Synthesizing Nested Loop Algorithms Using Nonlinear Transformation Method

Jang-Ping Sheu, Member, IEEE, and Chih-Yung Chang

Abstract—In this paper, we synthesize nested For-loops with partitions on the innermost loop. First, we present a nonlinear transformation algorithm to exploit the parallelism of the For-loops. By the mapping of nonlinear transformations, iterations of For-loops can be executed in a parallel form. The proposed algorithm is useful in exploiting the parallelism of For-loops with one or more partitions on the innermost loop. Then, we also design algorithms to partition and map the nested For-loops onto the fixed size systolic arrays. Based on the time and space mapping schemes, all the iterations of For-loops can be correctly executed on the array processors in a parallel form.

Index Terms—Data dependence, hyperplane method, nested For-loops, parallel processing, systolic arrays.

I. INTRODUCTION

FOR-LOOPS are widely used in programs such as matrix multiplication, recurrence evaluations, Gaussian elimination, LU decomposition, shortest path finding, and a version of discrete Fourier transform which spend a large amount of computing time. Since For-loops are the main source of parallelism in programs, it is attractive for researchers to exploit their parallelism. Although For-loops offer a large amount of parallelism, it is sometimes difficult to identify the parallelism. In order to exploit the concurrency operations of For-loops, it becomes necessary to construct the data dependence graph which indicates the dependence relations between statements [2], [4], [6], [11], [14]. The analysis of data dependencies in high-level language programs for the purpose of detecting concurrency of operations has drawn considerable attention. Kuck [5] has exploited the parallelism of simple loops and introduced the notion of dependence relations between assignment statements. Banerjee [1] extended the methodology of transforming ordinary programs into highly parallel forms. With investigation based upon dependencies between statements, they have provided algorithms for exploiting parallelism in loops.

Several approaches on the analysis of parallel execution of For-loops are based on the iteration level [2], [9], [11]. Lamport [6] tried to execute the loop body concurrently for all iterations which do not have any dependence relations between them. Moldovan [11] mapped n-dimensional For-loops to a t-dimensional time hyperplane and s-dimensional space hyperplane, where \( n = s + t \). Lee [7], [8] proposed the necessary and sufficient conditions for mapping the For-loops onto special purpose VLSI systolic linear arrays and multidimensional arrays.

It is a simple and fast method to use a linear transformation function to identify the execution time of each index point for constructing the parallel executable For-loops. However, for many algorithms, there is more than one partition on the innermost loop. If algorithms have partitions on the innermost loop, the degree of parallelism may not be improved by using the linear function. Therefore, we propose the nonlinear transformation functions for solving such For-loops. First, we propose a nonlinear time transformation algorithm to parallelize the execution of For-loops. Then, we construct the structure of systolic arrays and map each iteration of For-loops onto a processing cell of VLSI systolic arrays by using the proposed space mapping algorithm. Besides, we discuss how to partition the For-loops when the number of processors is fixed. Algorithms and examples are also given to show how to map all the iterations onto the systolic arrays structure and perform in a parallel form.

The rest of this paper is organized as follows. In Section II, we introduce the linear transformation method and then state the For-loops model and basic concepts of the nonlinear transformation method. In Section III, we will propose the nonlinear transformation algorithm for parallelizing the execution of the For-loops model. In Section IV, we present an approach to space transformation and discuss how to partition the For-loops into bands that are suitably executed on fixed size systolic arrays. Finally, some conclusions are given in Section V.

II. BACKGROUND AND BASIC CONCEPT

During parallel execution of a program, data dependence, which defines a partial execution order on the statements of a program, must be observed in order to preserve the semantics of the program. A data dependence is defined between two statements (not necessary distinct) \( S_1 \) and \( S_2 \), if a scalar or array variable is generated by \( S_1 \) and then used in \( S_2 \). We denote this relation by symbol \( S_1 \rightarrow S_2 \). Let \( I \) denote the set of all positive integers and \( I^n \) denote the set of \( n \)-tuples of positive integers. The index set of a loop body is a subset of \( I^n \) and is defined as

\[
J = \{ (j_1, \cdots, j_n) | l_1 \leq j_1 \leq u_1, \cdots, l_n \leq j_n \leq u_n \}
\]

where \( n \) is the depth of the nested loop, and \( l_i \) and \( u_i \) are the lower bound and upper bound corresponding to the
index variable \( j_1 \) in a sequential loop program. Assume that the data dependence relation exists in two statements \( S_1 \) and \( S_2 \) and \( S_1(j_1) \rightarrow S_2(j_2) \), where \( j_1 \) and \( j_2 \in \mathbb{N} \). The data dependence vector is defined by \( d = j_2 - j_1 \).

The positive value of dependence vector \( d \) indicates that the execution of two statements \( S_1(j_1) \) and \( S_2(j_2) \) performs with \( d \) iterations difference.

The hyperplane method proposed by Moldovan [11] applies a linear time function \( \Pi \) to transform each \( d \) into \( \Pi d \). To ensure a correct execution order, \( \Pi \) must satisfy the condition \( \Pi d_i > 0 \) for each dependence vector \( d_i \). Assume that any single computation or set computations performed at one index point \( j \in \mathbb{N} \) takes one unit time; thus, a computation indexed by \( j \) in the original algorithm will be processed at time \( \Pi j \).

By using this transformation function \( \Pi \), the iteration indexes \( j_1 \) and \( j_2 \) in \( \mathbb{N} \) satisfied \( \Pi(j_1) = \Pi(j_2) \) can be executed concurrently. The hyperplane method is an easy way for mapping the original loops to a parallel form. However, if there are some partitions on the innermost loop, the degree of parallelism is inverse proportional to the number of partitions by using the hyperplane method. We consider the model of n-nested For-loops with \( p \) partitions on the innermost loop. Let \( P_i \) denote the \( i \)th partition in the innermost loop body and \( J_i^n \) denote the corresponding index set performing on \( P_i \). The loop form can be viewed as follows.

For \( j_1 = l_1 \) to \( u_1 \) by \( k_1 \)

\[
\vdots
\]

For \( j_{n-1} = l_{n-1} \) to \( u_{n-1} \) by \( k_{n-1} \)

For \( j_n = l_n \) to \( u_n \) by \( k_n \)

\[
\vdots
\]

End \( j_n \)

End \( j_{n-1} \)

End \( j_1 \) \hfill (L1)

The values of \( l_i \) and \( u_i \) are the lower bound and upper bound corresponding to the index variable \( j_i \), and the values of \( l_n \) and \( u_n \) denote the lower bound and upper bound of index variable \( j_n \) on partition \( P_i \). Assume that there is no overlap on the index set of different partitions in the nested loops algorithm. Without loss of generality, we assume that \( l_i = u_i + 1 \) for \( 1 \leq i \leq p - 1 \) and \( k_j = 1 \) for \( 1 \leq j \leq n \). We first consider the For-loops model with two partitions and the model with \( p \) partitions can be examined in the next section. Assume that \( D_1 \) and \( D_2 \) are the dependence matrices of two partitions \( P_1 \) and \( P_2 \), respectively. Let \( J_1 \) and \( J_2 \) denote the corresponding index sets performing on partitions \( P_1 \) and \( P_2 \), respectively. There are three cases of dependence relation between \( J_1 \) and \( J_2 \).

Case 1: Some iterations of \( J_2 \) data dependent on some iterations of \( J_1 \).

Case 2: Some iterations of \( J_1 \) data dependent on some iterations of \( J_2 \).

Case 3: Both of case 1 and case 2 hold.

In order to simplify the discussion, we define the following terms:

**Definition 2-1:** A dependence vector \( d_i \) is an interdependence vector, denoted as \( d_i \), if

\[
d_i \in D_1 \quad \text{and} \quad \exists j \in J_1 \quad \text{such that} \quad j - d_i \in J_2
\]

\[
d_i \in D_2 \quad \text{and} \quad \exists j \in J_2 \quad \text{such that} \quad j - d_i \in J_1
\]

Let \( D_1^* \) and \( D_2^* \) denote the matrices of interdependence vectors of partitions \( P_1 \) and \( P_2 \), respectively. The interdependence matrix \( D^* \) is defined as \( D^* = D_1^* \cup D_2^* \).

**Definition 2-2:** The interdependence index set \( J^* \) is defined as

\[
J^* = \{ j \mid j \in J_1 \quad \text{and} \quad j - d_i \in J_2 \}
\]

\[
\cup \{ j - d_i \mid j \in J_1 \quad \text{and} \quad j - d_i \in J_2 \}
\]

\[
\cup \{ j \mid j \in J_2 \quad \text{and} \quad j - d_i \in J_1 \}
\]

\[
\cup \{ j - d_i \mid j \in J_2 \quad \text{and} \quad j - d_i \in J_1 \}
\]

for all \( d_i \in D_1 \) and \( d_j \in D_2 \).

To illustrate the ideas of the nonlinear transformation method, we start with the following example.

**Example 2-1:**

For \( j_1 = 0 \) to 5

For \( j_2 = 0 \) to 2

\[ A(j_1, j_2) = A(j_1 - 1, j_2 + 1) + A(j_1 - 1, j_2) \]

End \( j_2 \)

For \( j_2 = 3 \) to 5

\[ A(j_1, j_2) = A(j_1, j_2 - 1) * A(j_1 - 2, j_2 + 1) \]

End \( j_2 \)

End \( j_1 \)

In this example, the two-dimensional index set \( J = \{ (j_1, j_2) \mid 0 \leq j_1, j_2 \leq 5 \} \) has two partitions on the innermost loop. We have

\[
J_1 = \{ (j_1, j_2) \mid 0 \leq j_1 \leq 5, 0 \leq j_2 \leq 2 \}
\]

\[
J_2 = \{ (j_1, j_2) \mid 0 \leq j_1 \leq 5, 3 \leq j_2 \leq 5 \}
\]

The index dependence graph can be shown in Fig. 1. We can easily find four dependence vectors in this example:

\[
d_1 = (1, -1)^t = d_1^*, \quad d_2 = (1, 0)^t,
\]

\[
d_3 = (0, 1)^t = d_3^*, \quad \text{and} \quad d_4 = (2, -1)^t.
\]

The dependence matrices \( D_1 \) and \( D_2 \) corresponding to partitions \( P_1 \) and \( P_2 \) are

\[
D_1 = \begin{bmatrix} 1 & 1 \\ -1 & 0 \end{bmatrix} = [d_1 \, d_2] \quad \text{and} \quad D_2 = \begin{bmatrix} 0 & 2 \\ 1 & -1 \end{bmatrix} = [d_3 \, d_4],
\]
respectively. Obviously, from the index dependence graph as shown in Fig. 1, the interdependence index set is \( J^e = \{(j_1, j_2) | 0 \leq j_1 \leq 5, 2 \leq j_2 \leq 3\} \). By Definition 2-1, we can find an index \( j_2 = (1, 3) \in J^e_2 \) that satisfies \( j_2 - d_2 = (1, 2) \in J^e_1 \) and an index \( j_1 = (1, 2) \) that satisfies \( j_1 - d_1 = (0, 3) \in J^e_2 \). Thus, the dependences \( (1, -1)^t \) and \((0, 1)^t\) belong to the interdependence matrices \( D^e_1 \) and \( D^e_2 \), respectively. That is

\[
D^e_1 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \quad D^e_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}
\]

and

\[
D^e = D^e_1 \cup D^e_2 = \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}.
\]

After obtaining the interdependence matrix \( D^e \) and index set \( J^e \), we can determine the parallel execution order of each index in each partition with a linear transformation function. All the indexes belonging to \( J^e_1 \) and \( J^e_2 \) will be mapped by the linear transformations \( \Pi^1 \) and \( \Pi^2 \), respectively. However, this mapping will ignore the interdependence relations between partitions \( P_1 \) and \( P_3 \). In order to preserve the valid execution order, some delay \( C_j \) should be added to each index \( j \).

In the next section, we will discuss how to determine the interdependence matrix \( D^e \), index set \( J^e \), and the delay \( C_j \) for each index \( j \).

III. THE NONLINEAR TRANSFORMATION ALGORITHM

In this section, we will describe the nonlinear transformation algorithm for the nested For-loops with partitions on the innermost loop. First, we consider the loops with two partitions on the innermost loop. Before describing the nonlinear algorithm, we will prove some lemmas for determining the interdependence matrix \( D^e \) and interdependence index set \( J^e \).

**Lemma 3.1:** A dependence vector \( d = (a_1, a_2, \ldots, a_n) \) is in \( D^e_1 \) if and only if

\[
0 < a_n < (u_n^2 - l_n^2 + 1).
\]

**Proof:** Necessary: Assume that \( d \) is an interdependence vector \( d = (a_1, a_2, \ldots, a_n) \) is a dependence vector, there at least exist two iterations \( j_1 \in J^e_1 \) and \( j_2 \in J^e_2 \) such that \( j_2 = j_1 + d \). Let \( j_2 = (b_1, b_2, \ldots, b_n) \) and \( j_1 = (c_1, c_2, \ldots, c_n) \). This implies \( a_n = b_n - c_n > 0 \). Furthermore, we want to prove the right side of the nonuniform equation. Because \( d \in D_1 \), there exist at least two indexes \( j_2 \) and \( j_2' \) in \( J^e_2 \) such that \( j_2' = j_2 + d \). Let \( j_2' = (b_1', b_2', \ldots, b_n') \), then \( b_n' - b_n = a_n \). Since \( l_n^2 \leq b_n' \leq b_n \) and \( l_n^2 \leq b_n \leq u_n^2 \), it implies that \( b_n' - b_n \leq u_n^2 - l_n^2 \). Thus, \( a_n < u_n^2 - l_n^2 + 1 \) is satisfied.

Sufficient: Assume \( d \in D_2 \) and \( 0 < a_n < (u_n^2 - l_n^2 + 1) \). We will prove that \( d \) is an interdependence vector. That is, we only need to prove there at least exists one pair of indexes \( (j_1, j_2) \) such that \( j_1 \in J^e_1 \), \( j_2 \in J^e_2 \), and \( j_2 = j_1 + d \) for \( d \in D_2 \). Let \( j_2 = (b_1, b_2, \ldots, b_n) \) and \( j_1 = (c_1, c_2, \ldots, c_n) \). The relation \( j_2 = j_1 + d \) implies that \( b_n = c_n + a_n \). If we take \( i \leq b_i \leq u_i \) for each \( i \leq n - 1 \) and \( b_n = u_n' \), then it is obvious that \( j_2 \in J^e_2 \). Since \( 0 < a_n < (u_n^2 - l_n^2 + 1) \), it implies that \( 2l_n^2 - u_n^2 - 1 \leq b_n - a_n \leq l_n^2 - 1 \). Therefore, \( 2l_n^2 - u_n^2 - 1 \leq c_n \leq l_n^2 - 1 \). That is, \( c = (c_1, c_2, \ldots, c_n) \) falls in \( J^e_1 \). The proof is completed.

**Lemma 3.2:** A dependence vector \( d = (a_1, a_2, \ldots, a_n) \) is in \( D^e_1 \) if and only if

\[
-(u_n^1 - l_n^1 + 1) < a_n < 0.
\]

**Proof:** The proof of Lemma 3.2 is similar to Lemma 3.1 and we omit the detail.

Applying Lemmas 3.1 and 3.2 to Example 2.1, we can determine that \( d_1 \) and \( d_3 \) are the interdependence vectors and thus \( D^e = \left[ d_1, d_3 \right] \).

Suppose that the interdependence vectors are bidirectional, and \( a_n^1 \) and \( a_n^2 \) are the maximal absolute values of the n-th element of all the interdependence vectors corresponding to \( D^e_1 \) and \( D^e_2 \), respectively. Then, the interdependence index set \( J^e \) can be derived from checking the length of the n-th element value of the interdependence vectors by the following lemma.

**Lemma 3.3:** The interdependence index set \( J^e \) is the union of \( J^e_1 \) and \( J^e_2 \), where

\[
J^e_1 = \{(j_1, j_2, \ldots, j_n) | b_i \leq j_i \leq u_i, \text{ for } i \leq n - 1; u_n^1 - a_n^1 + 1 \leq j_n \leq u_n^1 \} \text{ and } J^e_2 = \{(j_1, j_2, \ldots, j_n) | b_i \leq j_i \leq u_i, \text{ for } i \leq n - 1; l_n^2 \leq j_n \leq l_n^2 + a_n^1 - 1 \}.
\]

**Proof:** Let \( d^e = (a_1, a_2, \ldots, a_n-1, a_n^2) \) be an interdependence vector in \( D_2^e \). Assume \( j_1 = (c_1, c_2, \ldots, c_n) \in J^e_1 \), \( j_2 = (b_1, b_2, \ldots, b_n) \in J^e_2 \), and \( j_2 = j_1 + d^e \). Then \( c_n = b_n - a_n^2 \). Since \( a_n^2 \) is a known fixed value, the minimal value of \( c_n \) occurs when \( b_n = l_n^2 = u_n^1 + 1 \) and then \( u_n^1 + 1 - a_n^2 \leq b_n = a_n^2 \). Since \( j_1 \in J^e_1 \), it is clear that \( c_n \leq u_n^1 \). Thus, we obtain \( u_n^1 - a_n^2 + 1 \leq c_n \leq u_n^1 \). This proves that

\[
J^e_1 = \{(j_1, j_2, \ldots, j_n) | b_i \leq j_i \leq u_i, \text{ for } i \leq n - 1; u_n^1 - a_n^2 + 1 \leq j_n \leq u_n^1 \}.
\]
Similarly, we can derive the interdependence index set 
\( J'_2 = \{(j_1, j_2, \cdots, j_n) | 1 \leq j_i \leq u_i, \) 
for \( i \leq n - 1; l'^2_i \leq j_i \leq l'^2_i + a^2_n - 1 \}. \)

Applying Lemma 3.3 to Example 2.1, we obtain \( a^2_1 = 1 \) and \( a^2_2 = 1 \). The two interdependence index sets are
\( J'_1 = \{(j_1, j_2)|0 \leq j_1 \leq 5, 2 \leq j_2 \leq 2 \} \) and 
\( J'_2 = \{(j_1, j_2)|0 \leq j_1 \leq 5, 3 \leq j_2 \leq 3 \}. \)
Thus, the interdependence index set \( J' \) is shown in Fig. 1.

A. Nonlinear Transformation Algorithm for For-loops with 2-Partitions

In order to preserve the correct execution order of indexes in each partition, the linear transformation functions should satisfy the constraint described in [11]:
\[
\begin{align*}
\Pi'^1 \cdot d_i &> 0, \\
\Pi'^2 \cdot d_j &> 0 \\
&\text{for each } d_i \in D_1 \\
&\text{and each } d_j \in D_2. \quad (3.1)
\end{align*}
\]

However, in our nonlinear transformation method, one more constraint is required. Now, let us consider the following fraction of index dependence graph as shown in Fig. 1 with deleting the dependence vector \( d_2 \).

\[
\begin{align*}
\bar{J}'_2 & = (3, 3) \\
\bar{J}'_1 & = (3, 2) \bar{J}'_3 = (4, 2)
\end{align*}
\]

If we take \( \Pi'^1 = (0, -1) \), then indexes \( j_1 \) and \( j_3 \) can be executed concurrently. However, \( j_2 \) depends on \( j_1 \) and \( j_3 \) depends on \( j_2 \). The partial order relation \( j_1 \rightarrow j_3 \) contradicts to the arrangement of concurrent execution of \( j_1 \) and \( j_3 \). To avoid the occurrence of this situation, the linear transformation function \( \Pi'^1 \) and \( \Pi'^2 \) should also satisfy the following conditions:
\[
\begin{align*}
\Pi'^1 (d_i + d'_j) &> 0 \quad \text{and} \quad \Pi'^2 (d_i + d'_j) > 0 \\
&\text{for all } d_i \in D'_1 \text{ and } d'_j \in D'_2. \quad (3.2)
\end{align*}
\]

Let \( P_1 \) and \( P_2 \) be two partitions on the innermost For-loop and \( J'_1 \) and \( J'_2 \) be their corresponding index sets. Assume that \( \Pi'^1 \) and \( \Pi'^2 \) are the chosen linear transformation functions corresponding to \( J'_1 \) and \( J'_2 \), i.e., \( \Pi'^1 : J'_1 \rightarrow \text{Seq1} \) and \( \Pi'^2 : J'_2 \rightarrow \text{Seq2} \), where Seq1 and Seq2 are the mapped execution sequences corresponding to sets \( J'_1 \) and \( J'_2 \), respectively. If the linear transformation functions \( \Pi'^1 \) and \( \Pi'^2 \) satisfy the conditions (3.1) and (3.2), then there exists at least one executing sequence that can combine original sequences Seq1 and Seq2 such that the total execution order is correct.

We now focus on combining these two sequences which are determined by the linear transformation functions \( \Pi'^1 \) and \( \Pi'^2 \). Since there exist some interdependence vectors in the index set \( J' \), the execution of \( J'_1 \) and \( J'_2 \) should be adjusted such that their execution order can also guarantee the precedence relations in \( J' \). Let us now consider the case 3 of the dependence relation existing between \( J'_1 \) and \( J'_2 \). We try to derive a formula for determining the delay \( C_j \) of each index \( j \). For simplicity, we will start from the assumption that the interdependence matrices \( D'_1 \) and \( D'_2 \) both consist of one interdependence vector. Let \( D'_1 = \{d_1\} \) and \( D'_2 = \{d_2\} \). Later, we will discuss how to determine the delay of each index when the interdependence matrix consists of more than one interdependence vector. Assume that the execution time of index \( j \) is
\[
\begin{align*}
\Pi'^1 (j) &= \Pi'^1 (j) + C_j \quad \text{for each } j \in J' \cap J'_1 \quad \text{and} \quad (3.3) \\
\Pi'^2 (j) &= \Pi'^2 (j) + C_j \quad \text{for each } j \in J' \cap J'_2. \quad (3.4)
\end{align*}
\]

In order to preserve the correct execution order of each index in \( J' \), the following two equations should be satisfied:
\[
\begin{align*}
\Pi'^1 (j) - \Pi'^2 (j) - d_1 &= \Pi'^1 (d_1) \quad \text{for } j \in J'_1 \text{ and } j \neq d_1 \in J'_2 \quad (3.5) \\
\Pi'^1 (j) - \Pi'^2 (j) - d_2 &= \Pi'^1 (d_2) \quad \text{for } j \in J'_2 \text{ and } j \neq d_2 \in J'_1. \quad (3.6)
\end{align*}
\]

Substituting (3.3) and (3.4) into (3.5), we obtain
\[
C_j = (\Pi'^2 - \Pi'^1) (j - d_1) + C_{j - d_1}
\]
for \( j \in J'_1 \) and \( j = d_1 \in J'_2 \).

Similarly, substituting (3.3) and (3.4) into (3.6), we obtain
\[
C_j = (\Pi'^1 - \Pi'^2) (j - d_2) + C_{j - d_2}
\]
for \( j \in J'_2 \) and \( j = d_2 \in J'_1 \).

The delay of index \( j \in J'_1 \) can be derived from the index \( j = d_1 \) and its delay \( C_{j - d_1} \). Since index \( j \rightarrow d_1 \) falls in the region of the set \( J'_2 \) and index \( j \rightarrow d_2 \) falls in the region of set \( J'_1 \), we can obtain a recursive equation by the following derivation:
\[
\begin{align*}
C_j &= (\Pi'^2 - \Pi'^1) (j - d_1) + C_{j - d_1} \\
&= (\Pi'^2 - \Pi'^1) (j - d_1) \\
&+ (\Pi'^1 - \Pi'^2) (j - d_2) + C_{j - d_2} \\
&= (\Pi'^2 - \Pi'^1) (d_2) + C_{j - d_2} \\
\end{align*}
\]
where \( C_j \) is the delay of index \( j \in J'_1 \) and \( C_{j - d_1} \) is the delay of index \( j \rightarrow d_1 \). Assume that \( j_1 \in J'_1 \) is the first execution index with a delay \( C_{j_1} \), we will express the delay \( C_{j_2} \) of index \( j \) by \( C_{j_2} \), \( j_2 \) and \( j_1 \). Given the value of \( C_{j_1} \), and \( j_1 \), we can compute the delay of each index \( j \) by
\[
\begin{align*}
C_{j_1 + d_1 + d_2} &= (\Pi'^2 - \Pi'^1) (d_2) + C_{j_1} \\
C_{j_1 + 2d_1 + 2d_2} &= (\Pi'^2 - \Pi'^1) (2d_2) + C_{j_1 + d_1 + d_2} \\
&= 2(\Pi'^2 - \Pi'^1) (d_2) + C_{j_1} \\
&\vdots \\
C_{j_1 + kd_1 + kd_2} &= k(\Pi'^2 - \Pi'^1) (d_2) + C_{j_1}.
\end{align*}
\]
Let \( j = \tilde{j} + k(d^1 + d^2) \) or \( k = \frac{j - \tilde{j}}{d^1 + d^2} \), we obtain
\[
C_j = \frac{j - \tilde{j}}{d^1 + d^2}(\Pi^2 - \Pi^1)(d^2) + C_{\tilde{j}},
\]
where \( C_j \) is the delay of index \( j \in J^1 \) and \( C_{\tilde{j}} \) is the delay of index \( \tilde{j} \in J^2 \). Similarly,
\[
C_j = \frac{j - \tilde{j}}{d^1 + d^2}(\Pi^2 - \Pi^1)(d^1) + C_{\tilde{j}},
\]
where \( C_j \) is the delay of index \( j \in J^2 \) and \( C_{\tilde{j}} \) is the delay of index \( \tilde{j} \in J^2 \). Thus, the nonlinear transformation formula is
\[
\Pi^1(j) = \Pi^1(\tilde{j}) + C_j,
\]
where \( C_j = \frac{j - \tilde{j}}{d^1 + d^2}(\Pi^2 - \Pi^1)(d^2) + C_{\tilde{j}}, \)
for \( j \in J^1 \cap J^2 \) and
\[
\Pi^2(j) = \Pi^2(\tilde{j}) + C_j,
\]
where \( C_j = \frac{j - \tilde{j}}{d^1 + d^2}(\Pi^2 - \Pi^1)(d^1) + C_{\tilde{j}}, \)
for \( j \in J^2 \cap J^2 \),
(3.7)
Equation (3.7) is derived under the assumption of \( D^*_1 = \{d^1\} \) and \( D^*_2 = \{d^2\} \). In fact, there may exist more than one interdependence vector in the interdependence matrices \( D^*_1 \) and \( D^*_2 \). The following lemma will show how to determine the interdependence vectors \( d^1 \) and \( d^2 \) in formula (3.7) when there exist at least two interdependence vectors in \( D^*_1 \) and \( D^*_2 \).

Lemma 3.4: Assume that more than one interdependence vector is in \( D^*_1 \) and \( D^*_2 \). Then the choice of \( d^1 \) and \( d^2 \) that satisfy
\[
\Pi^2 \min \{\Pi^2 d^1\} \quad \text{for} \quad \forall d^1 \in D^*_1 \quad \text{and} \quad \Pi^1 \min \{\Pi^1 d^2\} \quad \text{for} \quad \forall d^2 \in D^*_2
\]
will preserve the same data dependence relation in index set \( J^e \) after the time transformation.

Proof: Without loss of generality, let \( D^*_1 = \{d^1_1, d^1_2, \ldots, d^1_{m_1}\} \) and index \( j \) depend on indexes \( j - d^1_1, j - d^1_2, \ldots, j - d^1_{m_1} \). In order to preserve the same data dependence relation in \( J^e \), we need to add a delay \( C_j \) to index \( j \) such that the index \( j \) is performed after the indexes \( j - d^1_1, j - d^1_2, \ldots, j - d^1_{m_1} \). Thus, we only need to find the last executed index \( j - d^1_1 \) in indexes \( j - d^1_1, j - d^1_2, \ldots, j - d^1_{m_1} \) and add the delay \( C_j \) to index \( j \) such that index \( j \) is performed after the last executed index \( j - d^1_{m_1} \). Then the execution of index \( j \) will be performed after all the indexes \( j - d^1_1, j - d^1_2, \ldots, j - d^1_{m_1} \). Since \( j - d^1_1 \) is the last executed index in set \( \{j - d^1_1, j - d^1_2, \ldots, j - d^1_{m_1}\} \), it is obvious that
\[
\Pi^2(\tilde{j} - d^1_{m_1}) = \max\{\Pi^2(\tilde{j} - d^1_1), \Pi^2(\tilde{j} - d^1_2), \ldots, \Pi^2(\tilde{j} - d^1_{m_1})\}.
\]
That is, \( \Pi^2(\tilde{j} - d^1_{m_1}) - \Pi^2(\tilde{j} - d^1_1) \geq 0 \) or \( \Pi^2(\tilde{j} - d^1_1) \leq \Pi^2(\tilde{j} - d^1_{m_1}) \) for \( \forall \tilde{j} \in D^*_1 \) and \( \tilde{j} \neq \tilde{j} \). This implies \( d^1_{m_1} = d^1_1 \). Similarly, we can prove the case of \( \Pi^1 d^2 = \min\{\Pi^1 d^2\} \) for \( \forall \tilde{j} \in D^*_2 \). This completes the proof of Lemma 3.4.

From (3.7), we need to find two linear transformation functions \( \Pi^1 \) and \( \Pi^2 \) for partitions \( P_1 \) and \( P_2 \), respectively. The transformation functions \( \Pi^1 \) and \( \Pi^2 \) must satisfy the following conditions:
\[
\Pi^1 d^1_j > 0 \quad \text{and} \quad \Pi^2 d^2_j > 0,
\]
for \( \forall d^1_j \in D^1 \) and \( \forall d^2_j \in D^2 \)
\[
\Pi^1(d^1_{m_1} + d^2_j) > 0 \quad \text{and} \quad \Pi^2(d^1_{m_1} + d^2_j) > 0,
\]
for \( \forall d^1_{m_1} \in D^1 \) and \( \forall d^2_j \in D^2 \),
(3.8)
After this arrangement, each index \( j^e \) that does not fall in the region of \( J^e \) can adjust its execution time by adding the delay \( C_j \) if indexes \( j^e \) and \( \tilde{j} \) are transformed to the same execution time and \( \tilde{j} \in J^e \).

Algorithm: Nonlinear Transformation for For-loops with 2-partitions

Input: An \( n \)-dimensional nested For-loops algorithm with two partitions on the innermost loop.

Output: A parallel executing sequence for each index \( j \in J \).

Step 1: Determine the dependence vector matrix \( D \).
Step 2: Determine the interdependence matrix \( D^e \) by applying Lemmas 3.1 and 3.2.
Step 3: Determine the index set \( J^e \) by applying Lemma 3.3 if \( D^e \) is not an empty set.
Step 4: Find two linear transformation functions \( \Pi^1 \) and \( \Pi^2 \) such that \( \Pi^1 \) and \( \Pi^2 \) satisfy the condition (3.8).
Step 5: Using (3.7) to rearrange the executing order of each index \( j \in J^e \).
Step 6: Rearrange the executing time of each index that falls out of the index region \( J^e \) by adding the delay \( C_j \) if indexes \( j^e \) and \( \tilde{j} \) have the same execution order and \( \tilde{j} \in J^e \).

By this algorithm, the total computing time of the original For-loops algorithm is
\[
T = \max\{\max\{\Pi^1 j\} - \min\{\Pi^1 j\}, \max\{\Pi^2 j\} - \min\{\Pi^2 j\}\}
\]
\[
= \max\{\max\{\Pi^1 j + C_j\}, \min\{\Pi^2 j + C_j\} - \min\{\Pi^2(j + C_j)\}\}, \quad \text{where} \quad j \in J^1 \text{ and } j \in J^2.
\]
Now, let us consider the For-loops algorithm in Example 2.1. The \( D^e \) and \( J^e \) are obtained in Section II. In step 4 we choose \( \Pi^1 = (1, 0) \) and \( \Pi^2 = (1, 1) \), which satisfy the constraint (3.8). Let \( j = (0, 2) \) and \( C_{\tilde{j}} = 0 \). We can evaluate the executing time of each index \( j \in J^e \). For example, the delay of index \( j = (3, 2) \) is evaluated as follows:
\[
C_j = \frac{j - \tilde{j}}{d^1 + d^2}(\Pi^2 - \Pi^1)(d^2) + C_{\tilde{j}}
\]
\[
= \frac{(3, 0)}{(1, 0)}((0, 1), (0, 1)) + 0 = 3.
\]
The executing order of index \( (3, 2) \) is
\[
\Pi^1(j) = \Pi^1(\tilde{j}) + C_j = (1, 0)(3, 2) + 3 = 6.
\]
All the indexes can be examined in such a way. The parallel execution order of the For-loops in Example 2.1 is shown in
B. Nonlinear Transformation of For-loops with p-Partitions

In this subsection, we will consider the case that the number of partitions on the innermost For-loop is more than two. We assume that there only exist dependence relations between any two adjacent partitioned index sets. First, we consider the case of three partitions on the innermost For-loop and then consider the case of p partitions on the innermost loop, for \( p \geq 3 \). Assume that the index set \( J \) can be partitioned into three subsets \( J_n^0, J_n^1, \) and \( J_n^2 \). Let \( D^j \) denote the dependence matrix corresponding to index set \( J_n^0 \), \( D_{ij}^j \) denote the interdependence matrix between \( J_n^0 \) and \( J_n^1 \), and \( J_{ij}^0 \) denote the interdependence index set between the index sets \( J_n^0 \) and \( J_n^1 \). The main idea is that we can first determine the execution order of each index in set \( J_n^0 \cup J_n^1 \) by applying the nonlinear transformation method. Then we consider the set \( J_n^0 \cup J_n^1 \) as a set \( J_n^2 \) and determine the execution order of each index in set \( J_n^0 \cup J_n^1 \) by applying the nonlinear transformation method again. Consider the block dependence graph with \( n = 2 \) as shown in Fig. 3(a). First, we choose three linear transformation functions \( \Pi^{0}, \Pi^{1}, \) and \( \Pi^{2} \). Each pair of \( \Pi^{0}, \Pi^{1}, \) and \( \Pi^{2} \) must satisfy the condition (3.8). The execution order of each index \( j \) in \( J_n^0 \cup J_n^1 \) can be determined using nonlinear transformation function \( \Pi \) described in Section III-A. Then, we adopt the same procedure when \( J_n^0 \) is included as shown in Fig. 3(b). Because the block dependence graph shown in Fig. 3 is bidirectional, we can use a nonlinear transformation function \( \Pi^{1} \) to determine the execution time of each index in \( J_n^0 \cup J_n^1 \):

\[
\Pi^{12}(j) = \begin{cases} 
\Pi^{12}(j) + C^{12}_j, & \text{for } j \in J_n^0 \cap J_n^1 \\
\Pi^{0}(j) + C^{0}_j, & \text{for } j \in J_n^0 \cap J_n^2
\end{cases}
\]

(3.9)

Now, we want to determine the delay values \( C^{12}_j \) and \( C^{0}_j \). Let \( \Delta C \) be the delay time difference of two successive execution indexes in \( J_n^1 \). According to the dependence relations between sets \( J_n^0 \) and \( J_n^1 \), we have

\[
\Pi^{12}(j) - \Pi^{0}(j - d^1) = \Pi^{0}(d^1) + \Delta C
\]

for \( d^1 \in D_n^1, j \in J_n^0 \cap J_n^1 \), and \( j - d^1 \in J_n^1 \cap J_n^0 \).

(3.10)

Substituting (3.9) into (3.10) and (3.11), we have

\[
\Pi^{12}(j) = \Pi^{12}(j) + C^{12}_j
\]

where \( C^{12}_j = \frac{\Delta C}{d^1 - d^0} (\Pi^{0} - \Pi^{1})(d^1)

+ C_j + C^{12}_j + \frac{\Delta C}{d^1 - d^0} \Delta C - C_j
\]

for \( j \in J_n^0, j - d^1 \in J_n^1, d^1 \in D_n^1 \cap D_1 \)

and \( d^0 \in D_n^0 \cap D_0 \).

(3.12)
In this example, the index set can be partitioned into three subsets
\[ J_0^2 = \{(j_1, j_2) | 0 \leq j_1 \leq 10, 0 \leq j_2 \leq 3\}, \]
\[ J_1^2 = \{(j_1, j_2) | 0 \leq j_1 \leq 10, 4 \leq j_2 \leq 7\}, \]
\[ J_2^2 = \{(j_1, j_2) | 0 \leq j_1 \leq 10, 8 \leq j_2 \leq 11\}, \]
and the index dependence graph is shown in Fig. 4. The dependence matrices corresponding to sets \( J_0^2 \), \( J_1^2 \), and \( J_2^2 \) are
\[ D^0 = \begin{bmatrix} 1 & 1 \\ -1 & 0 \end{bmatrix} = [\ddot{d}_1 \ddot{d}_2], \]
\[ D_1 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = [\ddot{d}_3 \ddot{d}_4], \]
\[ D_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = [\ddot{d}_5 \ddot{d}_6], \]
respectively. The interdependence matrices are
\[ D_{01}^\prime = \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} = [\ddot{d}_1 \ddot{d}_3] \quad \text{and} \]
\[ D_{12}^\prime = \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix} = [\ddot{d}_4 \ddot{d}_6]. \]

By applying Lemma 3.3, we obtain the interdependence index sets
\[ J_{01}^0 = \{(j_1, j_2) | 0 \leq j_1 \leq 10, 3 \leq j_2 \leq 4\} \quad \text{and} \quad J_{12}^0 = \{(j_1, j_2) | 0 \leq j_1 \leq 10, 7 \leq j_2 \leq 8\}. \]

First, we select linear transformation functions \( \Pi^{00} = (1, -1), \Pi^{01} = (1, 0) \), and \( \Pi^{02} = (1, 1) \). Applying the nonlinear transformation algorithm on the index sets \( J_0^2 \) and \( J_2^2 \), we can determine the execution order of each index \( j \in J_0^1 \cap J_2^1 \). Then, we can use (3.12) and (3.13) to determine the execution order of index \( j \in J_{01}^0 \) when the index set \( J_0^2 \) is included. For instance, given \( j_i = (0, 4), C_{j_i} = 0 \), and \( C_{j_i}^{12} = 0 \), the delay of index \( j = (6, 4) \) can be obtained by
\[ C_{j_i}^{12} = \frac{\ddot{j} - \ddot{j}_i}{d^1 + d^0} (\Pi^{00} - \Pi^{01}) (\ddot{d}^1) + C_{j_i}, \]
\[ + C_{j_i}^{12} = \frac{\ddot{j} - \ddot{j}_i}{d^1 + d^0} \Delta C - C_{j_i} \]
\[ = \frac{(6, 0)}{(2, 0)}, \]
\[ = \frac{(6, 0)}{(2, 0)} \times 1 - 6 = 0. \]

Thus, the execution order of index \( (6, 4) \) is
\[ \Pi^{02}(j) = \Pi^{12}(j) + C_{j_i}^{12} = \Pi^{11}(j) + C_{j_i}^{12} = 6 + 6 + 0 = 12. \]

Given \( j_i = (0, 3) \) and \( C_{j_i}^0 = 3 \), the delay of index \( (6, 3) \) can be obtained by
\[ C_{j_i}^0 = \frac{\ddot{j} - \ddot{j}_i}{d^1 + d^0} (\Pi^{02} - \Pi^{01}) (d^1) + C_{j_i}^0, \]
\[ + \frac{\ddot{j} - \ddot{j}_i}{d^1 + d^0} \Delta C \]
\[ = \frac{(6, 0)}{(2, 0)} (0, 1)(1, 1) + 3 + \frac{(6, 0)}{(2, 0)} \times 1 = 9. \]

The execution order of the indexes that fall out of the set \( J_{01}^0 \) can be determined according to the order of the nearest index \( j \in J_{01}^0 \). Now, let us consider the following example.

**Example 3.1:**

For \( j_1 = 0 \) to 10
For \( j_2 = 0 \) to 3
\[ A(j_1, j_2) = A(j_1 - 1, j_2 + 1) + A(j_1 - 1, j_2) \]
End \( j_2 \)
For \( j_2 = 4 \) to 7
\[ A(j_1, j_2) = A(j_1 - 1, j_2 - 1) * A(j_1 - 1, j_2 + 1) \]
End \( j_2 \)
For \( j_2 = 8 \) to 11
\[ A(j_1, j_2) = A(j_1 - 1, j_2) - A(j_1, j_2 - 1) \]
End \( j_2 \)
End \( j_1 \).
Thus, the execution order of index \((6, 3)\) is
\[
\Pi^{12}(j) = \Pi^0(j) + C^0 = (1, -1)(6, 3) + 9 = 12.
\]

All the indexes can be examined in such a way and the transformed order can be correctly executed in the parallel form. The final result of parallel execution order is shown in Fig. 5. All the indexes grouped together and marked with the same number can be executed concurrently. Based on the above derivation, we will describe the nonlinear transformation method when the number of partitions on the innermost loop is \(p\) for \(p \geq 3\). First, we select \(p\) linear transformation functions \(\Pi^{i0}, \Pi^{i1}, \ldots, \Pi^{ip-1}\). These \(p\) transformation functions should satisfy the following conditions:

\[
\Pi^{0}\tilde{d}_j > 0 \quad \text{for} \quad 0 \leq i \leq p - 1 \quad \text{and} \quad \forall \tilde{d}_j \in D^i;
\]
\[
\Pi^{i}(d^j_j + \tilde{d}_k) > 0 \quad \text{for} \quad 0 \leq i \leq p - 2;
\]
\[
\Pi^{i}(d^j_j + \tilde{d}_k) > 0 \quad \text{for} \quad 0 \leq i \leq p - 1;
\]
\[
\forall d^j_j \in D^i, \forall \tilde{d}_k \in D^i_{r+1} \quad \text{and} \quad \forall \tilde{d}_k \in D^i_{p-1}. \quad (3.14)
\]

The \(p\) partitions can be combined into \([p/2]\) sets, that is, \(S^0, S^1, \ldots, S^p \), where \(p1 = [p/2]\). Each index set \(J^p\) belongs to the set \(S^j\) if \(j = \lfloor i/2 \rfloor\). For example, the set \(S^0\) consists of \(J^0\) and \(J^0\) and the set \(S^1\) consists of \(J^p\) and \(J^p\). Then we will adjust the execution order of each index in set \(S^1\) by adding some delay, for \(0 \leq i \leq p1\). The delay of each index can be determined by formula (3.7). At the second stage, we will combine sets \(S^0, S^1, \ldots, S^p \) into \(p\) sets \(S^0, S^p, \ldots, S^{p-1}\), where \(p = [p/2]\). Each set \(S^1\) belongs to set \(S^2\) if \(j = \lfloor i/2 \rfloor\). We will adjust the execution order of each index \(j\) in \(S^2\) by adding some delay which can be derived similar to the derivation of formulas (3.12) and (3.13), for \(0 \leq i \leq p2\). At stage \(k\), we will combine sets \(S^0, \ldots, S^{pk-1}\) into \(pk\) sets \(S^0, \ldots, S^{pk}\), where \(pk = [(pk - 1)/2]\). And then we will adjust the execution order of each index in set \(S^1\), for \(0 \leq i \leq pk\). These steps will be continuous until the value of \(pk\) is 1. That is, the execution order of each index is adjusted under the consideration of \(p\)-partitions globally.

IV. Space Mapping Algorithm

In this section, we will be concerned with the mapping of For-Loop algorithms into VLSI systolic arrays.

A. Mapping Algorithms into Systolic Arrays

In this subsection, we first consider the space mapping under the assumption that the number of processors is equal to the number of iterations. The problem of mapping the algorithm into fixed size systolic arrays is considered in the next subsection. Before describing our method, we introduce
the space mapping proposed by Moldovan [11]. Moldovan proposed a space transformation matrix \( S \) to transform each index into one cell of systolic arrays. Each cell consists of a small number of registers, ALU, and control logic. A mesh-connected array processor is a tuple \( (Z^{n-1}, P) \), where \( Z^{n-1} \) is the index set of the array and \( P \in Z^{(n-1)\times r} \) is a matrix of interconnection primitives. The position of each processing cell is described by its Cartesian coordinates. The interconnection between cells is described by the difference vectors between the coordinates of adjacent cells. The matrix of interconnection primitives is \( P = [\bar{p}_1 \bar{p}_2 \cdots \bar{p}_r] \), where \( \bar{p}_j \) is a column vector indicating a unique direction of a communication link.

We can use a utilization matrix \( K = [k_{ji}] \) to multiply the interconnection matrix \( P \). Each entry \( k_{ji} \) of matrix \( K \) has a positive value to denote the number of utilizations of the \( j \)th column vector \( \bar{p}_j \) in matrix \( P \). The transformation \( S \) can then be selected such that the transformed dependencies \( SD \) are mapped into VLSI array modeled as \( (Z^{n-1}, P) \). This can be written as \( SD = PK \). This equation indicates that the total number of steps through interconnection links is equal to the distance of two processors that execute two indexes with data dependence matrix \( D \). Matrix \( K = [k_{ji}] \) is such that

\[
k_{ji} \geq 0 \quad \text{and} \quad 0 \leq \sum_j (k_{ji}) \leq \Pi \hat{d}_i.
\]

It is possible that some interconnection primitives will not be used. These correspond to rows of matrix \( K \) with zero elements.

In our space transformation algorithm, there are more constraints needed than Moldovan's space transformation algorithm. The most important thing is to guarantee the steps of data routing must be less than or equal to the time difference between data generated and data used. Consider the index dependence graph in Fig. 2 again, for a fixed index \( j \in J^n \), we should evaluate the time difference between data generated and data used corresponding to data dependence vector \( d_j \). Let \( \hat{j}_k \) be an index point in \( J^n \cap J^e \) and \( \hat{j}_{k+1} \) be the successor of \( \hat{j}_k \) according to the time transformation \( \Pi^1 \). By applying formula (3.6), the difference of the added time delays between \( \hat{j}_k \) and \( \hat{j}_{k+1} \) is equal to a constant, say \( C_1 \). That is, the delay difference

\[
C_{\hat{j}_{k+1}} - C_{\hat{j}_k} = C_{\hat{j}_k} - C_{\hat{j}_{k-1}} = \text{constant} \ C_1,
\]

where \( C_1 = \frac{d}{d^1 + d^2} (\Pi^2 - \Pi^1) (d^2) \).
Similarly, the added time delay difference between indexes \( j_k, j_{k+1} \in J^p_2 \cap J' \) is a constant \( C_2 \), where

\[
C_2 = \frac{d}{d^1 + d^2} (\Pi^1 - \Pi^2)(d^1).
\]

Therefore, if two successive execution indexes with dependence vector \( d_i \) in set \( J^p \) and set \( J^p \) and \( J^p_2 \), the time difference between them are \( \Pi^1 d_i + C_1 \) and \( \Pi^2 d_i + C_2 \), respectively.

For any two successive execution indexes \( j_k \) and \( j_{k+1} \in J^p_1 \), the space transformation function \( S \) should ensure that the number of steps of data communication between processors \( S(j_k) \) and \( S(j_{k+1}) \) is less than or equal to \( \Pi^1 d_i + C_1 \). Similarly, we should ensure that the number of communication steps of any successive execution indexes in \( J^p_2 \) is less than or equal to \( \Pi^2 d_i + C_2 \). Consider the index dependence graph as shown in Fig. 2; all indexes grouped by a bold line can be executed concurrently. Let \( G_i \) denote the execution group with execution order \( i \). There are two categories of execution groups. The first one is that each execution group has at least one index in \( J^p \) and the second one is that all indexes of each execution group do not belong to \( J^p \). Consider any two successive execution groups \( G_i \) and \( G_{i+1} \). If both of them belong to the first category, the execution time difference between these two groups is a constant value \( \Pi^1 d_i + C_1 \). If there is at least one execution group belonging to the second category, the time difference between these two groups is \( \Pi^1 d_i \). Thus, the utilization matrix \( K \) should satisfy the conditions

\[
\begin{align*}
\sum_i (k_{ij}) &\leq \Pi^1 d_i + C_1 & \text{for } d_i \in D^1 - D^p & \text{and} \\
\sum_i (k_{ij}) &\leq \Pi^1 d_i & \text{for } d_i \in D^1 \cap D^p.
\end{align*}
\]

Similarly, the matrix \( K \) should also satisfy the following constraints

\[
\begin{align*}
\sum_i (k_{ij}) &\leq \Pi^2 d_j + C_2 & \text{for } d_j \in D^2 - D^p & \text{and} \\
\sum_i (k_{ij}) &\leq \Pi^2 d_j & \text{for } d_j \in D^2 \cap D^p.
\end{align*}
\]

Consider the Example 2.1. Let \( S = \begin{bmatrix} a & c \\ b & d \end{bmatrix} \). The dependence vector matrix is

\[
D = \begin{bmatrix} 1 & 1 & 0 & 2 \\ -1 & 0 & 1 & -1 \end{bmatrix} = [d_1 d_2 d_3 d_4] = [d_1^1 d_2^1 d_3^1 d_4^1],
\]

By solving the equation \( SD = PK \), i.e.,

\[
\begin{bmatrix} a - c & a & c & 2a - c \\ b - d & b & d & 2b - d \end{bmatrix} = \begin{bmatrix} 0 & 1 & -1 & -1 & 1 & 0 & 1 & -1 \\ 0 & 1 & -1 & -1 & 1 & -1 & 1 & -1 \end{bmatrix} K,
\]

we have one possible solution:

\[
S = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \quad \text{and} \quad K = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix}.
\]

This solution satisfies the constraints (4.1) and (4.2) because

i) \( \sum_i (k_{i1}) = 1 \leq \Pi^1 d_i = 1 \)

ii) \( \sum_i (k_{i2}) = 1 \leq \Pi^1 d_2 + C_1 = 1 + 1 = 2 \)

iii) \( \sum_i (k_{i3}) = 1 \leq \Pi^2 d_3 = 1 \)

iv) \( \sum_i (k_{i4}) = 2 \leq \Pi^2 d_4 + C_2 = 1 + 1 = 2 \).

All the indexes can be mapped by the space transformation function \( S \). Each index \( j \) is assigned onto one processing cell \( S_j \) as shown in Fig. 6. After mapping index \( j \) into a processing cell \( (x, y) \), if the execution of index \( j \) depends on another index \( j' \) by a dependence vector \( d_i \), then the cell \( (x, y) \) has two links with direction \( S d_i \) as its input link and output link.

For instance, in Fig. 1 the execution of index (1,2) depends on the execution of indexes (0,3) and (0,2) with the dependence vectors \( d_1^1 = (1, -1, 1) \) and \( d_2^1 = (1, 0, 1) \), respectively. Since the index (1,2) executes on cell (1,-2), the cell should connect links with direction \( S d_1^1 = (1, 1) \) and \( S d_2^1 = (1, 0) \). These links can be determined by the utilization matrix \( K \). That is, there is a link in the direction of \( p_j \) if the value of \( k_{ji} \) is 1. The structure of array processors is shown in Fig. 6. For receiving data at the right time, the detail structure of each cell with delay buffers are shown in Fig. 7.

By applying the above space transformation, the processors size is equal to the number of iterations in original For-loops algorithm. This makes it impossible in technique and cost considerations. In practice, the number of iterations is usually larger than the number of processing cells. In the next subsection, we will pay attention to the problem of how to partition the mapped For-loops algorithm into bands such that each band can be executed on a fixed size array processor.

B. Partitioning Algorithms into Fixed Size Systolic Arrays

In this subsection, we first introduce our approach to the partitioning problem of Example 2.1 and then discuss the general case. We want to map the For-loops algorithm in Example 2.1 into a VLSI array with \( M \) processors and assume that the program size \( N >> M \). For instance, assume \( M = 6 \) and the index space is

\[
I(J) = \{(j_1, j_2) \mid 0 \leq j_1 \leq 11, 0 \leq j_2 \leq 11\}.
\]

Then the problem size is \( N = 12 \times 12 = 144 \). Assume there are two partitions \( J^p_1 \) and \( J^p_2 \) on the innermost For-loop, where

\[
J^p_1 = \{(j_1, j_2) \mid 0 \leq j_1 \leq 11, 0 \leq j_2 \leq 5\} \quad \text{and} \quad J^p_2 = \{(j_1, j_2) \mid 0 \leq j_1 \leq 11, 6 \leq j_2 \leq 11\}.
\]
A possible solution to this problem is illustrated in Fig. 8. We select the partitioning transformation \( \Pi_p \) to divide the index set \( L(J) \) in bands such that each band requires only six processors, three for performing the addition operations of index points in set \( J_1^2 \) and three for performing the multiplication operations of index points in set \( J_2^2 \). Note that we evenly distribute equal size of processors to each partition if the number of iterations of \( J_1^2 \) and \( J_2^2 \) is equal. The band boundaries must satisfy the following equation:

\[
\Pi_p \mod 3 = 0 \quad \text{for} \quad j \in J_1^2, \quad \text{and} \\
\Pi_p \mod 3 = 0 \quad \text{for} \quad j \in J_2^2. 
\] (4.3)

Equation (4.3) assumes that at any given moment no more than six index points in index set \( J \) are processed, in which three belong to index set \( J_1^2 \) and three belong to \( J_2^2 \). Let \( \Pi_p = (2, 1) \) and select the time transformation functions

\[
\Pi_1^2 j = (1,0) j + C_j \quad \text{for} \quad j \in J_1^2 \quad \text{and} \\
\Pi_2^2 j = (1,1) j + C_j \quad \text{for} \quad j \in J_2^2
\]

which determine the sequence of computations inside the bands. Each band is swept by lines whose equation is \( \Pi_1^2 j \) = constant for \( j \in J_1^2 \) and \( \Pi_2^2 j \) = constant for \( j \in J_2^2 \). All index points along each such line are executed simultaneously, and notice that there are at most three such points on each line inside one band. Inside each band, the indexes are executed in a partial ordering imposed by the time transformations \( \Pi_1^1 \) and \( \Pi_2^2 \). In Fig. 8, we mark the computation order of each such line by a positive integer aside it. The time hyperplanes \( \Pi_1^1 \) and \( \Pi_2^2 \) together with the partitioning hyperplanes \( \Pi_p \) which partitions the original For-loops algorithm and map it onto an array with only six processors. Each index \( j \) will be assigned to a band numbered \( b = \left\lfloor (\Pi_p j) / 3 \right\rfloor \) and the processor \( P_i \) for executing index \( j \) is determined by the formula \( i = \Pi_p j \mod 3 \). For example, index point \((1,3)\) is allocated to processor \( P_2 \) in band 2.

Consider Fig. 9, for a given band \( B_1 \), the data generated by index point \( j_1 \in J_1^2 \) \( J_2^2 \) should be used by another index point \( j_2 \in J_2^2 \cap J_1^2 \) in the near neighbor band \( B_2 \). Thus, these data should be transferred from processor numbered \( P_2 \) to the shift registers \( FIFO \). Similarly, the data generated by index point \( j_2 \in J_2^2 \) \( J_2^2 \) should be transferred from processor \( P_2 \) to \( FIFO \) used by the index \( j_1 \in J_1^2 \) \( J_1^2 \) in the near neighbor band. Moreover, the data generated by index \( j_1 \in J_1^2 \) in band \( B_1 \) may be used by another index \( j_2 \in J_2^2 \) in band \( B_2 \). These data should be transferred from processor \( P_2 \) into shift registers \( FIFO \) for use at the execution of band \( B_2 \). All the indexes can be checked that they can be processed with a correct executing order under our partitioning method. The maximum number of locations inside the \( FIFO \) A and \( FIFO \) \( B \) are both 6. In general, this number is related to the size of the problem.

The problem we address now is to map the \( n \)-depth For-loops algorithm into the systolic arrays with \( n - 1 \) dimensions. Assuming that the processor size is \( M = m_1 \times m_2 \times \cdots \times m_{n-1} \) and the problem size is \( N \), where \( N >> M \), we can easily partition the iterations into bands by the following partitioning algorithm. For simplicity, we assume that the size of \( p \) partitioned index sets are equal.

**Algorithm: Mapping and Partitioning Procedure**

**Input:** An \( n \)-depth For-Loops with \( p \) partitions on the innermost Loop.

**Output:** A parallel executing sequence that the number of concurrently executing indexes is less than or equal to the number of processors.

**Step 1:** Select \( p \) nonlinear transformation functions \( \Pi_0, \Pi_1, \cdots, \Pi^{p-1} \) by applying the Nonlinear Transformation Algorithm.

**Step 2:** Generate all possible utilization matrices \( K \) which satisfy conditions (4.1), (4.2), and (4.3).

**Step 3:** Find all possible space transformations \( S \) which
satisfy the conditions
a) $S$ can be solved from equation $SD = PK$.
b) The transformation matrix $T = \begin{bmatrix} \Pi_1^T \\ \Pi_2^T \\ \vdots \\ \Pi_{n-1}^T \end{bmatrix}$ is non-singular.

Step 4: For each valid transformation $T$, the partitioning hyperplanes $\Pi_{pi}$ are given by the row of matrix $S$

$$S = \begin{bmatrix} \Pi_{p1} \\ \Pi_{p2} \\ \vdots \\ \Pi_{p(n-1)} \end{bmatrix}.$$  

Step 5: Partition the index set $J$ into bands such that each index $j$ in band boundaries satisfies the equation

$$\left(\Pi_{pj} \mod m_i\right) = 0 \quad \text{for} \quad 1 \leq i \leq n-2 \quad \text{and} \quad \left(\Pi_{pj} \mod (m_i/p)\right) = 0 \quad \text{for} \quad i = n-1.$$

Step 6: From all the possible transformations select the one which requires the least number of bands.

Step 7: The mapping of indexes to processors is as follows:
- each index point $j$ is processed in a processor whose $i$th coordinate is $\Pi_{pi} j \mod m_i$.

Note that index point $j$ is partitioned into band $B_{b_1 b_2 \cdots b_{n-1}}$ whose $i$th coordinate $b_i$ is $[\Pi_{pj} j / 3]$. A possible policy that orders the execution of bands in a lexicographical manner can be selected by the same way described by Moldovan [11]. That is, for fixed $\Pi_{p1}, \cdots, \Pi_{p(n-2)}$ execute all bands given by $\Pi_{p(n-1)}$ then change $\Pi_{p(n-2)}$ and execute all $\Pi_{p(n-1)}$ again, etc.

Fig. 7. The detail layout of each cell in Fig. 6.
algorithm is also constructed. For a selected space mapping function $S$, the layout of each cell in the systolic array can be determined. An index $j \in J^2$ can be processed with an order $H_j^2$ on the number ordered $Sj$. Besides, the partitioning algorithm is also presented for executing the mapped algorithm on the fixed size array processors. We select partitioning functions which partition the iterations into bands such that there are at most $m$ iterations concurrently executing on the array processors with size $m$ at any given moment.

V. CONCLUSION

In this paper, we propose a nonlinear transformation algorithm to solve the For-loops with partitions on the innermost loop. For each partitioned index set $J^2$, we select a nonlinear transformation function $H^2$ to map all the indexes onto a parallel execution form. All the indexes $j \in J^2$ satisfy the equation $H^2j = C$ can be processed concurrently, where $C$ is a constant. Furthermore, the structure of the $(n-1)$-dimensional systolic arrays for implementing the mapped

REFERENCES


**Jang-Ping Sheu** (S’85–M’86) was born in Taiwan, Republic of China, on April 28, 1959. He received the B.S. degree in computer science from Tamkang University, Taiwan, in June 1981 and the M.S. and Ph.D. degrees in computer science from the National Tsing Hua University, Taiwan, in June 1983 and January 1987, respectively. He joined the faculty of the Department of Electrical Engineering at the National Central University, Taiwan, as an Associate Professor in February 1987. His current research interests include parallel processing and distributed computing systems.

**Chih-Yung Chang** received the B.S. degree in information engineering from the Chung-Yung University in 1988 and the M.S. degree in information engineering from the Tatung Institute of Technology in 1990. Since September 1990, he has been working toward the Ph.D. degree in the Department of Electrical Engineering, National Central University, Taiwan, Republic of China. His research interests are parallel algorithms and parallel compilers.