{ij})I \in C(a_{ij})$ but $x \notin N(a_{ij})$. By Lemma 1 we know that there exists a k such that $\Lambda^T x \geq \Lambda^T a_{ij}^k$. Now, we know that $a_{ij}^k \in N(a_{ij})$, hence, $\Lambda^T a_{ij}^k + \Lambda^T(d_i - e_{ij}) + \gamma_i - \gamma_j > 0$. Since, $\Lambda^T x \geq \Lambda^T a_{ij}^k \geq 0$, we have $\Lambda^T x + \Lambda^T(d_i + e_{ij}) + \gamma_i - \gamma_j > 0$. Thus, (3) is satisfied and hence the linearly indexed code has an affine schedule. □

As stated before, the significance of the above theorem lies in reducing the size of the set of linear inequalities of the form (3) that one has to check in order to determine an affine schedule. To be more precise, for every dependence of the form

$$x_i(P_i I + d_i) = f_i(x_j(Q_{ij} I + e_{ij})) \tag{6}$$

one has to consider for every $x \in N(a_{ij}$ an inequality of the type

$$\Lambda^T x + \Lambda^T(d_i - e_{ij}) + (\gamma_i - \gamma_j) > 0$$

Where $N(a_{ij}$ is the neighborhood set of the cone $C(a_{ij})$ that describes the range space of $(P_i - Q_{ij})$. We showed that $N(a_{ij})$ has finite cardinality; hence, the number of linear inequalities generated by every dependence of the form (6) is fixed and is determined by the size of $N(a_{ij})$.

One can further simplify the above analysis and derive a sufficient condition for the existence of affine schedules as stated by the following theorem.

Theorem 2 *Let the conditions in Assumption 1 be true. That is, for every dependence of the form*

$$x_i(P_i + d_i) = f(x_j(Q_{ij} + e_{ij})) \quad \forall\, I \in \mathbf{I_i}$$

we have a cone $C(a_{ij})$ *that defines the range space of* $(P_i - Q_{ij})$ *as determined by the index space* $\mathbf{I_i}$. *Then, the given linearly indexed code will have an affine schedule if there exist* Λ *and* Γ *such that*

1.

$$\Lambda^T(d_i - e_{ij}) + \gamma_i - \gamma_j \; > \; 0$$

2. Every vector $\{a_{ij}\}$*, defining the cone* $C(a_{ij})$ *satisfies* $\Lambda^T a_{ij} \geq 0$.

Proof: We have to show that the following inequality is satisfied by all $I = (P_i - Q_{ij})J, \;\; J \in \mathbf{I_i}$,

$$\Lambda^T I + \Lambda^T(d_i - e_{ij}) + \gamma_i - \gamma_j \; > \; 0.$$

However, from Assumption 1 we know that

$$I = \sum \pi_i a_{ij}, \qquad \pi_i \geq 0.$$

Also by condition 2 in the statement of the theorem, we have $\Lambda^T a_i j \geq 0$; hence $\Lambda^T I = \sum \pi_i \Lambda^T a_{ij} \; \geq 0$. We are also given $\Lambda^T(d_i - e_{ij}) + \gamma_i - \gamma_j \; > \; 0$. Hence, we have $\Lambda^T I + \Lambda^T(d_i - e_{ij}) + \gamma_i - \gamma_j \; > \; 0$, for all $I = (P_i - Q_{ij})J, \;\; J \in \mathbf{I_i}$. In other words, Λ and Γ define an affine schedule for the given linearly indexed code. □

A sufficient condition for the existence of affine schedules has been presented in [20]; however, we are going to show that the conditions presented are stronger. In particular, they restrict the scheduling vector Λ to be such that for every affine dependence one should have

$$\Lambda^T(P_i - Q_{ij}) = 0. \tag{7}$$

Once Λ is so restricted, then the sufficient condition can be expressed solely in terms of the constant index displacement vectors d_i and e_{ij}. That is for every affine dependence one should have

$$\Lambda^T(d_i - e_{ij}) + \gamma_i - \gamma_j \; > \; 0.$$

As illustrated in one of the examples in this section, it is not always possible to satisfy (7); however, one may still be able to determine affine schedules by procedures suggested by Theorems 1 and 2.

The other major drawback of a strong condition of the form (7) can be explained in the context of localization. Recall that in Section 4.2 we observed that the global dependences in the dependence graph of linearly indexed code lie in the range space of the matrix $(P_i - Q_{ij})$. Thus an algorithm obtained after localizing the global dependences will have dependences that are in the range space of $(P_i - Q_{ij})$. If however, $\Lambda^T(P_i - Q_{ij}) = 0$, then the dependences introduced by the localization algorithm, cannot be separated by the scheduling vector Λ. Hence, the affine schedule for the linearly indexed code will not be a valid schedule for the localized algorithm.

The following two examples illustrate some of the concepts that have been stressed in this section.

Example 1: Consider the following linearly indexed assignment code

$$x(i,j) \; = f(y(0, -i-1)) \quad \forall \;\; 0 \leq i,j \leq n$$

Here,

$$P = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}; \quad Q = \begin{bmatrix} 0 & 0 \\ -1 & 0 \end{bmatrix}; \quad e = \begin{bmatrix} 0 \\ -1 \end{bmatrix}; \quad d = 0.$$

and

$$P - Q = P = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}.$$

Also the index space $\mathbf{I} = \{(i,j) | \ 0 \leq i,j \leq n\}$, hence the range space of $(P-Q)$ as determined by the index space $\mathbf{I}$ is a cone $C(a_i)$ defined by the vectors

$$a_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}; \qquad a_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

Only vector within the cone $C(a_i)$, which has smaller length is the 0 vector. Hence, the neighborhood set of the cone , is given as $N(a_i) = \{a_1,\ a_2,\ \mathbf{0}\}$. Thus, following Theorem 1, for an affine schedule to exist one should have

$$\Lambda^T x - \Lambda^T e + \gamma_x - \gamma_y \ > \ 0$$

where x takes the three values in $N(a_i)$. We also should have $\Lambda^T a_i \geq 0$. If these five inequalities are written in the matrix form then we obtain

$$[\gamma_x\ \gamma_y] \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ -1 & -1 & -1 & 0 & 0 \end{bmatrix} + [\lambda_1\ \lambda_2] \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 2 & 2 & 1 & 1 & 1 \end{bmatrix} \geq [1\ 1\ 1\ 1\ 1].$$

Note that the above set of inequalities is very similar to those we obtained while discussing affine schedules for RIAs. If the above linearly indexed statement is the only one in the algorithm, then a solution to the scheduling problem is given by,

$$\lambda_1 = 1; \ \ \lambda_2 = 1; \ \ \gamma_x = \gamma_y = 0.$$

□

Example 2: In this example we shall show a simple example where the sufficient conditions discussed in [20], [21] fail to yield an affine schedule; however, we shall show that our procedure can be easily used to determine an affine schedules. Let us consider the following two linerly indexed assignment codes:

$$x_1(i,j) = f_1(y_1(i-1, -i-1)) \quad \text{and} \quad x_2(i,j) = f_1(y_2(j-1, -j))$$

where $0 \leq i,j \leq n$. In the above statements the indexing matrices and there differences are

$$P_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad Q_1 = \begin{bmatrix} 1 & 0 \\ -1 & 0 \end{bmatrix}, \quad P_1 - Q_1 = \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix};$$

$$P_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad Q_2 = \begin{bmatrix} 0 & -1 \\ 0 & 1 \end{bmatrix}, \quad P_2 - Q_2 = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}.$$

Now, one can easily verify that the only vector Λ that satisfies (7) is $\Lambda = 0$ *i.e.* , $\Lambda^T(P_1 - Q_1) = 0$ and $\Lambda^T(P_2 - Q_2) = 0$ imply that $\Lambda = 0$. Hence, the above set of linearly indexed codes cannot be scheduled using the sufficient conditions stated in [20], [21].

One can however, carry out an analysis similar to that in example 1 and show that a valid scheduling vector can be given as:

$$\lambda_1 = 1,\ \lambda_2 = 1, \quad \Gamma = 0.$$

□

3 Further Analysis of Schedules

In [11], [18], it is shown that if uniform affine schedules do not exist for any given RIA, then one can determine schedules that are still affine but not uniform; however, the I/O latency of the algorithm can no longer be $O(N)$. In fact, the I/O latency will be in general $O(N^k)$ $(k \geq 2)$, where k is determined by the depth of the computability tree. One can also show by construction (see [11]) that such RIAs (which we have termed as non-systolic) have polynomially long paths in their dependence graphs.

In the case of linearly indexed codes, however, if one fails to find an affine schedule it does not rule out the possibility that the algorithm may still have an I/O latency of $O(N)$. This is so, because the dependence graph of the algorithm can be partitioned into several regions. Each such region can have affine schedules that are also consistent along the boundaries with the schedules of other regions. On the other hand, we can easily construct example where there are quadratic paths in the dependence graphs and decomposing the dependence graph into subgraphs will not help.

Thus, for scheduling linearly indexed codes, one not only has to pay attention to the dependences but also to the structure and extent of the dependence graph. This added complexity has made the development of an elegant theory for scheduling very hard. In fact, no results that can answer the general scheduling problem for linearly indexed codes is available right now. In this section we shall look at a few examples, and illustrate some of the points that we have made in the preceding discussion.

3.1 Matrix Transposition

The matrix transposition problem can be described as:

$$x(i,j) = y(j,i) \qquad \forall\, 1 \leq i,j \leq n$$

Here

$$P - Q = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \quad \text{and} \quad (P-Q)I = (i-j)\begin{bmatrix} 1 \\ -1 \end{bmatrix}.$$

If an affine schedule exist then it should satisfy $\Lambda^T(P-Q) + \gamma_x - \gamma_y > 0$. Clearly, one cannot determine a non-trivial affine schedule that works for all (i,j). Because for $i > j$, one should have

$$\Lambda^T \begin{bmatrix} 1 \\ -1 \end{bmatrix} > 0$$

and for $i < j$, one should have

$$\Lambda^T \begin{bmatrix} -1 \\ 1 \end{bmatrix} < 0.$$

The remedy is quite obvious in this case and involves breaking up the dependence graph into three regions.

1. $i = j$
   In this region there is no global data dependency and hence there is no need for an affine schedules.

2. $i > j$
   In this region, we want $\Lambda_1^T(P-Q)I > 0$. Which implies we want $[1\ -1]\Lambda_1 > 0$. And this is satisfied by the vector $\Lambda_1^T = [2\ 1]$.

3. $j > i$

   A little thought will convince one that this is the dual of the previous case. Hence a valid scheduling vector is $\Lambda_2^T = [1\ 2]$.

Thus in this example, we were able to determine affine schedule, for the given code in a piecewise fashion. This piecewise scheduling procedure turns out to be very useful and illustrated next on a more involved problem.

3.2 Gauss-Jordan Elimination Algorithm

In the Gaussian elimination algorithm (see *e.g.* , [5]) a given matrix A is transformed into an upper-triangular matrix U by repeatedly applying elementary row or column operations. In the Gauss-Jordan elimination algorithm, one attempts to transform the given matrix into a diagonal matrix D, using the same kind of row operations. In fact in the ordinary Gaussian elimination algorithm, at the k^{th} step, the k^{th} row is used to annihilate the a_{ik} entries for all rows $i > k$ *i.e.* , it annihilates all the entries of the updated matrix that are below the diagonal. In the Gauss-Jordan elimination algorithm on the other hand, one attempts to annihilate the a_{ik} entries for all rows $i \neq k$ *i.e.* , it tries to annihilate all elements both above and below the diagonal. Thus the end result in the latter algorithm is to obtain a diagonal matrix instead of an upper-triangular one.

Localized algorithms for Gaussian elimination algorithm without pivoting can be determined in a systematic fashion and for a detailed description the reader is referred to [18]. Because of space constraints here, we cannot go into the details of localizing the Gauss-Jordan elimination algorithm (see [14] for more details). However, it suffices to mention here that the similarity of operations with the Gaussian elimination algorithm, allows us to directly write down a linearly indexed single assignment code for the Gauss-Jordan elimination algorithm:

$$\textbf{For all} \quad \text{triples } (i,j,k)\,, \quad i \neq k, \quad k \leq j \leq N \text{ and } 1 \leq k \leq N-1 \textbf{ do}$$

$$a(i,\ j,\ k+1) \ :=\ a(i,\ j,\ k) - \frac{a(i,\ k,\ k)a(k,\ j,\ k)}{a(k,\ k,\ k)}$$

and $a(i,j,k+1) = a(i,j,k)$ if $i = k$. Note that, at each step one could divide the k^{th} row by the diagonal entry a_{kk} and thereby obtain an identity matrix as output instead of a general diagonal matrix. However, it only adds to the complexity of the notations and does not help in illustrating any new concept, and this additional step has been skipped here.

A trivial affine schedule can be obtained for the above linearly indexed code by choosing $\Lambda = [0\ 0\ 1]^T$. However, this is a schedule that satisfies the condition stated in (7), and has the usual drawbacks. In particular, the scheduling vector is orthogonal to the global dependences, and hence will not be a valid schedule for any localized version of the above SAC. For example, $a(i,j,k+1)$ depends on $a(k,j,k)$, hence the difference in the indexing matrices is

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

and obviously $[0\ 0\ 1](P-Q) = 0$. Hence, a more interesting problem is to determine affine schedules for the localized algorithm.

The localized algorithm for the region $k \geq i \geq N$ will be the same as the one for Gaussian elimination without pivoting (readers are again referred to [18], [14] for more details) and can be written as

$$l(i,\ j+1,\ k) = \begin{cases} l(i,\ j,\ k) \text{ if } j \geq (k+1) \\ \dfrac{a(i,\ j,\ k)}{u(i,\ j,\ k)} \text{ if } j = k \end{cases}$$

$$u(i+1,\ j,\ k) = \begin{cases} u(i,\ j,\ k) \text{ if } i \geq (k+1) \\ a(i,\ j,\ k) \text{ if } i = k \end{cases}$$

$$a(i,\ j,\ k+1) = a(i,\ j,\ k) - l(i,\ j,\ k)u(i,\ j,\ k)$$

In the region $1 \leq i \leq k-1$, however, the variable u needs to be propagated in a different direction and the localized algorithm for the region is can be given as

$$l_1(i,\ j+1,\ k) = \begin{cases} l_1(i,\ j,\ k) \text{ if } j \geq (k+1) \\ \dfrac{a(i,\ j,\ k)}{u(i,\ j,\ k)} \text{ if } j = k \end{cases}$$

$$u_1(i+1,\ j,\ k) = \begin{cases} u_1(i,\ j,\ k) \text{ if } i \geq (k+1) \\ a(i,\ j,\ k) \text{ if } i = k \end{cases}$$

$$a(i,\ j,\ k+1) = a(i,\ j,\ k) - l_1(i,\ j,\ k)u_1(i,\ j,\ k)$$

Thus the displacement matrix in the region 1, given by $k \leq i \leq N$, is

$$D_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

and that in the region 2 is

$$D_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Now if we want scheduling vectors Λ_1 and Λ_2 such that $\Lambda_1^T D_1 \geq [1\ 1\ 1]$ and $\Lambda_2^T D_2 \geq [1\ 1\ 1]$, then a possible set of solution is $\Lambda_1^T = [1\ 1\ 1]$ and $\Lambda_2^T = [1\ -1\ 1]$. Thus, the two regions have two different schedules and one can verify that these schedules are also consistent along the boundary.

4 Concluding Remarks

In this paper, we have investigated procedures to schedule linearly indexed codes by using affine functions. We have shown that under certain general assumptions, affine schedules can be determined by solving a set of linear inequalities. We have also studied cases where affine schedules do not exist. It was shown that failure to determine an affine schedule for the whole algorithm, need not imply that one cannot partition the given linearly indexed code and determine different affine schedules for different partitions. However, the problem of scheduling any given linearly indexed code or localized algorithms has not been solved completely. This problem is made complicated by the fact that the schedules are not only determined by the dependences (as is the case for RIAs) but also by the structure and the size of the complete dependence graph.

References

[1] Jichun Bu and Ed F. Deprettere. Converting sequential iterative algorithms to recurrent equations for automatic design of systolic arrays. *Submitted to IEEE Transaction on Computers*, Dec. 1987.

[2] J. M. Delosme and I. C. F. Ipsen. An Illustration of a Methodolgy for the Construction of efficient Systolic Architectures in VLSI. In *Proceedings Second Int. Symp. on VLSI Tech.* Taipei, Taiwan, 1985.

[3] V. Van Dongen and P. Quinton. Uniformization of Linear Recurrence Equations: A step towards the automatic synthesis of Systolic arrays. In *Proc. Int. Conf. on Systolic Arrays*, pages 473–482, May 1988.

[4] J. A. B. Fortes and D. I. Moldovan. Data broadcasting in linearly scheduled array processors. *Proc. 11th Ann. Symp. on Computer Architecture*, pages 224–231, June 1984.

[5] G. H. Golub and C. F. VanLoan. *Matrix Computations.* John Hopkins University Press, Baltimore, MD, 1983.

[6] 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.

[7] 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.

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

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

[10] Sanjay V. Rajopadhye. *Synthesis, Verification and Optimization of Systolic Arrays.* PhD thesis, University of Utah, Salt Lake City, December 1986.

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

[12] S. K. Rao. *Systolic Arrays and their Extensions.* Prentice Hall, to appear in 1989.

[13] S. K. Rao and T. Kailath. Regular Iterative Algorithms and their Implementations on Processor Arrays. *Proceedings of the IEEE*, 6, no.3:259–282, March 1988.

[14] V. P. Roychowdhury and T. Kailath. Scheduling linearly indexed assignment codes. Submitted to Journal of Parallel and Distributed Computing, Jan. 1989.

[15] V. P. Roychowdhury and T. Kailath. Subspace Scheduling and Parallel Implementation of Nonsystolic Regular Iterative Algorithms. to appear in the Journal of VLSI Signal Processing, April 1989.

[16] V. P. Roychowdhury, L. Thiele, S. K. Rao, and T. Kailath. On the Localization of Algorithms for VLSI Processor Arrays. *IEEE Workshop on VLSI Signal Processing*, Monterey, CA,:237–246, Nov. 1988.

[17] V. P. Roychowdhury, L. Thiele, S. K. Rao, and T. Kailath. On the Localization of Algorithms for VLSI Processor Arrays. Submitted to IEEE Trans. on Computers, September, 1988.

[18] Vwani P. Roychowdhuty. *Derivation, Extensions, and Parallel Implementations of Regular Iterative Algorithms.* PhD thesis, Stanford University, Stanford, California, Dec. 1988.

[19] Y. Wong and J. M. Delosme. Broadcast Removal in Systolic Algorithms. In *Proc. Int. Conf. on Systolic Arrays*, pages 403–412, May 1988.

[20] Y. Yaacoby and P. R. Capello. Scheduling a system of affine recurrence equations onto systolic arrays. In *Proc. of International Conf. on Systolic Arrays*, pages 373–382, May 1988.

[21] Y. Yaacoby and P. R. Capello. Scheduling a system of affine recurrence equations onto systolic arrays. Technical Report 19, Dept. of Computer Science, UCSB, Santa Barbara, Feb. 1988.