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 transformation, 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 concurrence 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 concurrence 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

Manuscript received October 2, 1990; revised March 4, 1991. This work was supported by the National Science Council of the Republic of China under Grant NSC 80-0408-E-008-09.

The authors are with the Department of Electrical Engineering, National Central University, Chung-Li 32054, Taiwan, R.O.C.

IEEE Log Number 9100387.

index variable j_i in a sequential loop program. Assume that the data dependence relation exists in two statements S_1 and S_2 and $S_1(\bar{j}_1) \rightarrow S_2(\bar{j}_2)$, where $\bar{j}_1$ and $\bar{j}_2 \in I^n$ [2]. The *data dependence vector* is defined by $\bar{d} = \bar{j}_2 - \bar{j}_1$. The positive value of dependence vector $\bar{d}$ indicates that the execution of two statements $S_1(\bar{j}_1)$ and $S_2(\bar{j}_2)$ performs with $\bar{d}$ iterations difference.

The hyperplane method proposed by Moldovan [11] applies a linear time function Π to transform each $\bar{d}$ into $\Pi\bar{d}$. To ensure a correct execution order, Π must satisfy the condition $\Pi\bar{d}_i > 0$ for each dependence vector $\bar{d}_i$. Assume that any single computation or set computations performed at one index point $\bar{j} \in I^n$ takes one unit time; thus, a computation indexed by $\bar{j}$ in the original algorithm will be processed at time $\Pi\bar{j}$. By using this transformation function Π, the iteration indexes $\bar{j}_1$ and $\bar{j}_2 \in I^n$ satisfied $\Pi(\bar{j}_1) = \Pi(\bar{j}_2)$ 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 ith 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
  ⋮
  For $j_{n-1} = l_{n-1}$ to u_{n-1} by k_{n-1}
    For $j_n = l_n^1$ to u_n^1 by k_n^1
      P_1
    End j_n
    For $j_n = l_n^2$ to u_n^2 by k_n^2
      P_2
    End j_n
    ⋮
    For $j_n = l_n^p$ to u_n^p by k_n^p
      P_p
    End j_n
  End j_{n-1}
  ⋮
End j_1 (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^i and u_n^i 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 $l_n^{i+1} = u_n^i + 1$ 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^n and J_2^n denote the corresponding index sets performing on partitions P_1 and P_2, respectively. There are three cases of dependence relation between J_1^n and J_2^n.

Case 1: Some iterations of J_2^n data dependent on some iterations of J_1^n.

Case 2: Some iterations of J_1^n data dependent on some iterations of J_2^n.

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 $\bar{d}_i$ is an *interdependence vector*, denoted as $\bar{d}_i^e$, if

$$\bar{d}_i \in D_1 \quad \text{and} \quad \exists\, \bar{j} \in J_1^n \quad \text{such that } \bar{j} - \bar{d}_i \in J_2^n \quad \text{or}$$
$$\bar{d}_i \in D_2 \quad \text{and} \quad \exists\, \bar{j} \in J_2^n \quad \text{such that } \bar{j} - \bar{d}_i \in J_1^n.$$

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

Definition 2-2: The *interdependence index set* J^e is defined as

$$\begin{aligned} J^e = &\{\bar{j} \mid \bar{j} \in J_1^n \text{ and } \bar{j} - \bar{d}_i \in J_2^n\} \\ &\cup \{\bar{j} - \bar{d}_i \mid \bar{j} \in J_1^n \text{ and } \bar{j} - \bar{d}_i \in J_2^n\} \\ &\cup \{\bar{j} \mid \bar{j} \in J_2^n \text{ and } \bar{j} - \bar{d}_j \in J_1^n\} \\ &\cup \{\bar{j} - \bar{d}_j \mid \bar{j} \in J_2^n \text{ and } \bar{j} - \bar{d}_j \in J_1^n\} \\ &\text{for all } \bar{d}_i \in D_1 \text{ and } \bar{d}_j \in D_2. \end{aligned}$$

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

$$J_1^2 = \{(j_1, j_2) \mid 0 \leq j_1 \leq 5, 0 \leq j_2 \leq 2\} \quad \text{and}$$
$$J_2^2 = \{(j_1, j_2) \mid 0 \leq j_1 \leq 5, 3 \leq j_2 \leq 5\}.$$

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

$$\bar{d}_1 = (1, -1)^t = \bar{d}_1^e, \quad \bar{d}_2 = (1, 0)^t,$$
$$d_3 = (0, 1)^t = \bar{d}_3^e, \quad \text{and} \quad \bar{d}_4 = (2, -1)^t.$$

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

$$D_1 = \begin{bmatrix} 1 & 1 \\ -1 & 0 \end{bmatrix} = [\bar{d}_1 \bar{d}_2] \quad \text{and} \quad D_2 = \begin{bmatrix} 0 & 2 \\ 1 & -1 \end{bmatrix} = [\bar{d}_3 \bar{d}_4],$$

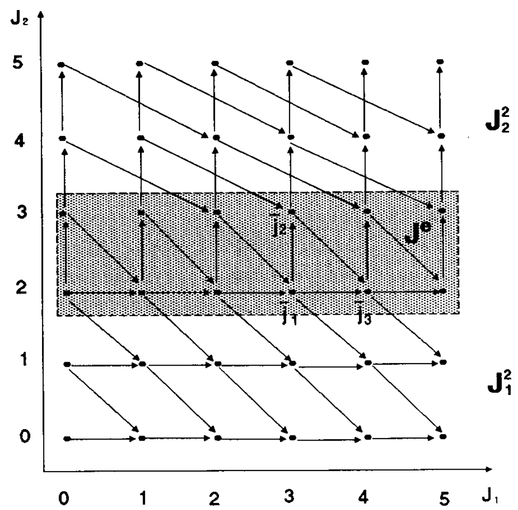


Fig. 1. The index dependence graph of Example 2.1.

respectively. Obviously, from the index dependence graph as shown in Fig. 1, the interdependence index set is $J^e = \{(j_1, j_2) | 0 \le j_1 \le 5, 2 \le j_2 \le 3\}$. By Definition 2-1, we can find an index $\bar{j}_2 = (1,3) \in J_2^2$ that satisfies $\bar{j}_2 - \bar{d}_3 = (1,2) \in J_1^2$ and an index $\bar{j}_1 = (1,2)$ that satisfies $\bar{j}_1 - \bar{d}_1 = (0,3) \in J_2^2$. Thus, the dependence vectors $(1,-1)^t$ and $(0,1)^t$ belong to the interdependence matrices D_1^e and D_2^e, respectively. That is

$$D_1^e = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \qquad D_2^e = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad \text{and}$$

$$D^e = D_1^e \cup D_1^e = \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}.$$

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_1^2 and J_2^2 will be mapped by the linear transformations Π^{*1} and Π^{*2}, respectively. However, this mapping will ignore the interdependence relations between partitions P_1 and P_2. In order to preserve the valid execution order, some delay $C_{\bar{j}}$ should be added to each index $\bar{j}$.

In the next section, we will discuss how to determine the interdependence matrix D^e, index set J^e, and the delay $C_{\bar{j}}$ for each index $\bar{j}$.

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 $\bar{d} = (a_1, a_2, \cdots, a_n) \in D_2^e$ if and only if

$$0 < a_n < (u_n^2 - l_n^2 + 1).$$

Proof: Necessary: Assume that $\bar{d}$ is an interdependence vector $\in D_2^e$. Since $\bar{d} = (a_1, a_2, \cdots, a_n)$ is a dependence vector, there at least exist two iterations $\bar{j}_1 \in J_1^n$ and $\bar{j}_2 \in J_2^n$ such that $\bar{j}_2 = \bar{j}_1 + \bar{d}$. Let $\bar{j}_2 = (b_1, b_2, \cdots, b_n)$ and $\bar{j}_1 = (c_1, c_2, \cdots, c_n)$. This implies $a_n = b_n - c_n > 0$. Furthermore, we want to prove the right side of the nonuniform equation. Because $\bar{d} \in D_2$, there exist at least two indexes $\bar{j}_2$ and $\bar{j}_2' \in J_2^n$ such that $\bar{j}_2' = \bar{j}_2 + \bar{d}$. Let $\bar{j}_2' = (b_1', b_2', \cdots, b_n')$, then $b_n' - b_n = a_n$. Since $l_n^2 \le b_n' \le u_n^2$ and $l_n^2 \le b_n \le u_n^2$, it implies that $b_n' - b_n \le u_n^2 - l_n^2$. Thus, $a_n < u_n^2 - l_n^2 + 1$ is satisfied.

Sufficient: Assume $\bar{d} \in D_2$ and $0 < a_n < (u_n^2 - l_n^2 + 1)$. We will prove that $\bar{d}$ is an interdependence vector. That is, we only need to prove there at least exists one pair of indexes $(\bar{j}_1, \bar{j}_2)$ such that $\bar{j}_1 \in J_1^n, \bar{j}_2 \in J_2^n$, and $\bar{j}_2 = \bar{j}_1 + \bar{d}$ for $\bar{d} \in D_2$. Let $\bar{j}_2 = (b_1, b_2, \cdots, b_n)$ and $\bar{j}_1 = (c_1, c_2, \cdots, c_n)$. The relation $\bar{j}_2 = \bar{j}_1 + \bar{d}$ implies that $b_n = c_n + a_n$. If we take $l_i \le b_i \le u_i$ for each $i \le n-1$ and $b_n = l_n^2$, then it is obvious that $\bar{j}_2 \in J_2^n$. Since $0 < a_n < (u_n^2 - l_n^2 + 1)$, it implies that $2l_n^2 - u_n^2 - 1 \le b_n - a_n \le l_n^2 - 1$. Therefore, $2l_n^2 - u_n^2 - 1 \le c_n \le l_n^2 - 1$. That is, $c = (c_1, c_2, \cdots, c_n)$ falls in J_1^n. The proof is completed. □

Lemma 3.2: A dependence vector $\bar{d} = (a_1, a_2, \cdots, a_n) \in D_1^e$ 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 $\bar{d}_1$ and $\bar{d}_3$ are the interdependence vectors and thus $D^e = [\bar{d}_1 \bar{d}_3]$.

Suppose that the interdependence vectors are bidirectional, and $\boldsymbol{a}_n^1$ and $\boldsymbol{a}_n^2$ are the maximal absolute values of the nth element of all the interdependence vectors corresponding to D_1^e and D_2^e, respectively. Then, the interdependence index set J^e can be derived from checking the length of the nth element value of the interdependence vectors by the following lemma.

Lemma 3.3: The interdependence index set J^e is the union of J_1^e and J_2^e, where

$$J_1^e = \{(j_1, j_2, \cdots, j_n) | l_i \le j_i \le u_i, \text{ for } i \le n-1; u_n^1 - \boldsymbol{a}_n^2 + 1 \le j_n \le u_n^1\} \quad \text{and}$$

$$J_2^e = \{(j_1, j_2, \cdots, j_n) | l_i \le j_i \le u_i, \text{ for } i \le n-1; l_n^2 \le j_n \le l_n^2 + \boldsymbol{a}_n^1 - 1\}.$$

Proof: Let $\bar{d}^e = (a_1, a_2, \cdots, a_{n-1}, \boldsymbol{a}_n^2)$ be an interdependence vector $\in D_2^e$. Assume $\bar{j}_1 = (c_1, c_2, \cdots, c_n) \in J_1^n, \bar{j}_2 = (b_1, b_2, \cdots, b_n) \in J_2^n$, and $\bar{j}_2 = \bar{j}_1 + \bar{d}^e$. Then $c_n = b_n - \boldsymbol{a}_n^2$. Since $\boldsymbol{a}_n^2$ is a known fixed value, the minimal value of c_n occurs when $b_n = l_n^2 = u_n^1 + 1$ and then $u_n^1 + 1 - \boldsymbol{a}_n^2 \le b_n - \boldsymbol{a}_n^2$. Since $\bar{j}_1 \in J_1^n$, it is clear that $c_n \le u_n^1$. Thus, we obtain $u_n^1 - \boldsymbol{a}_n^2 + 1 \le c_n \le u_n^1$. This proves that

$$J_1^e = \{(j_1, j_2, \cdots, j_n) | l_i \le j_i \le u_i, \text{ for } i \le n-1; u_n^1 - \boldsymbol{a}_n^2 + 1 \le j_n \le u_n^1\}.$$

Similarly, we can derive the interdependence index set

$$J_2^e = \{(j_1, j_2, \cdots, j_n) | l_i \le j_i \le u_i, \\ \text{for } i \le n-1; l_n^2 \le j_n \le l_n^2 + a_n^1 - 1\}. \quad \square$$

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

$$J_1^e = \{(j_1, j_2) | 0 \le j_1 \le 5, 2 \le j_2 \le 2\} \quad \text{and} \\ J_2^e = \{(j_1, j_2) | 0 \le j_1 \le 5, 3 \le j_2 \le 3\}.$$

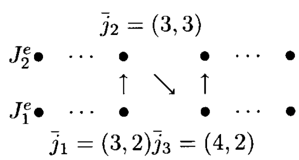
Thus, the interdependence index set J^e is shown in Fig. 1.

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]:

$$\Pi^{*1}\bar{d}_i > 0, \qquad \Pi^{*2}\bar{d}_j > 0 \\ \text{for each } \bar{d}_i \in D_1 \\ \text{and each } \bar{d}_j \in D_2. \tag{3.1}$$

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 $\bar{d}_2$.



If we take $\Pi^{*1} = (0, -1)$, then indexes $\bar{j}_1$ and $\bar{j}_3$ can be executed concurrently. However, $\bar{j}_2$ depends on $\bar{j}_1$ and $\bar{j}_3$ depends on $\bar{j}_2$. The partial order relation $\bar{j}_1 \to \bar{j}_3$ contradicts to the arrangement of concurrent execution of $\bar{j}_1$ and $\bar{j}_3$. To avoid the occurrence of this situation, the linear transformation function Π^{*1} and Π^{*2} should also satisfy the following conditions:

$$\Pi^{*1}(\bar{d}_i^e + \bar{d}_j^e) > 0 \quad \text{and} \quad \Pi^{*2}(\bar{d}_i^e + \bar{d}_j^e) > 0 \\ \text{for all } \bar{d}_i^e \in D_1^e \text{ and } \bar{d}_j^e \in D_2^e. \tag{3.2}$$

Let P_1 and P_2 be two partitions on the innermost For-loop and J_1^n and J_2^n be their corresponding index sets. Assume that Π^{*1} and Π^{*2} are the chosen linear transformation functions corresponding to J_1^n and J_2^n, i.e., $\Pi^{*1} : J_1^n \to$ *Seq*1 and $\Pi^{*2} : J_2^n \to$ *Seq*2, where *Seq*1 and *Seq*2 are the mapped execution sequences corresponding to sets J_1^n and J_2^n, respectively. If the linear transformation functions Π^{*1} and Π^{*2} satisfy the conditions (3.1) and (3.2), then there exists at least one executing sequence that can combine original sequences *Seq*1 and *Seq*2 such that the total execution order is correct.

We now focus on combining these two sequences which are determined by the linear transformation functions Π^{*1} and Π^{*2}. Since there exist some interdependence vectors in the index set J^e, the execution of J_1^n and J_2^n should be adjusted such that their execution order can also guarantee the precedence relations in J^e. Let us now consider the case 3 of the dependence relation existing between J_1^n and J_2^n. We try to derive a formula for determining the delay $C_{\bar{j}}$ of each index $\bar{j}$. For simplicity, we will start from the assumption that the interdependence matrices D_1^e and D_2^e both consist of one interdependence vector. Let $D_1^e = \{\bar{d}^1\}$ and $D_2^e = \{\bar{d}^2\}$. 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 $\bar{j}$ is

$$\Pi^1(\bar{j}) = \Pi^{*1}(\bar{j}) + C_{\bar{j}} \quad \text{for each } \bar{j} \in J^e \cap J_1^n \quad \text{and} \tag{3.3}$$

$$\Pi^2(\bar{j}) = \Pi^{*2}(\bar{j}) + C_{\bar{j}} \quad \text{for each } \bar{j} \in J^e \cap J_2^n. \tag{3.4}$$

In order to preserve the correct execution order of each index in J^e, the following two equations should be satisfied:

$$\Pi^1(\bar{j}) - \Pi^2(\bar{j} - \bar{d}^1) = \Pi^{*1}(\bar{d}^1) \quad \text{for } \bar{j} \in J_1^n \text{ and } \bar{j} - \bar{d}^1 \in J_2^n \tag{3.5}$$

$$\Pi^2(\bar{j}) - \Pi^1(\bar{j} - \bar{d}^2) = \Pi^{*2}(\bar{d}^2) \quad \text{for } \bar{j} \in J_2^n \text{ and } \bar{j} - \bar{d}^2 \in J_1^n. \tag{3.6}$$

Substituting (3.3) and (3.4) into (3.5), we obtain

$$C_{\bar{j}} = (\Pi^{*2} - \Pi^{*1})(\bar{j} - \bar{d}^1) + C_{\bar{j} - \bar{d}^1} \\ \text{for } \bar{j} \in J_1^n \text{ and } \bar{j} - \bar{d}^1 \in J_2^n.$$

Similarly, substituting (3.3) and (3.4) into (3.6), we obtain

$$C_{\bar{j}} = (\Pi^{*1} - \Pi^{*2})(\bar{j} - \bar{d}^2) + C_{\bar{j} - \bar{d}^2} \\ \text{for } \bar{j} \in J_2^n \text{ and } \bar{j} - \bar{d}^2 \in J_1^n.$$

The delay of index $\bar{j} \in J_1^n$ can be derived from the index $\bar{j} - \bar{d}^1$ and its delay $C_{\bar{j} - \bar{d}^1}$. Since index $\bar{j} - \bar{d}^1$ falls in the region of the set J_2^n and index $\bar{j} - \bar{d}^1 - \bar{d}^2$ falls in the region of set J_1^n, we can obtain a recursive equation by the following derivation:

$$\begin{aligned} C_{\bar{j}} &= (\Pi^{*2} - \Pi^{*1})(\bar{j} - \bar{d}^1) + C_{\bar{j} - \bar{d}^1} \\ &= (\Pi^{*2} - \Pi^{*1})(\bar{j} - \bar{d}^1) \\ &\quad + (\Pi^{*1} - \Pi^{*2})(\bar{j} - \bar{d}^1 - \bar{d}^2) + C_{\bar{j} - \bar{d}^1 - \bar{d}^2} \\ &= (\Pi^{*2} - \Pi^{*1})(\bar{d}^2) + C_{\bar{j} - \bar{d}^1 - \bar{d}^2} \end{aligned}$$

where $C_{\bar{j}}$ is the delay of index $\bar{j} \in J_1^n$ and $C_{\bar{j} - \bar{d}^1 - \bar{d}^2}$ is the delay of index $\bar{j} - \bar{d}^1 - \bar{d}^2 \in J_1^n$. Assume that $\bar{j}_i \in J_1^n$ is the first execution index with a delay $C_{\bar{j}_i}$, we will express the delay $C_{\bar{j}_i}$ of index $\bar{j}$ by $C_{\bar{j}_i}$ and $\bar{j}_i$. Given the value of $C_{\bar{j}_i}$ and $\bar{j}_i$, we can compute the delay of each index $\bar{j}$ by

$$\begin{aligned} C_{\bar{j}_i + \bar{d}^1 + \bar{d}^2} &= (\Pi^{*2} - \Pi^{*1})(\bar{d}^2) + C_{\bar{j}_i} \\ C_{\bar{j}_i + 2\bar{d}^1 + 2\bar{d}^2} &= (\Pi^{*2} - \Pi^{*1})(\bar{d}^2) + C_{\bar{j}_i + \bar{d}^1 + \bar{d}^2} \\ &= 2(\Pi^{*2} - \Pi^{*1})(\bar{d}^2) + C_{\bar{j}_i} \\ &\vdots \\ C_{\bar{j}_i + k\bar{d}^1 + k\bar{d}^2} &= k(\Pi^{*2} - \Pi^{*1})(\bar{d}^2) + C_{\bar{j}_i}. \end{aligned}$$

Let $\bar{j} = \bar{j}_i + k(\bar{d}^1 + \bar{d}^2)$ or $k = \frac{\bar{j}-\bar{j}_i}{\bar{d}^1+\bar{d}^2}$, we obtain

$$C_{\bar{j}} = \frac{\bar{j} - \bar{j}_i}{\bar{d}^1 + \bar{d}^2}(\Pi^{*2} - \Pi^{*1})(\bar{d}^2) + C_{\bar{j}_i}$$

where $C_{\bar{j}}$ is the delay of index $\bar{j} \in J_1^n$ and $C_{\bar{j}_i}$ is the delay of index $\bar{j}_i \in J_1^n$. Similarly,

$$C_{\bar{j}} = \frac{\bar{j} - \bar{j}_i}{\bar{d}^1 + \bar{d}^2}(\Pi^{*1} - \Pi^{*2})(\bar{d}^1) + C_{\bar{j}_i}$$

where $C_{\bar{j}}$ is the delay of index $\bar{j} \in J_2^n$ and $C_{\bar{j}_i}$ is the delay of index $\bar{j}_i \in J_2^n$. Thus, the nonlinear transformation formula is

$$\begin{aligned} \Pi^1(\bar{j}) &= \Pi^{*1}(\bar{j}) + C_{\bar{j}} \\ &\text{where } C_{\bar{j}} = \frac{\bar{j} - \bar{j}_i}{\bar{d}^1 + \bar{d}^2}(\Pi^{*2} - \Pi^{*1})(\bar{d}^2) + C_{\bar{j}_i}, \\ &\qquad \text{for } \bar{j} \in J_1^n \cap J^e \quad \text{and} \\ \Pi^2(\bar{j}) &= \Pi^{*2}(\bar{j}) + C_{\bar{j}} \\ &\text{where } C_{\bar{j}} = \frac{\bar{j} - \bar{j}_i}{\bar{d}^1 + \bar{d}^2}(\Pi^{*1} - \Pi^{*2})(\bar{d}^1) + C_{\bar{j}_i}, \\ &\qquad \text{for } \bar{j} \in J_2^n \cap J^e. \end{aligned} \tag{3.7}$$

Equation (3.7) is derived under the assumption of $D_1^e = \{\bar{d}^1\}$ and $D_2^e = \{\bar{d}^2\}$. In fact, there may exist more than one interdependence vector in the interdependence matrices D_1^e and D_2^e. The following lemma will show how to determine the interdependence vectors $\bar{d}^1$ and $\bar{d}^2$ in formula (3.7) when there exist at least two interdependence vectors in D_1^e and D_2^e.

Lemma 3.4: Assume that more than one interdependence vector is in D_1^e and D_2^e. Then the choice of $\bar{d}^1$ and $\bar{d}^2$ that satisfy

$$\begin{aligned} \Pi^{*2}\bar{d}^1 &= \min\{\Pi^{*2}\bar{d}_i^e\} \quad \text{for } \forall\, \bar{d}_i^e \in D_1^e \quad \text{and} \\ \Pi^{*1}\bar{d}^2 &= \min\{\Pi^{*1}\bar{d}_j^e\} \quad \text{for } \forall\, \bar{d}_j^e \in D_2^e \end{aligned}$$

will preserve the same data dependence relation in index set J^e after the time transformation.

Proof: Without loss of generality, let $D_1^e = \{\bar{d}_1^e, \bar{d}_2^e, \cdots, \bar{d}_k^e\}$ and index $\bar{j}$ depend on indexes $\bar{j} - \bar{d}_1^e, \bar{j} - \bar{d}_2^e, \cdots, \bar{j} - \bar{d}_k^e$. In order to preserve the same data dependence relation in J^e, we need to add a delay $C_{\bar{j}}$ to index $\bar{j}$ such that the index $\bar{j}$ is performed after the indexes $\bar{j} - \bar{d}_1^e, \bar{j} - \bar{d}_2^e, \cdots, \bar{j} - \bar{d}_k^e$. Thus, we only need to find the last executed index $\bar{j} - \bar{d}_l^e$ in indexes $\{\bar{j} - \bar{d}_1^e, \bar{j} - \bar{d}_2^e, \cdots, \bar{j} - \bar{d}_k^e\}$ and add the delay $C_{\bar{j}}$ to index $\bar{j}$ such that index $\bar{j}$ is performed after the last executed index $\bar{j} - \bar{d}_i^e$. Then the execution of index $\bar{j}$ will be performed after all the indexes $\bar{j} - \bar{d}_1^e, \bar{j} - \bar{d}_2^e, \cdots, \bar{j} - \bar{d}_k^e$. Since $\bar{j} - \bar{d}_l^e$ is the last executed index in set $\{\bar{j} - \bar{d}_1^e, \bar{j} - \bar{d}_2^e, \cdots, \bar{j} - \bar{d}_k^e\}$, it is obvious that

$$\Pi^{*2}(\bar{j} - \bar{d}_l^e) = \max\{\Pi^{*2}(\bar{j} - \bar{d}_1^e), \Pi^{*2}(\bar{j} - \bar{d}_2^e), \cdots, \Pi^{*2}(\bar{j} - \bar{d}_k^e)\}.$$

That is, $\Pi^{*2}(\bar{j} - \bar{d}_l^e) - \Pi^{*2}(\bar{j} - \bar{d}_i^e) \geq 0$ or $\Pi^{*2}(\bar{d}_l^e) \leq \Pi^{*2}(\bar{d}_i^e)$ for $\forall\, \bar{d}_i^e \in D_1^e$ and $\bar{d}_i^e \neq \bar{d}_l^e$. This implies $\bar{d}_l^e = \bar{d}^1$. Similarly, we can prove the case of $\Pi^{*1}\bar{d}^2 = \min\{\Pi^{*1}\bar{d}_j^e\}$ for $\forall\, \bar{d}_j^e \in D_2^e$. This completes the proof of Lemma 3.4. □

From (3.7), we need to find two linear transformation functions Π^{*1} and Π^{*2} for partitions P_1 and P_2, respectively. The transformation functions Π^{*1} and Π^{*2} must satisfy the following conditions:

$$\begin{aligned} &\Pi^{*1}\bar{d}_i > 0 \quad \text{and} \quad \Pi^{*2}\bar{d}_j > 0, \\ &\qquad \text{for } \forall\, \bar{d}_i \in D_1 \text{ and } \forall\, \bar{d}_j \in D_2 \\ &\Pi^{*1}(\bar{d}_i^e + \bar{d}_j^e) > 0 \quad \text{and} \quad \Pi^{*2}(\bar{d}_i^e + \bar{d}_j^e) > 0, \\ &\qquad \text{for } \forall\, \bar{d}_i^e \in D_1^e \text{ and } \forall \bar{d}_j^e \in D_2^e. \end{aligned} \tag{3.8}$$

After this arrangement, each index $\bar{j}'$ that does not fall in the region of J^e can adjust its execution time by adding the delay $C_{\bar{j}}$ if indexes $\bar{j}'$ and $\bar{j}$ are transformed to the same execution time and $\bar{j} \in J^e$.

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 $\bar{j} \in J$.

- 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 Π^{*1} and Π^{*2} such that Π^{*1} and Π^{*2} satisfy the condition (3.8).
- Step 5: Using (3.7) to rearrange the executing order of each index $\bar{j} \in J^e$.
- Step 6: Rearrange the executing time of each index that falls out of the index region J^e by adding the delay $C_{\bar{j}}$ if indexes $\bar{j}'$ and $\bar{j}$ have the same execution order and $\bar{j} \in J^e$.

By this algorithm, the total computing time of the original For-loops algorithm is

$$\begin{aligned} T &= \max\{\max(\Pi^1\bar{j}) - \min(\Pi^1\bar{j}),\ \max(\Pi^2\bar{j}') \\ &\quad - \min(\Pi^2\bar{j}')\} \\ &= \max\{\max(\Pi^{*1}\bar{j} + C_{\bar{j}}) \\ &\quad - \min(\Pi^{*1}\bar{j} + C_{\bar{j}}),\ \max(\Pi^{*2}\bar{j}' + C_{\bar{j}'}) \\ &\quad - \min(\Pi^{*2}\bar{j}' + C_{\bar{j}'})\}, \quad \text{where } \bar{j} \in J_1^n \text{ and } \bar{j}' \in J_2^n. \end{aligned}$$

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 $\bar{j}_i = (0,2)$ and $C_{\bar{j}_i} = 0$. We can evaluate the executing time of each index $\bar{j}$ in J^e. For example, the delay of index $\bar{j} = (3,2)$ is evaluated as follows:

$$\begin{aligned} C_{\bar{j}} &= \frac{\bar{j} - \bar{j}_i}{\bar{d}^1 + \bar{d}^2}(\Pi^{*2} - \Pi^{*1})(\bar{d}^3) + C_{\bar{j}_i} \\ &= \frac{(3,0)}{(1,0)}(0,1), (0,1) + 0 = 3. \end{aligned}$$

The executing order of index (3,2) is

$$\Pi^1(\bar{j}) = \Pi^{*1}(\bar{j}) + C_{\bar{j}} = (1,0)(3,2) + 3 = 6.$$

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

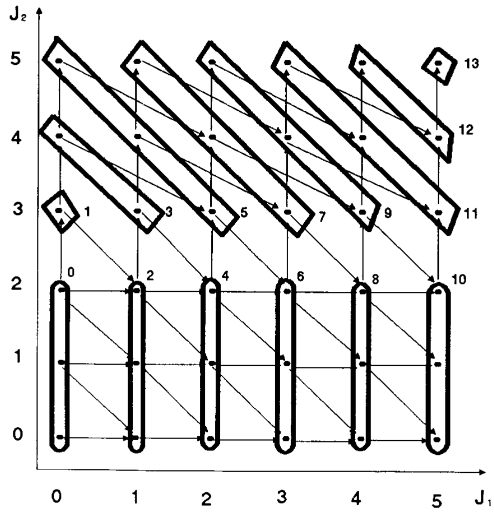


Fig. 2. The parallel execution order of Fig. 1.

Fig. 2. Note that all the indexes grouped by a bold line can be executed concurrently.

For comparing the total executing time of using linear and nonlinear transformation method, we assume that index set $J = \{(j_1, j_2)\} | 0 \leq j_1 \leq N, 0 \leq j_2 \leq N\}$ and

$$J_1^n = \{(j_1, j_2) | 0 \leq j_1 \leq N, 0 \leq j_2 \leq \lfloor N/2 \rfloor\}$$
$$J_2^n = \{(j_1, j_2) | 0 \leq j_1 \leq N, \lfloor N/2 \rfloor + 1 \leq j_2 \leq N\}.$$

The total execution time T_1 by using linear transformation method with $\Pi = (2, 1)$ is

$$T_1 = \left\lceil \frac{1 + \max\{\Pi(\bar{j}_1 - \bar{j}_2)\}}{\min \Pi \bar{d}_i} \right\rceil = \frac{1 + (2N + 1)}{1} = 3N + 1.$$

The total execution time T_2 by using our algorithm is

$$\begin{aligned} T_2 &= \max\{\max(\Pi^{*1}\bar{j} + C_{\bar{j}}) - \min(\Pi^{*1}\bar{j} + C_{\bar{j}}), \\ &\quad \max(\Pi^{*2}\bar{j}' + C_{\bar{j}'}) - \min(\Pi^{*2}\bar{j}' + C_{\bar{j}'})\} \\ &= \max\{2N, 2N + N - (\lfloor N/2 \rfloor + 1)\} \\ &= 3N - \lfloor N/2 \rfloor - 1. \end{aligned}$$

Thus, the ratio of total execution time improved in this example by using nonlinear transformation method is

$$(T_1 - T_2)/T_1 = (\lfloor N/2 \rfloor + 2)/(3N + 1) \doteq \frac{1}{6}.$$

Note that, if the dependence relations between the partitioned index sets J_1^n and J_2^n are not bidirectional, formula (3.7) can be simplified from the modification of expression (3.3) and (3.4). If index sets J_1^n depends on index set J_2^n, the formula (3.4) can be changed into $\Pi^2(\bar{j}) = \Pi^{*2}(\bar{j})$. Similarly, if index set J_2^n depends on index set J_1^n, we change the formula (3.3) into $\Pi^1(\bar{j}) = \Pi^{*1}(\bar{j})$.

In this subsection, we have described the nonlinear transformation method for the For-loops with only two partitions on the innermost loop. In fact, there may be more than two partitions on the innermost loop. We will state the nonlinear transformation algorithm for the case of p-partitions in the next subsection.

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_0^n, J_1^n, and J_2^n. Let D^i denote the dependence matrix corresponding to index set J_i^n, D_{ij}^e denote the interdependence matrix between J_i^n and J_j^n, and J_{ij}^e denote the interdependence index set between the index sets J_i^n and J_j^n. The main idea is that we can first determine the execution order of each index in set $J_1^n \cup J_2^n$ by applying the nonlinear transformation method. Then we consider the set $J_1^n \cup J_2^n$ as a set J_{12}^n and determine the execution order of each index in set $J_0^n \cup J_{12}^n$ 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 Π^{*0}, Π^{*1}, and Π^{*2}. Each pair of Π^{*0}, Π^{*1}, and Π^{*2} must satisfy the condition (3.8). The execution order of each index $\bar{j}$ in $J_1^n \cup J_2^n$ can be determined by using nonlinear transformation function $\Pi = \Pi^{12}$ described in Section III-A. Then, we adopt the same procedure when J_0^n 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 Π^{012} to determine the execution time of each index in $J_0^n \cup J_{12}^n$:

$$\Pi^{012}(\bar{j}) = \begin{cases} \Pi'^{12}(\bar{j}) = \Pi^{12}(\bar{j}) + C_{\bar{j}}^{12}, & \text{for } \bar{j} \in J_{01}^e \cap J_1^n \\ \Pi^0(\bar{j}) = \Pi^{*0}(\bar{j}) + C_{\bar{j}}^0, & \text{for } \bar{j} \in J_{01}^e \cap J_0^n. \end{cases} \tag{3.9}$$

Now, we want to determine the delay values $C_{\bar{j}}^{12}$ and $C_{\bar{j}}^0$. Let ΔC be the delay time difference of two successive execution indexes in J_1^n. According to the dependence relations between sets J_1^n and J_0^n, we have

$$\Pi'^{12}(\bar{j}) - \Pi^0(\bar{j} - \bar{d}^1) = \Pi^{*1}(\bar{d}^1) + \Delta C$$
$$\text{for } \bar{d}^1 \in D_1^e, \bar{j} \in J_1^n \cap J_{01}^e, \text{ and } \bar{j} - \bar{d}^1 \in J_0^n \cap J_{01}^e. \tag{3.10}$$

$$\Pi^0(\bar{j}) - \Pi'^{12}(\bar{j} - \bar{d}^0) = \Pi^{*0}(\bar{d}^0)$$
$$\text{for } \bar{d}^0 \in D_0^e, \bar{j} \in J_0^n \cap J_{01}^e, \text{ and } \bar{j} - \bar{d}^0 \in J_1^n \cap J_{01}^e. \tag{3.11}$$

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

$$\begin{aligned} &\Pi^{12}(\bar{j}) = \Pi^{*12}(\bar{j}) + C_{\bar{j}}^{12} \\ &\text{where } C_{\bar{j}}^{12} = \frac{\bar{j} - \bar{j}_i}{\bar{d}^1 - \bar{d}^0}(\Pi^{*0} - \Pi^{*1})(\bar{d}^0) \\ &\qquad + C_{\bar{j}_i} + C_{\bar{j}_i}^{12} + \frac{\bar{j} - \bar{j}_i}{\bar{d}^1 + \bar{d}^0}\Delta C - C_{\bar{j}} \\ &\qquad \text{for } \bar{j} \in J_1^n, \bar{j} - \bar{d}^1 \in J_0^n, \bar{d}^1 \in D_{01}^e \cap D_1 \\ &\qquad \text{and } \bar{d}^0 \in D_{01}^e \cap D^0, \end{aligned} \tag{3.12}$$

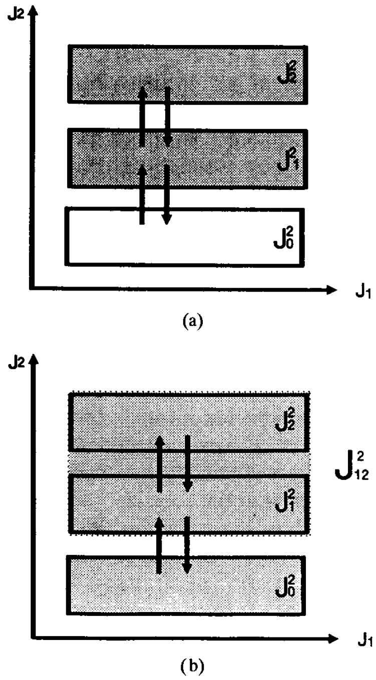


Fig. 3. (a) The block dependence graph of sets J_1^2 and J_2^2. (b) The block dependence graph of sets J_{12}^2 and J_0^2.

$$\begin{aligned}\Pi^0(\bar{j}) &= \Pi^{*0}(\bar{j}) + C_{\bar{j}}^0 \\ \text{where } C_{\bar{j}}^0 &= \frac{\bar{j}-\bar{j}_i}{\bar{d}^1-\bar{d}^0}(\Pi^{*1}-\Pi^{*0})(\bar{d}^1) \\ &\quad + C_{\bar{j}_i}^0 + \frac{\bar{j}-\bar{j}_i}{\bar{d}^1+\bar{d}^0}\Delta C \\ &\text{for } \bar{j}\in J_0^n, \bar{j}-\bar{d}^0 \in J_1^n, \bar{d}^1 \in D_{01}^e \cap D_1 \\ &\text{and } \bar{d}^0 \in D_{01}^e \cap D^0. \qquad (3.13)\end{aligned}$$

The execution order of the indexes that fall out of the set J_{01}^e can be determined according to the order of the nearest index $\bar{j} \in J_{01}^e$. 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.

In this example, the index set can be partitioned into three subsets

$$\begin{aligned}J_0^2 &= \{(j_1, j_2) \mid 0 \le j_1 \le 10, 0 \le j_2 \le 3\}, \\ J_1^2 &= \{(j_1, j_2) \mid 0 \le j_1 \le 10, 4 \le j_2 \le 7\}, \\ J_2^2 &= \{(j_1, j_2) \mid 0 \le j_1 \le 10, 8 \le j_2 \le 11\},\end{aligned}$$

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

$$\begin{aligned}D^0 &= \begin{bmatrix} 1 & 1 \\ -1 & 0 \end{bmatrix} = [\bar{d}_1 \bar{d}_2], \\ D_1 &= \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = [\bar{d}_3 \bar{d}_4], \quad \text{and} \\ D_2 &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = [\bar{d}_5 \bar{d}_6],\end{aligned}$$

respectively. The interdependence matrices are

$$\begin{aligned}D_{01}^e &= \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} = [\bar{d}_1 \bar{d}_3] \quad \text{and} \\ D_{12}^e &= \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix} = [\bar{d}_4 \bar{d}_6].\end{aligned}$$

By applying Lemma 3.3, we obtain the interdependence index sets

$$\begin{aligned}J_{01}^e &= \{(j_1, j_2) \mid 0 \le j_1 \le 10, 3 \le j_2 \le 4\} \quad \text{and} \\ J_{12}^e &= \{(j_1, j_2) \mid 0 \le j_1 \le 10, 7 \le j_2 \le 8\}.\end{aligned}$$

First, we select linear transformation functions $\Pi^{*0} = (1, -1), \Pi^{*1} = (1, 0)$, and $\Pi^{*2} = (1, 1)$. Applying the nonlinear transformation algorithm on the index sets J_1^n and J_2^n, we can determine the execution order of each index $\bar{j} \in J_1^n \cap J_2^n$. Then, we can use (3.12) and (3.13) to determine the execution order of index $\bar{j} \in J_{01}^e$ when the index set J_0^2 is included. For instance, given $\bar{j}_i = (0, 4), C_{\bar{j}_i} = 0$, and $C_{\bar{j}_i}^{12} = 0$, the delay of index $\bar{j} = (6, 4)$ can be obtained by

$$\begin{aligned}C_{\bar{j}}^{12} &= \frac{\bar{j}-\bar{j}_i}{\bar{d}^1+\bar{d}^0}(\Pi^{*0}-\Pi^{*1})(\bar{d}^0) + C_{\bar{j}_i} \\ &\quad + C_{\bar{j}_i}^{12} + \frac{\bar{j}-\bar{j}_i}{\bar{d}^1+\bar{d}^0}\Delta C - C_{\bar{j}} \\ &= \frac{(6,0)}{(2,0)}(0,-1)(1,-1) + 0 + 0 \\ &\quad + \frac{(6,0)}{(2,0)} \times 1 - 6 = 0.\end{aligned}$$

Thus, the execution order of index $(6, 4)$ is

$$\begin{aligned}\Pi^{012}(\bar{j}) &= \Pi^{12}(\bar{j}) + C_{\bar{j}}^{12} \\ &= \Pi^{*1}(\bar{j}) + C_{\bar{j}} + C_{\bar{j}}^{12} = 6 + 6 + 0 = 12.\end{aligned}$$

Given $\bar{j}_i = (0, 3)$ and $C_{\bar{j}_i}^0 = 3$, the delay of index $(6, 3)$ can be obtained by

$$\begin{aligned}C_{\bar{j}}^0 &= \frac{\bar{j}-\bar{j}_i}{\bar{d}^1+\bar{d}^0}(\Pi^{*1}-\Pi^{*0})(\bar{d}^1) + C_{\bar{j}_i}^0 + \frac{\bar{j}-\bar{j}_i}{\bar{d}^1+\bar{d}^0}\Delta C \\ &= \frac{(6,0)}{(2,0)}(0,1)(1,1) + 3 + \frac{(6,0)}{(2,0)} \times 1 = 9.\end{aligned}$$

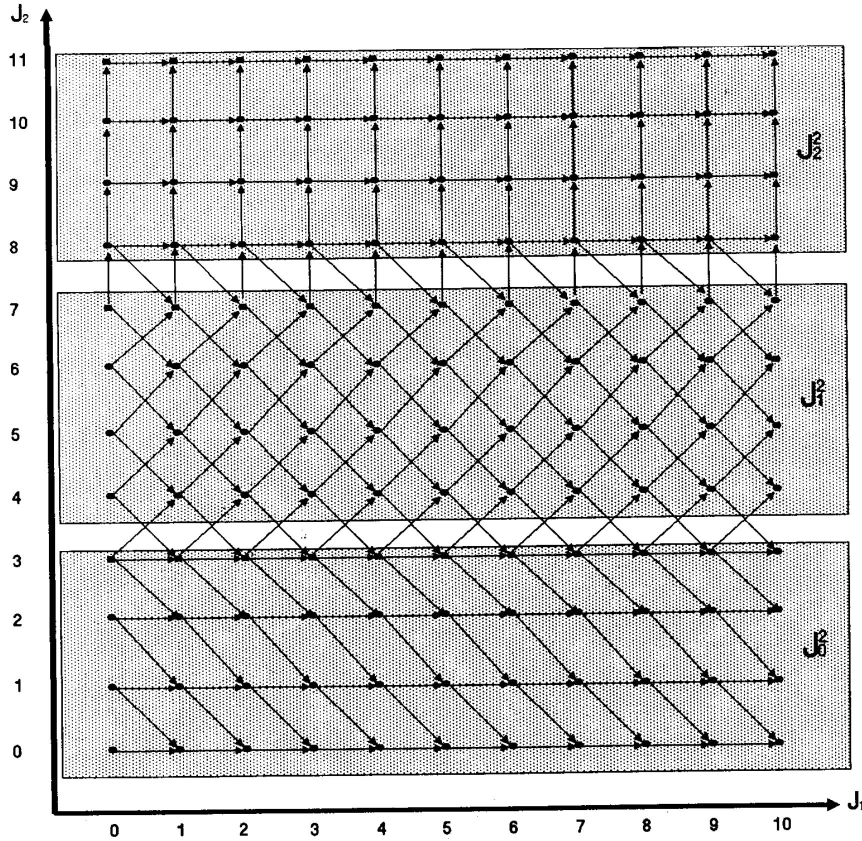


Fig. 4. The index dependence graph of Example 3.1.

Thus, the execution order of index (6, 3) is

$$\Pi^{012}(\bar{j}) = \Pi^{*0}(\bar{j}) + C_{\bar{j}}^{0} = (1,-1)(6,3) + 9 = 12.$$

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 $\Pi^{*0}, \Pi^{*1}, \cdots, \Pi^{*p-1}$.These p transformation functions should satisfy the following conditions:

$$\begin{aligned}
&\Pi^{*i}\bar{d}_j > 0 && \text{for } 0 \leq i \leq p-1 \text{ and } \forall\ \bar{d}_j \in D^i;\\
&\Pi^{*i}(\bar{d}_j^e + \bar{d}_k^e) > 0 && \text{for } 0 \leq i \leq p-2;\\
&\Pi^{*i}(\bar{d}_j^e + \bar{d}_l^e) > 0 && \text{for } 0 \leq i \leq p-1;\\
&\qquad \forall\ \bar{d}_j^e \in D_i^e, \forall\ \bar{d}_k^e \in D_{i+1}^e \text{ and } \forall\ \bar{d}_l^e \in D_{i-1}^e. && (3.14)
\end{aligned}$$

The p partitions can be combined into $\lceil p/2 \rceil$ sets, that is, $S_1^0, S_1^1, \cdots, S_{p1}^1$, where $p1 = \lceil p/2 \rceil$. Each index set J_i^n belongs to the set S_j^1 if $j = \lfloor i/2 \rfloor$. For example, the set S_0^1 consists of J_0^n and J_1^n and the set S_1^1 consists of J_2^n and J_3^n. Then we will adjust the execution order of each index in set S_i^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 $S_0^1, S_1^1, \cdots, S_{p1}^1$ into $p2$ sets $S_0^2, S_1^2, \cdots, S_{p2}^2$, where $p2 = \lceil p1/2 \rceil$. Each set S_i^1 belongs to set S_j^2 if $j = \lfloor i/2 \rfloor$. We will adjust the execution order of each index $\bar{j}$ in S_i^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 $S_0^{k-1}, \cdots, S_{pk-1}^{k-1}$ into pk sets $S_0^k, \cdots, S_{pk}^k$, where $pk = \lceil (pk-1)/2 \rceil$. And then we will adjust the execution order of each index in set S_i^k, 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

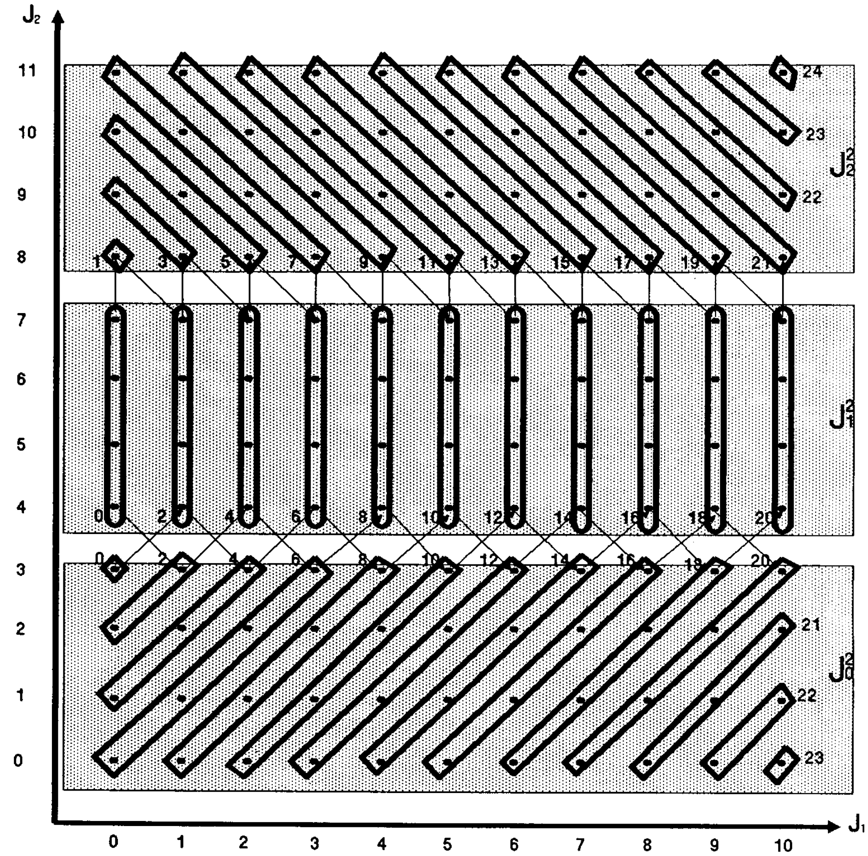


Fig. 5. The parallel execution sequence of Fig. 4.

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 jth 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

$$k_{ji} \ge 0 \quad \text{and} \quad 0 \le \sum_j (k_{ji}) \le \Pi \bar{d}_i.$$

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 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 $\bar{j} \in J_1^n$, we should evaluate the time difference between data generated and data used corresponding to data dependence vector $\bar{d}_i$. Let $\bar{j}_k$ be an index point in $J_1^n \cap J^e$, and $\bar{j}_{k+1}$ be the successor of $\bar{j}_k$ according to the time transformation Π^1. By applying formula (3.6), the difference of the added time delays between $\bar{j}_k$ and $\bar{j}_{k+1}$ is equal to a constant, say C_1. That is, the delay difference

$$C_{\bar{j}_{k+1}} - C_{\bar{j}_k} = C_{\bar{j}_k} - C_{\bar{j}_{k-1}} = \text{ constant } C_1,$$
$$\text{where } C_1 = \frac{\bar{d}}{\bar{d}^1 + \bar{d}^2}\left(\Pi^{*2} - \Pi^{*1}\right)\left(\bar{d}^2\right).$$

Similarly, the added time delay difference between indexes $\bar{j}_k, \bar{j}_{k+1} \in J_2^n \cap J^e$ is a constant C_2, where

$$C_2 = \frac{\bar{d}}{\bar{d}^1 + \bar{d}^2}(\Pi^{*1} - \Pi^{*2})(\bar{d}^1).$$

Therefore, if two successive execution indexes with dependence vector $\bar{d}_i$ in set $J^e \cap J_1^n$ and set $J^e \cap J_2^n$, the time difference between them are $\Pi^{*1}\bar{d}_i + C_1$ and $\Pi^{*2}\bar{d}_i + C_2$, respectively.

For any two successive execution indexes $\bar{j}_k$ and $\bar{j}_{k+1} \in J_1^e$, the space transformation function S should ensure that the number of steps of data communication between processors $S(\bar{j}_k)$ and $S(\bar{j}_{k+1})$ is less than or equal to $\Pi^{*1}\bar{d}_i + C_1$. Similarly, we should ensure that the number of communication steps of any successive execution indexes in J_2^e is less than or equal to $\Pi^{*2}\bar{d}_i + C_2$. 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^e and the second one is that all indexes of each execution group do not belong to J^e. Consider any two successive execution groups G_i and $G_{i+1} \subset J_1^n$. If both of them belong to the first category, the execution time difference between these two groups is a constant value $\Pi^{*1}\bar{d}_j + C_1$. If there is at least one execution group belonging to the second category, the time difference between these two groups is $\Pi^{*1}\bar{d}_j$. Thus, the utilization matrix K should satisfy the conditions

$$\begin{aligned} &\sum_i (k_{ij}) \le \Pi^{*1}\bar{d}_j + C_1 && \text{for } \bar{d}_j \in D^1 - D^e \quad \text{and} \\ &\sum_i (k_{ij}) \le \Pi^{*1}\bar{d}_j && \text{for } \bar{d}_j \in D^1 \cap D^e. \end{aligned} \tag{4.1}$$

Similarly, the matrix K should also satisfy the following constraints

$$\begin{aligned} &\sum_i (k_{ij}) \le \Pi^{*2}\bar{d}_j + C_2 && \text{for } \bar{d}_j \in D^2 - D^e \quad \text{and} \\ &\sum_i (k_{ij}) \le \Pi^{*2}\bar{d}_j && \text{for } \bar{d}_j \in D^2 \cap D^e. \end{aligned} \tag{4.2}$$

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

$$D = \begin{bmatrix} 1 & 1 & 0 & 2 \\ -1 & 0 & 1 & -1 \end{bmatrix} = [\bar{d}_1\bar{d}_2\bar{d}_3\bar{d}_4] = [\bar{d}_1^e\bar{d}_2\bar{d}_3^e\bar{d}_4].$$

By solving the equation $SD = PK$, i.e.,

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

we have one possible solution:

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

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

i) $\sum_i (k_{i1}) = 1 \le \Pi^{*1}\bar{d}_1^e = 1$

ii) $\sum_i (k_{i2}) = 1 \le \Pi^{*1}\bar{d}_2 + C_1 = 1 + 1 = 2$

iii) $\sum_i (k_{i3}) = 1 \le \Pi^{*2}\bar{d}_3^e = 1$

iv) $\sum_i (k_{i4}) = 2 \le \Pi^{*2}\bar{d}_4 + C_2 = 1 + 1 = 2.$

All the indexes can be mapped by the space transformation function S. Each index $\bar{j}$ is assigned onto one processing cell $S\bar{j}$ as shown in Fig. 6. After mapping index $\bar{j}$ into a processing cell (x, y), if the execution of index $\bar{j}$ depends on another index $\bar{j}'$ by a dependence vector $\bar{d}_i$, then the cell (x, y) has two links with direction $S\bar{d}_i$ 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 $\bar{d}_1^e = (1, -1)^t$ and $\bar{d}_2 = (1, 0)^t$, respectively. Since the index $(1, 2)$ executes on cell $(1, -2)$, the cell should connect links with direction $S\bar{d}_1^e = (1, 1)$ and $S\bar{d}_1 = (1, 0)$. 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 fixed size array processors.

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

$$L(J) = \{(j_1, j_2) | 0 \le j_1 \le 11, 0 \le j_2 \le 11\}.$$

Then the problem size is $N = 12 \times 12 = 144$. Assume there are two partitions J_1^2 and J_2^2 on the innermost For-loop, where

$$J_1^2 = \{(j_1, j_2) | 0 \le j_1 \le 11, 0 \le j_2 \le 5\} \quad \text{and}$$
$$J_2^2 = \{(j_1, j_2) | 0 \le j_1 \le 11, 6 \le j_2 \le 11\}.$$

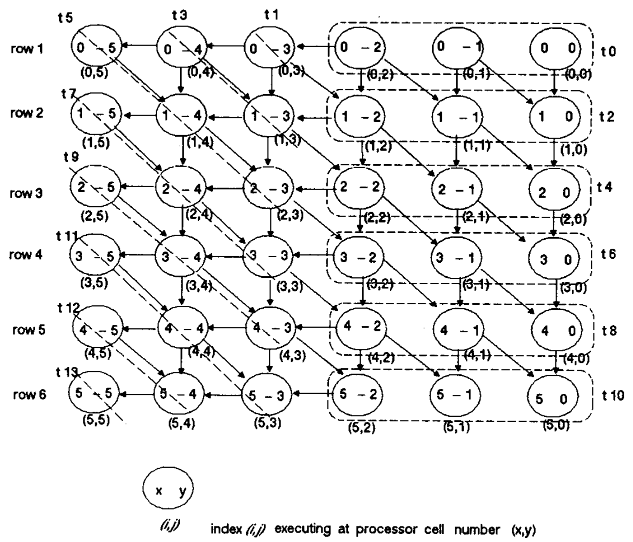


Fig. 6. The cells organization for implementing Example 2.1.

A possible solution to this problem is illustrated in Fig. 8. We select the partitioning transformation Π_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:

$$\begin{aligned}\Pi_p\bar{j} \bmod 3 = 0 \quad & \text{for } \bar{j} \in J_1^2, \quad \text{and} \\ \Pi_p\bar{j} \bmod 3 = 0 \quad & \text{for } \bar{j} \in J_2^2. \end{aligned} \tag{4.3}$$

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

$$\begin{aligned}\Pi^1\bar{j} &= (1,0)\bar{j} + C_{\bar{j}} \quad \text{for } \bar{j} \in J_1^2 \quad \text{and} \\ \Pi^2\bar{j} &= (1,1)\bar{j} + C_{\bar{j}} \quad \text{for } \bar{j} \in J_2^2\end{aligned}$$

which determine the sequence of computations inside the bands. Each band is swept by lines whose equation is $\Pi^1\bar{j} =$ constant for $\bar{j} \in J_1^2$ and $\Pi^2\bar{j} =$ constant for $\bar{j} \in J_2^2$. 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 Π^1 and Π^2. In Fig. 8, we mark the computation order of each such line by a positive integer aside it. The time hyperplanes Π^1 and Π^2 together with the partitioning hyperplanes Π_p which partitions the original For-loops algorithm and map it onto an array with only six processors. Each index $\bar{j}$ will be assigned to a band numbered $b = \lfloor(\Pi_p\bar{j})/3\rfloor$ and the processor P_i for executing index $\bar{j}$ is determined by the formula $i = \Pi_p\bar{j} \bmod 3$. 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 $\bar{j}_1 \in J_1^2 \cap J^e$ should be used by another index point $\bar{j}_2 \in J_2^2 \cap J^e$ in the near neighbor band B_2. Thus, these data should be transferred from processor numbered P_2 to the shift registers *FIFO* B'. Similarly, the data generated by index point $\bar{j}_2 \in J_2^2 \cap J^e$ should be transferred from processor P_2' to *FIFO* A' used by the index $\bar{j}_1 \in J_1^2 \cap J^e$ in the near neighbor band. Moreover, the data generated by index $\bar{j}_1 \in J_1^2$ in band B_1 may be used by another index $\bar{j}_1' \in J_1^2$ in band B_2. These data should be transferred from processor P_2 into shift registers *FIFO A* 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

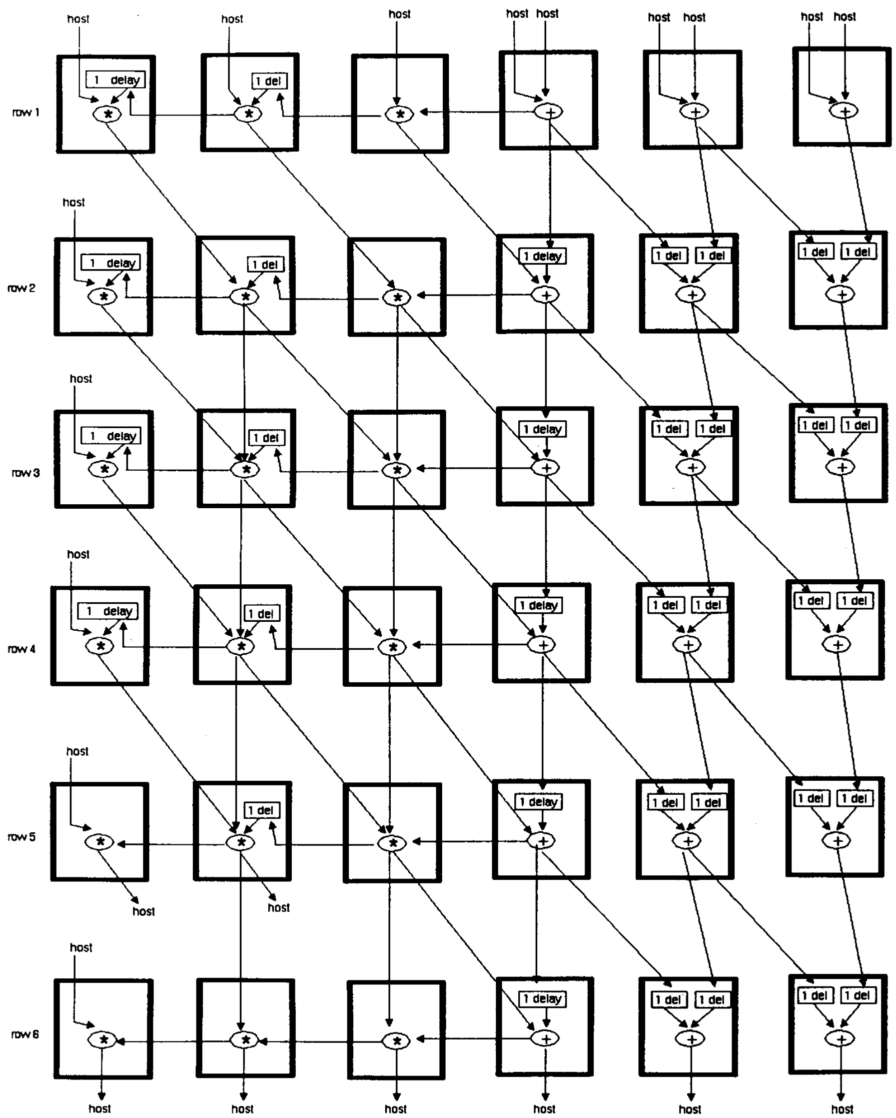


Fig. 7. The detail layout of each cell in Fig. 6.

satisfy the conditions

a) S can be solved from equation $SD = PK$.

b) The transformation matrix $T = \begin{bmatrix} \Pi^i \\ S \end{bmatrix}$ is nonsingular.

Step 4: For each valid transformation T, the partitioning hyperplanes Π_{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 $\bar{j}$ in band boundaries satisfies the equation

$$(\Pi_{pi}\bar{j}) \bmod m_i = 0 \quad \text{for } 1 \le i \le n-2 \text{ and}$$
$$(\Pi_{pi}\bar{j}) \bmod (m_i/p) = 0 \quad \text{for } i = n-1.$$

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 $\bar{j}$ is processed in a processor whose ith coordinate is $\Pi_{pi}\bar{j} \bmod m_i$.

Note that index point $\bar{j}$ is partitioned into band $B_{b_1 b_2 \cdots b_{n-1}}$ whose ith coordinate b_i is $\lfloor \Pi_{pi}\bar{j}/3 \rfloor$. 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.

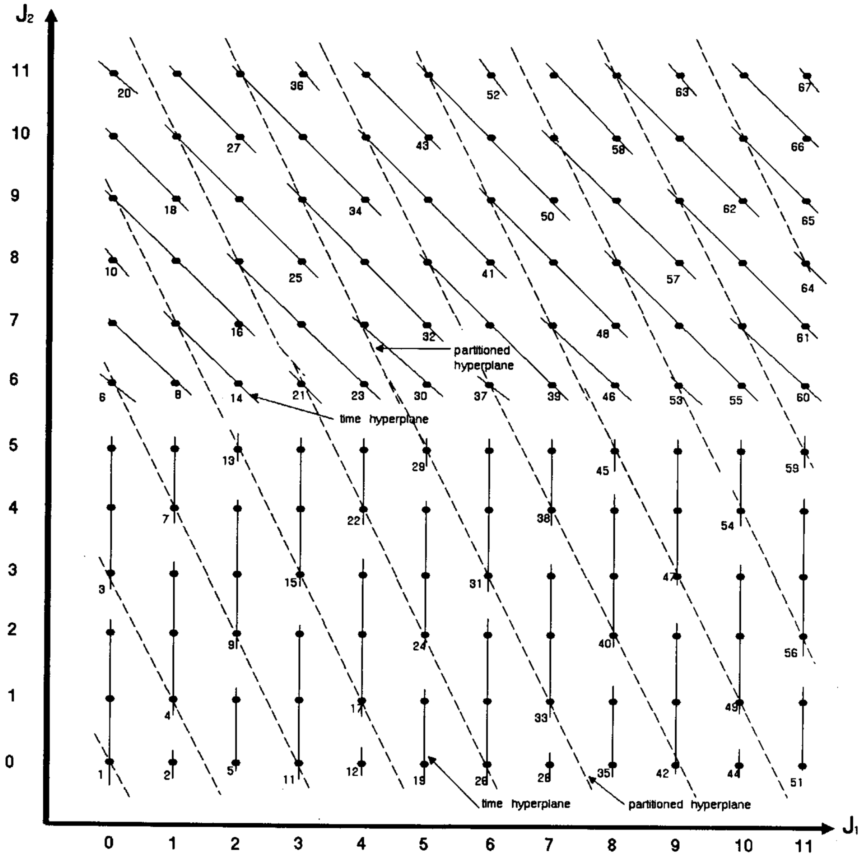


Fig. 8. Partitioning of Example 2.1 with larger index set.

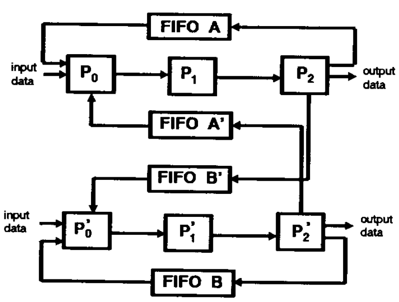


Fig. 9. Partitioned index set of Fig. 8 is mapped onto six processors with FIFO queues.

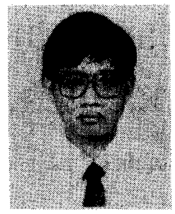
V. Conclusion

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_i^n, we select a nonlinear transformation function Π^i to map all the indexes onto a parallel execution form. All the indexes $\bar{j} \in J_i^n$ satisfy the equation $\Pi^i \bar{j} = C$ can be processed concurrently, where C is a constant. Furthermore, the structure of the $(n-1)$-dimensional systolic arrays for implementing the mapped 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 $\bar{j} \in J_i^n$ can be processed with an order $\Pi^i \bar{j}$ on the cell numbered $S\bar{j}$. 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.

References

[1] U. Banerjee, S.H. Chen, and D.J. Kuck, "Time and parallel processors bounds for Fortran-like loops," *IEEE Trans. Comput.*, vol. C-28, pp. 660–670, Sept. 1979.
[2] Z. Chen and C. C. Chang, "Iteration-level parallel execution of DO loops with a reduced set of dependence relations," *J. Parallel Distributed Comput.*, vol. 4, pp. 488–504, 1987.
[3] Guerra and Melhem, "Synthesis of systolic algorithm design," *Parallel Comput.*, vol. 12, pp. 195–207, 1989.
[4] R.W. Heuft and W.D. Little, "Improved time and parallel processors bounds for Fortran-like loops," *IEEE Trans. Comput.*, vol. C-31, no. 1, pp. 78–82, Jan. 1982.
[5] D.J. Kuck, Y. Muraoka, and S.C. Chen, "On the number of operations simultaneously executable in Fortran-like programs and their resulting speedup," *IEEE Trans. Comput.*, vol. C-21, Dec. 1972.
[6] L. Lamport, "The parallel execution of DO loops," *Commun. ACM*, pp. 83–93, Feb. 1974.

[7] P. Lee and Z. M. Kedam, "Synthesizing linear array algorithms from nested for loop algorithm," *IEEE Trans. Comput.*, vol. 37, pp. 1578–1598, Dec. 1988.
[8] ——, "Mapping nested loop algorithms into multi-dimensional systolic arrays," in *Proc. 1989 Int. Conf. Parallel Processing*, vol. III, 1989, pp. 206–210.
[9] T. C. Lin and D. I. Moldovan, "Tradeoffs in mapping algorithms to array processors," in *Proc. 1985 Int. Conf. Parallel Processing*, 1985, pp. 719–726.
[10] L. S. Lin, C. W. Ho, and J. P. Sheu, "On the parallelism of nested for-loops using index shift method," in *Proc. Int. Conf. Parallel Processing*, vol. 2, 1990, pp. 119–123.
[11] D. I. Moldovan and J. A. B. Fortes, "Partitioning and mapping algorithms into fixed size systolic arrays," *IEEE Trans. Comput.*, vol. C-35, pp. 1–12, Jan. 1986.
[12] M. Wolfe and Kuck, "Advanced loop interchanging," in *Proc. 1986 Int. Conf. Parallel Processing*, 1986, pp. 536–543.
[13] D. A. Padua, D. J. Kuck, and D. H. Lawrie, "High-speed multiprocessors and compilation techniques," *IEEE Trans. Comput.*, vol. C-29, pp. 763–776, Sept. 1980.
[14] D. A. Padua and J. Wolfe, "Advanced computer optimizations for supercomputers," *Commun. ACM*, vol. 29, pp. 1184–1201, Dec. 1986.
[15] J. K. Peir and Cytron, "Minimum distance: A method for partitioning recurrences for multiprocessors," *IEEE Trans. Comput.*, vol. 38, Aug. 1989.
[16] C. D. Polychronopoulos, D. J. Kuck, and D. A. Padua, "Utilizing multidimensional loop parallelism on large-scale parallel processor systems," *IEEE Trans. Comput.*, vol. 38, Sept. 1989.



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.