Partitioning and Mapping Nested Loops on Multiprocessor Systems

Jang-Ping Sheu, *Member, IEEE*, and Tsu-Huei Tai

Abstract—**Intensive scientific algorithms can usually be formulated as nested loops which are the main source of parallelism. When a nested loop is executed in parallel, the total execution time is composed of two parts—the computation time and the communication time. For a message-passing multiprocessor system, performance declines rapidly as the communication overhead is higher than the corresponding computation. In this paper, a method for parallel executing nested loops with constant loop-carried dependencies on message-passing multiprocessor systems to reduce the communication overhead is presented. First, we partition the nested loop into blocks which result in little communication without concern for the topology of machines. For a given linear time transformation found by the hyperplane method, the iterations of a nested loop are partitioned into blocks such that the communication among the blocks is reduced while the execution ordering defined by the time transformation is not perturbed. Then, the partitioned blocks generated by the partitioning method are mapped onto multiprocessor systems according to the specific properties of various machines. We propose a heuristic mapping algorithm for the hypercube machines.**

Index Terms— **Hypercubes, hyperplane method, message-passing multiprocessor systems, parallelizing compilers, systolic arrays, wavefront method.**

I. Introduction

In many numerical programs, nested loops are the most time-consuming parts and usually offer the most amount of parallelism. Lamport [9] proposed two parallel approaches to exploit this parallelism called the *hyperplane* method and the *coordinate* method according to different constraints upon the time transformation. In the past few years, several researchers studied the problem of transforming nested loops into parallel forms and mapping them onto special purpose architectures. Chen [3], [4], Lee and Kedem [11], Liu, Ho, and Sheu [13], Miranker and Winkler [14], and Moldovan and Fortes [15] synthesized systolic arrays based on the hyperplane method.

In message-passing multiprocessor systems, the communication overhead is still one order of magnitude higher than the corresponding computation [1]. Thus, an efficient parallel execution of algorithms or programs requires low ratio of communication overhead when computation is performed. Because of this requirement, various techniques have been developed for partitioning and mapping algorithms onto multiprocessor systems to reduce the communication overhead [2], [5], [7], [8], [17], [18], [20].

Some approaches, such as the *greatest common divisors* method [16], [18], the *minimum distance* method [18], Shang and Fortes' method [20], and D'Hollander's method [5], partition the iterations of nested loops into independent blocks so that there are no dependence relations between computations that belong to different blocks. In other words, these methods separate the iterations into several blocks so that there is no data communication or synchronization between them. For many important nested loop algorithms, such as matrix multiplication, discrete Fourier transform, convolution, transitive closure, and so forth, these index sets cannot be partitioned into independent blocks. Therefore, these algorithms will execute sequentially by their methods. The *grouping* method [8] partitions the algorithms into blocks with limited communication and gains more parallelism. For algorithms which cannot be partitioned into independent blocks, the grouping method will get better performance than the above partitioning methods.

In this paper, we concentrate on partitioning and mapping nested loops with constant loop-carried dependencies for execution on message-passing multiprocessor systems. In the partitioning phase, we divide the nested loop into blocks which reduce the interblock communication, without regard to the machine topology: First, the execution ordering of the iterations is defined by a given time function which is based on Lamport's hyperplane method [9]. Then, the iterations are partitioned into blocks so that the execution ordering is not disturbed and the amount of interblock communication is minimized. In the second, or mapping phase, we map the partitioned blocks to the topology of the target machine. These blocks are mapped onto a fixed size multiprocessor system in such a manner that the blocks which have to exchange data frequently are allocated to the same processor or neighboring processors. In this paper, we only consider the mapping for hypercubes.

The rest of this paper is organized as follows. Section II introduces the nested loop model, the hyperplane method, and the basic concept and terms used in the partitioning method. In Section III, an algorithm for partitioning the index set (iterations) of nested loops is described. In Section IV, we discuss the problems of mapping the partitioned blocks onto multiprocessor systems, and present a mapping method for hypercube computers. Moreover, the performance analysis of the partitioning and mapping algorithms by the matrix-vector multiplication is described. Finally, our conclusion is presented in Section V.

Manuscript received August 1, 1990; revised January 17, 1991. This work was supported by the National Science Council of the Republic of China under Grants NSC 80-0408-E-008-09 and NSC 81-0408-E-008-04.

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

IEEE Log Number 9102434.

II. Basic Concept and Definitions

Throughout this paper, the set of real numbers and the set of integers are denoted by $\mathbf{R}$ and $\mathbf{Z}$, respectively. The set of nonnegative real numbers and the set of nonnegative integers are denoted by $\mathbf{R}^+$ and $\mathbf{Z}^+$, respectively. The symbols $\mathbf{R}^n$ and $\mathbf{Z}^n$ represent the nth Cartesian powers of all real numbers and integers, respectively. Finally, we use boldface characters to represent the vertices in an n-dimensional graph; when the vertices are used in an equation, they represent the coordinates of these vertices. In this paper, we consider an n-nested loop of the form

```
for I1 = l1 to u1 by k1
  for I2 = l2 to u2 by k2
          ⋮
    for In = ln to un by kn
     Statement1;
     Statement2;
          ⋮
     Statement m;
     end
  end
end
```

where l_j and u_j are integer-valued linear expressions possibly involving $I_1, I_2, \cdots, I_{j-1}$ for $1 < j \leq n$. Without loss of generality, we assume that $l_j \leq u_j$ and $k_j = 1$ for all $1 \leq j \leq n$. We make additional assumptions about the statements. These statements contain no I/O instructions, no transfer of control to any statement outside the loop, and no subroutine or function calls which can modify data. These assumptions are first used by Lamport [9]. The index set $J^n = \{(i_1, i_2, \cdots, i_n) | l_j \leq i_j \leq u_j$, for $j = 1, \cdots, n\}$ is the set of loop index (or the iteration space).

Definition 1: Dependence vector: In an n-nested loop, suppose variable A is generated at iteration $\bar{i} = (i_1, i_2, \cdots, i_n)$ and used at iteration $\bar{j} = (j_1, j_2, \cdots, j_n)$, then the *dependence vector* $\bar{d}$ of variable A is a vector $(d_1, d_2, \cdots, d_n)^t \in \boldsymbol{Z}^n$ where $d_k = j_k - i_k$ for $1 \leq k \leq n$. □

Note that, the hyperplane method can only be used on nested loops with constant loop-carried dependencies [8], [15]. In other words, every index point of the index set has the same set of dependence vectors.

Example 1: Consider a 2-nested loop $L1$.

```
for i = 0 to 3
  for j = 0 to 3
S1 : A[i + 1, j + 1] := A[i + 1, j] + B[i, j];        (L1)
S2 : B[i + 1, j] := A[i, j] * 2 + C;
 end
end.
```

The index set $J^2 = \{(i, j) \mid 0 \leq i, j \leq 3\}$. The dependence vectors of variable A are $\bar{d}_1 = (0, 1)^t$ and $\bar{d}_2 = (1, 1)^t$. The dependence vector of variable B is $\bar{d}_3 = (1, 0)^t$. The set of dependence vectors $D = \{\bar{d}_1 \bar{d}_2 \bar{d}_3\}$. In the index set of a nested loop, two iterations can be executed concurrently if and only if they are independent of each other. From this point of view, Lamport proposed the hyperplane method [9]. The index set is traversed by many hyperplanes defined by a linear function Π if $\Pi \bar{d}_i > 0$ for any $\bar{d}_i \in D$. Because there is no dependence relation between the points lying on the same hyperplane, these points can be executed simultaneously. Given Π, the execution time of every index point can be determined. For loop $(L1)$, let the time function $\Pi = (1, 1)$. The hyperplanes $i + j =$ *constant* are shown in Fig. 1.

The hyperplane method is composed of two parts, the time transformation and space transformation [15]. The time transformation is used to determine the execution time of each index point and the space transformation assigns the index points onto processing elements. The hyperplane method is used widely in the synthesis of loops in systolic arrays [4], [11], [13]–[15]. Based on this method, the execution ordering of index points can be scheduled easily. Thus, the time transformation of the hyperplane method is used in our approach. However, the space transformation used for systolic arrays is not suitable for message-passing multiprocessor systems; we must do this in another way. We propose the following partitioning and mapping algorithms for multiprocessor systems: First, we partition the nested loops into larger blocks which result in little interblock communication, without regard to the topology of the target machine. Then, these blocks are mapped onto message-passing multiprocessor systems according to the specific properties of various machines. Some terms are defined in the following before the description of our algorithm.

We can represent an n-nested loop L as a directed graph Q in an n-dimensional space. Each vertex in Q represents an iteration (also called the index point) and has a coordinate $(i_1, i_2, \cdots, i_n)$ in the space if the corresponding iteration has a loop index $(i_1, i_2, \cdots, i_n)$. There is an arc from vertex $\boldsymbol{v}_i$ to vertex $\boldsymbol{v}_j$, if the iteration corresponding to $\boldsymbol{v}_j$ depends on the iteration corresponding to $\boldsymbol{v}_i$. Such a graph is called the *computational structure* [8] of the nested loop L. The computational structure of loop $(L1)$ is shown in Fig. 1.

Definition 2: Computational structure [8]: A *computational structure* Q of a nested loop L is a two-tuple, $Q = (V, D)$, where $V = \{\bar{i} \mid \bar{i} = (i_1, i_2, \cdots, i_n) \in J^n\}$ is the set of vertices in Q, and D is the set of dependence vectors. □

In a computational structure $Q = (V, D)$, a vertex $\boldsymbol{v}_j \in V$ is *dependent* on another vertex $\boldsymbol{v}_i \in V$ along a vector $\bar{d}$, if $\boldsymbol{v}_j - \boldsymbol{v}_i = \bar{d}$. Moreover, a vertex $\boldsymbol{v}_j$ is reachable from a vertex $\boldsymbol{v}_i$ via the set of vectors $\{\bar{d}_1, \cdots, \bar{d}_k\}$ if $\boldsymbol{v}_j = \boldsymbol{v}_i + a_1 \bar{d}_1 + \cdots + a_k \bar{d}_k$, where $a_l \in \boldsymbol{Z}^+$, $1 \leq l \leq k$, and $\boldsymbol{v}_i, \boldsymbol{v}_j \in V$. If vertex $\boldsymbol{v}_j$ is reachable from $\boldsymbol{v}_i$ via a path $\boldsymbol{v}_i$, $\boldsymbol{v}_{i+1}, \cdots, \boldsymbol{v}_{i+k}, \boldsymbol{v}_j$, then $\boldsymbol{v}_j$ should depend on the vertex $\boldsymbol{v}_{i+k}$, and $\boldsymbol{v}_{i+l}$ is dependent on $\boldsymbol{v}_{i+l-1}$, for $1 \leq l \leq k$. On the other hand, if two vertices $\boldsymbol{v}_i, \boldsymbol{v}_j \in V$ and there does not exist a path between $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$, then $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$ are *independent*, i.e., $\boldsymbol{v}_i$ is not reachable from the vertex $\boldsymbol{v}_j$ and vice versa. Note that any computational structure correspondent to a nested loop must be acyclic.

Definition 3: Projection vector: Let $\bar{p} = (p_1, \cdots, p_n)$ denote a *projection vector*. We define the *projection* of the vector $\bar{j} = (j_1, \cdots, j_n)$ with respect to the projection vector $\bar{p}$ as $\bar{j}^p = (j_1^p, \cdots, j_n^p)$, where $\bar{j}^p = \bar{j} - \frac{\bar{j} \cdot \bar{p}}{\bar{p} \cdot \bar{p}} \bar{p}$, i.e., $\bar{j}^p$ is the projection of vector $\bar{j}$ onto a plane that is perpendicular to $\bar{p}$. □

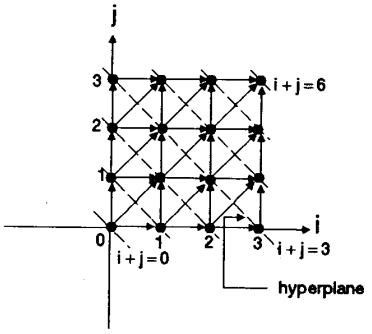


Fig. 1. The computational structure and hyperplanes of loop $(L1)$.

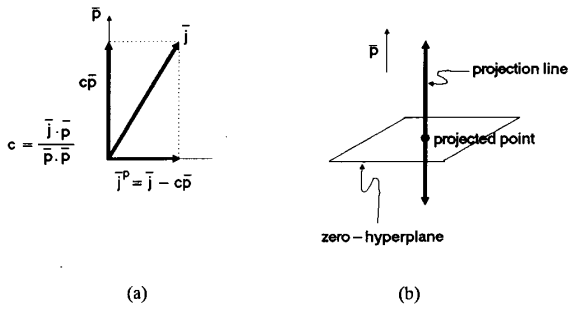


Fig. 2. (a) Projection of vector $\bar{j}$ with the projection vector $\bar{p}$. (b) Projected point and the corresponding projection line.

Fig. 2(a) is the graphical representation of the projection method. For a computational structure $Q = (V, D)$ of a nested loop L, the projection of a vertex $v_i \in V$ and a dependence vector $\bar{d}_i \in D$ with respect to the vector $\bar{p}$ are called the *projected point* and the *projected dependence vector*, respectively.

Definition 4: Projection line: The *projection line* which corresponds to a projected point v along the direction $\bar{p}$ is the line with the parametric equation $\bar{j} = v + t\bar{p}$, $t \in \mathbf{R}$. □

From the geometric view, $\bar{p}$ is the normal vector of the hyperplanes such that $\bar{p}\bar{x} = constant$, where $\bar{x} \in \mathbf{R}^n$. The hyperplane $\bar{p}\bar{x} = 0$ is called the *zero-hyperplane*. If we project a vector $\bar{j}$ with the projection vector $\bar{p}$ by the manner defined in Definition 3 then it is similar to projecting $\bar{j}$ onto the zero-hyperplane. As a result, the projection of $\bar{j}$ is lying on the zero-hyperplane. Fig. 2(b) shows the relation between a projected point and the corresponding projected line.

Definition 5: Projected structure: The *projected structure* $Q^p = (V^p, D^p)$ of an n-dimensional computational structure $Q = (V, D)$ with the projection vector $\bar{p}$ is a directed graph such that

a) The vertex set V^p is the set formed by the projected points.
b) D^p is the set of projected dependence vectors.
c) There is an arc from v_i^p to v_j^p, for v_i^p, $v_j^p \in V^p$, iff there exists a projected dependence vector $\bar{d}_k^p$, such that $v_j^p - v_i^p = \bar{d}_k^p$. □

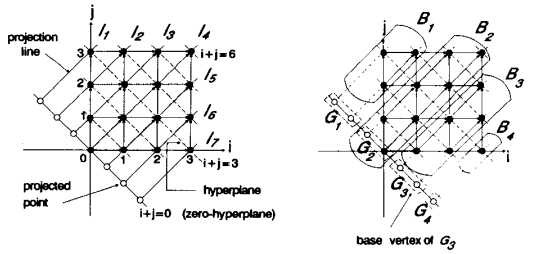
Suppose the dimension of a computational structure Q is n. Because Q is projected onto the zero-hyperplane whose dimension is $n-1$, the projected structure Q^p is an $n-1$ dimensional structure. From the definitions of the projection line and projected point, every projected point defines a unique projection line with the projection vector $\bar{p}$. Assume that there are two vertices v_i^p and v_j^p of a projected structure. Vertex v_i^p is said to be dependent on v_j^p , if there exists a projected dependence vector $\bar{d}_k^p \in D^p$ such that $v_i^p = v_j^p + \bar{d}_k^p$. This means that each index point on the corresponding projection line of v_i^p depends on a index point which lies on the corresponding projection line of v_j^p except the boundary index points of the index set J^n.

Consider the Example 1, we use Π = (1, 1) as the projection vector, i.e., $\bar{p} = \Pi$. Let $\bar{j}^p = (j_1^p, j_2^p)$ be the projected point corresponding to the index point $\bar{j} = (j_1, j_2)$, and $\bar{j}^p = \bar{j} - \frac{\bar{j}\cdot\Pi}{\Pi\cdot\Pi}\Pi = \bar{j} - \frac{j_1+j_2}{2}\Pi$, where $0 \le i, j \le 3$. We get seven projected points, $V^p = \{(-3/2, 3/2), (-1, 1), (-1/2, 1/2), (0, 0), (1/2, -1/2), (1, -1), (3/2, -3/2)\}$. Seven projection lines are defined with the projected points along the direction Π = (1, 1). The projected points and projection lines are depicted in Fig. 3(a) where the projected points except (0, 0) are illustrated by open circles. From Fig. 3(a), we know that the index points lying on the same projection line do not belong to the same hyperplane. Furthermore, the index points of neighboring lines do not lie on the same hyperplane, either; for instance, the index points belonging to the line l_4 lying on the hyperplanes $i+j=0$, $i+j=2$, $i+j=4$, and $i+j=6$; and the index points belonging to l_3 lying on the hyperplanes $i+j=1$, $i+j=3$, and $i+j=5$. Therefore, the index points located on these two lines can be grouped into a block and assigned to the same processor. Fig. 3(b) depicts the partitioning of loop $L1$. There are four groups and each consists of two projected points except the boundary group G_4. Let block B_i be the set of index points which are projected to the projected points belonging to group G_i. If we assign each block to one processor, the number of data dependencies between index points is 33, and only 12 of them require interprocessor (interblock) communication.

Definition 6: The *partitioning* $G_\Pi(Q)$ of a computational structure $Q = (V, D)$ with the projection vector (time function) $\Pi = (a_1, \cdots, a_n)$ is the partitioning of all vertices of Q into disjoint blocks $B_0, \cdots, B_{\alpha-1}$ where α is the total number of partitioned blocks in a nested loop. Suppose the disjoint groups of the projected structure $Q^p = (V^p, D^p)$ are denoted by $G_0, \cdots, G_{\alpha-1}$ which correspond to $B_0, \cdots, B_{\alpha-1}$, respectively. Then,

1) each group G_i, for $0 \le i \le \alpha - 1$, contains r projected points except the boundary groups, i.e., G_4 of Fig. 3(b);
2) for each group G_i, $0 \le i \le \alpha - 1$, there exists an ordering along a direction $\bar{d}^p \in D^p$ for all vertices in G_i, i.e., $(v_0^p, \cdots, v_{r-1}^p)$, such that $v_{j+1}^p - v_j^p = \bar{d}^p$ for $0 \le j \le r-2$;
3) and each block $B_i = \bigcup_{v_k^p \in G_i} \{\bar{j} \in J^n \mid \bar{j} = v_k^p + t\Pi, t \in \mathbf{R}\}$, for $0 \le i \le \alpha - 1$. □

Each subset B_i is called a *partitioned block* of $G_\Pi(Q)$. The first vertex v_0^p of G_i in the ordering is called the *base vertex*. A



(a) (b)

Fig. 3. The projected structure of loop ($L1$) with Π = (1, 1). (a) Projected points, projection line, and hyperplanes. (b) Groups and the corresponding blocks of loop ($L1$).

group G_i is dependent on another group G_i along $\bar{d}^p$ if there is a vertex $v_k^p \in G_i$ which depends on a vertex $u_l^p \in G_j$ along $\bar{d}^p$. In Fig. 3(b), the projected dependence vectors $\bar{d}_1^p$, $\bar{d}_2^p$, and $\bar{d}_3^p$ are $(-1/2, 1/2)^t$, $(0,0)^t$, and $(1/2, -1/2)^t$, respectively. Along the direction of $(-1/2, 1/2)^t$, the base vertex of G_3 is (1, −1).

III. Partitioning of Index Set In Nested Loops

The *partitioning* problem considered in this paper can be stated as follows. Given an n-nested loop L, the problem is to determine the partitioned blocks of L such that 1) the amount of interblock communication is as small as possible and 2) the execution ordering defined by the time transformation function Π is not disturbed. The partitioning algorithm is composed of two phases: the projection phase and the grouping phase. First, in the projection phase, the vertex set of Q (or the index set of L) is projected onto the zero-hyperplane, $\Pi\bar{x}$ = 0. Every projected point represents some iterations whose corresponding vertices in Q are projected to this point. In other words, these iterations lie on the corresponding projection line of the projected point. Because this projection line is perpendicular to the hyperplane, these computations of the index points lying on the projection line will not be executed at the same time. Therefore, the iterations located on any one projection line can be assigned to the same processor without increasing the total number of execution steps.

Next, we group the projected points into a group whose corresponding projection lines do not interfere with each other. If there are more projected points within a group, then there are fewer data exchanges among the corresponding partitioned blocks. We want to group as many projected points into one group as we can. As a consequence, a good grouping scheme will make the size of groups and the corresponding blocks as large as possible.

Definition 7: Grouping and auxiliary grouping vectors: The *grouping vector* is a vector used for grouping the projected points into groups. All the projected points are grouped along the grouping vector. The *auxiliary grouping vectors* are a set of vectors which determine the base vertex of every group in the projected structure. □

Suppose D^p is the set of projected dependence vectors that corresponds to the dependence vectors D of a nested loop L. The matrix formed by the projected dependence vectors is denoted by $mat(D^p)$; the rank of a matrix A is represented by $rank(A)$. From the definition of a vector space, n linearly independent vectors form a basis of an n-dimensional vector space. That is, an n-dimensional vector space can be generated using exactly n linearly independent vectors. Since the dimension of the projected structure is $n-1$, there are at most $n-1$ vectors that can be used as grouping and auxiliary grouping vectors in the grouping phase. If we use more than $n-1$ vectors, then some vectors can be expressed as a linear combination of the other $n-1$ linearly independent vectors. Thus, it will cause a conflict when we determine the base vertices of groups. Now, we will explain how to choose the grouping and auxiliary grouping vectors.

When two index points are interdependent, there are data transfers occurring between them. If these index points belong to the same partitioned block, then the number of interblock data exchanges will be reduced. Since the dependence relation between two projected points represents the dependence relations of the index points that are projected to these projected points, we will gather the projected points which have dependence relations among them. We want to choose the grouping and auxiliary grouping vectors from D^p such that the communication among partitioned blocks is minimized. Let r_i be the smallest positive integer such that $r_i\bar{d}_i^p \in \mathbb{Z}^n$. Then, r_i projected points can be grouped along the direction of $\bar{d}_i^p$ as a group and all the index points which are projected into this group are executed at different execution steps. This means that these index points can be mapped to the same processor. We will prove this in Lemma 1 of the Appendix. Suppose $\bar{d}_l^p$ is the projected dependence vector whose r_l is the largest, i.e., $r_l = \max_{\bar{d}_i^p \in D^p}\{r_i\}$. If there is more than one projected dependence vector whose r_i is equal to the largest value, then we choose one from these vectors arbitrarily. The vector $\bar{d}_l^p$ is called the grouping vector. The size of each group is $r = r_l$.

The selection of the auxiliary grouping vectors is discussed in the following. From the definition of rank, the rank of a matrix represents the maximum number of linearly independent columns (or rows) of this matrix [10]. Since $rank(mat(D^p)) = \beta$, it is possible to choose β linearly independent vectors from D^p. We select $\beta - 1$ vectors from $D^p - \{\bar{d}_l^p\}$ as the auxiliary grouping vectors such that these $\beta-1$ vectors and the grouping vector are linearly independent. Since all the data exchanges are caused by the data dependence relations, we can reduce the amount of communications among the groups by choosing the auxiliary grouping vectors from $D^p - \{\bar{d}_l^p\}$. We will show these facts in Lemma 2 and Lemma 3 of the Appendix.

Definition 8: Forward and backward neighboring groups: Let the set of auxiliary grouping vectors be denoted by Ψ. Suppose u_0^p, v_0^p, and w_0^p are the base vertices of groups G_i, G_j, and G_k. If $u_0^p = v_0^p - r\bar{d}_l^p$ and $u_0^p = w_0^p + r\bar{d}_l^p$, then G_j and G_k are the *forward and backward neighboring groups* of G_i along the grouping vector $\bar{d}_l^p$, respectively. If $u_0^p = v_0^p - \bar{d}_j^p$ and $u_0^p = w_0^p + \bar{d}_j^p$, where $\bar{d}_j^p \in \Psi$, then G_j and G_k are the forward and backward neighboring groups of G_i along the auxiliary grouping vector $\bar{d}_j^p$, respectively. □

After the grouping and auxiliary grouping vectors have been determined, we will group the projected points into groups.

The idea is similar to the *region growing* method [6] in image processing that is used to find all groups of the projected structure. We group some projected points into a group and from this determine the other groups along the directions of the grouping and auxiliary grouping vectors. First, the projected points are grouped along the direction of the grouping vector. The projected structure is split into many parallel lines along the direction of $\bar{d}_l^p$. We select one line from these parallel lines arbitrarily, and choose the projected point v_0^p lying on the selected line to be the base vertex of the first group G_0. Then, we group r projected points along the direction of the grouping vector from v_0^p and generate the first group G_0. The following step is to determine the base vertices of the other groups and the members of them. We use the first group as the initial seed group. From the seed group, we get the backward and forward neighboring groups along the directions of the grouping and each of the auxiliary grouping vectors. After this, the neighboring groups found above are used as the seed groups and from these groups find all neighboring groups which have not been determined in the previous step. Using the groups found in the last step as the seed groups, we determine the other groups in the way described above repeatedly until there exists no neighboring group of the seed groups.

If there are some projected points that have not been gathered into groups, we select an ungrouped line, and group r projected points on the line into a group. Using the group as the initial seed group, we perform the grouping procedure described above until all the projected points have been gathered into groups. After all groups of the projected structure have been determined, the corresponding partitioned block of $G_i = \{v_0^p, \cdots, v_{r-1}^p\}$ is the set

$$B_i = \bigcup_{v_k^p \in G_i} \{\bar{j} \in J^n \mid \bar{j} = v_k^p + t\Pi, t \in \mathbf{R}\}.$$

If we group the projected points by the manner described above, then Lemma 2 proves that a group only depends on another group along the direction of the grouping vector and each of the auxiliary grouping vectors. In addition, Lemma 3 proves that a group depends on at most two groups along the direction of vectors $\in D^p - (\Psi \cup \{\bar{d}_l^p\})$. The formal partitioning algorithm is described as follows:

Algorithm 1: (Partitioning a nested loop)

Input: A *computational structure* $Q = (V, D)$ of an n-nested loop L and a time transformation function $\Pi = (a_1, \cdots, a_n)$ found by the hyperplane method as the projection vector.

Output: A set of partitioned blocks $G_\Pi(Q) = \{B_0, \cdots, B_{\alpha-1}\}$ of the computational structure Q.

Projection Phase: /* Phase 1 */

Project the vertex set V, and the dependence vectors D, onto the zero-hyperplane with respect to Π. After projection, we get two sets—the set of projected points V^p and the set of projected dependence vectors D^p. These two sets constitute the projected structure $Q^p = (V^p, D^p)$ of Q.

Grouping Phase: /* Phase 2 */

/* In Step 1 and Step 2, we select vectors as grouping and auxiliary grouping vectors, where $\beta = rank(mat(D^p))$. */

Step 1: Let r_i be the smallest positive integer such that $r_i \bar{d}_i^p \in \mathbf{Z}^n$, for $\bar{d}_i^p \in D^p$. Then, the size of every group is $r = \max_{\bar{d}_i^p \in D^p}\{r_i\}$. Assume that $\bar{d}_l^p \in D^p$ is the vector whose $r_l = r$. Then we choose $\bar{d}_l^p$ as the grouping vector. If there are many vectors $\in D^p$ whose r_i values are equal to r, then we choose one arbitrarily.

Step 2: Choose the other $\beta - 1$ projected dependence vectors as the auxiliary grouping vectors from $D^p - \{\bar{d}_l^p\}$ such that these $\beta - 1$ vectors and $\bar{d}_l^p$ are linearly independent. Let the set of the auxiliary grouping vectors be denoted by $\Psi = \{\bar{d}_1^p, \cdots, \bar{d}_{\beta-1}^p\}$.

/* After Step 2, the projected structure can be split into many parallel lines along the direction of $\bar{d}_l^p$. */

Step 3: Select a line arbitrarily; choose a projected point which is lying on this line as the base vertex of a group. Then, starting from the base vertex, group r projected points along $\bar{d}_l^p$ into a group. Let the group be the initial seed group.

Step 4: From the seed groups, find all the backward and forward neighboring groups. Then, the groups found above are used as the seed groups. Repeat this step until there exists no neighboring group of the seed groups.

Step 5: After Step 4, if there are some lines whose projected points are not grouped, go to Step 3.

Step 6: For every group of the projected structure $G_i = \{v_0^p, \cdots, v_{r-1}^p\}$, the corresponding partitioned block $B_i \in G_\Pi(Q)$ is the set

$$\bigcup_{v_k^p \in G_i} \{\bar{j} \in J^n \mid \bar{j} = v_k^p + t\Pi, t \in \mathbf{R}\}.$$

□

Algorithm 1 generates a partitioning so that every partitioned block contains as many index points as possible under the time function Π. Hence, the interblock communication is as small as possible and the execution ordering defined by Π is not disturbed.

Theorem 1: The partitioned blocks generated by Algorithm 1 obey the scheduling defined by the time function Π.

Proof: Algorithm 1 groups the projected points by the manner of Lemma 1. From Lemma 1, the iterations that correspond to a group will not execute simultaneously. Hence, the partitioned blocks generated by Algorithm 1 do not disturb the scheduling defined by the time function Π. □

Theorem 2: Let there be m dependence vectors and one grouping vector and $\beta - 1$ auxiliary grouping vectors, where $\beta = rank(mat(D^p))$. Then, a group has to send data to at most $2m - \beta$ groups.

Proof: From Lemma 2, a group will send data to another group along each direction of $\bar{d}_i^p \in \Psi \cup \{\bar{d}_l^p\}$. From Lemma 3, the group will send data to at most two groups along each direction of $\bar{d}_i^p \in D^p - (\Psi \cup \{\bar{d}_l^p\})$. Then, there are at most β and $2(m - \beta)$ groups that depend on the group along the direction $\bar{d}_i^p \in \Psi \cup \{\bar{d}_l^p\}$ and $\bar{d}_i^p \in D^p - (\Psi \cup \{\bar{d}_l^p\})$,

respectively. Thus, the number of groups depends on a group is at most $\beta + 2(m - \beta) = 2m - \beta$. □

Example 2: Consider the matrix multiplication algorithm:

```
for i = 0 to 3
  for j = 0 to 3
    for k = 0 to 3
      C[i,j] := C[i,j] + A[i,k] * B[k,j];          (L2)
    end
  end
end.
```

This program can be rewritten into the following equivalent form.

```
for i = 0 to 3
  for j = 0 to 3
    for k = 0 to 3
    A^(i,j,k)[i,k] := A^(i,j-1,k)[i,k];
    B^(i,j,k)[k,j] := B^(i-1,j,k)[k,j];
    C^(i,j,k)[i,j] := C^(i,j,k-1)[i,j];          (L3)
    C^(i,j,k)[i,j] := C^(i,j,k)[i,j]+A^(i,j,k)[i,k]*B^(i,j,k)[k,j];
    end
  end
end.
```

The dependence matrix is

$$\begin{matrix}\begin{bmatrix}0 & 1 & 0\\ 1 & 0 & 0\\ 0 & 0 & 1\end{bmatrix}.\\ \bar{d}_A\ \bar{d}_B\ \bar{d}_C\end{matrix}$$

The computational structure Q is shown in Fig. 4. Let the time function $\Pi = (1, 1, 1)$, then the zero-hyperplane is $i+j+k = 0$.

Projection Phase: Q is projected onto the zero-hyperplane. The projected structure $Q^p = (V^p, D^p)$ is depicted in Fig. 5. There are 37 projected points shown by solid circles; the points shown by the open circles are the projected points generated by the projection when the upper (or lower) bounds of the loop indexes are larger. The projected dependence vectors are $D^p = \{\bar{d}_A^p = (-1/3, 2/3, -1/3)^t,\ \bar{d}_B^p = (2/3, -1/3, -1/3)^t,\ \bar{d}_C^p = (-1/3, -1/3, 2/3)^t\}$.

Grouping Phase:

/* Since rank($\begin{bmatrix}-1/3 & -1/3 & 2/3\\ 2/3 & -1/3 & -1/3\\ -1/3 & 2/3 & -1/3\end{bmatrix}$) = 2, we select two projected dependence vectors such that one is the grouping vector and the other one is the auxiliary grouping vector. */

Step 1: The size of a block is $r = \max_{\bar{d}_i^p \in D^p}\{r_i\} = 3$. Then, there are three projected points in one group except the boundary groups. We select the grouping vector arbitrarily. Let $\bar{d}_l^p = \bar{d}_A^p = (-1/3, 2/3, -1/3)^t$.

Step 2: The auxiliary grouping vector is $\bar{d}_C^p = (-1/3, -1/3, 2/3)^t$.

/* After Step 2, the projected structure is split by $\bar{d}_A^p$ into seven parallel lines, $l_0, \cdots, l_6$. */

Step 3: Select the projected points $(-1, -1, 2)$, $(-4/3, -1/3, 5/3)$, $(-5/3, 1/3, 4/3)$ from l_0 to form the group G_1 and let $(-1, -1, 2)$ be the base vertex v_0^p as shown in Fig. 5. The vertex v_2^p in this group is $v_2^p = (-1, -1, 2) + 2\bar{d}_A^p = (-5/3, 1/3, 4/3)$.

Step 4: Determine the base vertices of other groups along $\bar{d}_C^p = (-1/3, -1/3, 2/3)$. The resulting groups are depicted in Fig. 6, every group is shown by a dashed box.

Step 5: In this example, all points can be grouped, and terminate the step.

/* Consider the group G_{10} as shown in Fig. 6; there are two groups G_7 and G_{11} depending on G_{10} along the direction $\bar{d}_A^p = (-1/3, 2/3, -1/3)$ and $\bar{d}_C^p = (-1/3, -1/3, 2/3)$, respectively. In addition, it also sends data to G_{12} and G_{13}

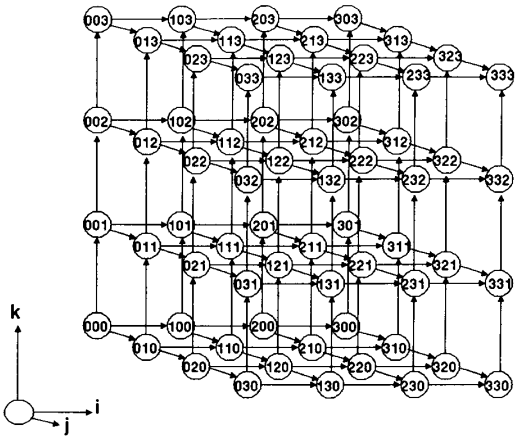
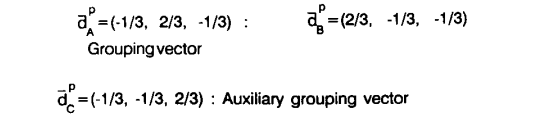


Fig. 4. The computational structure of matrix multiplication.



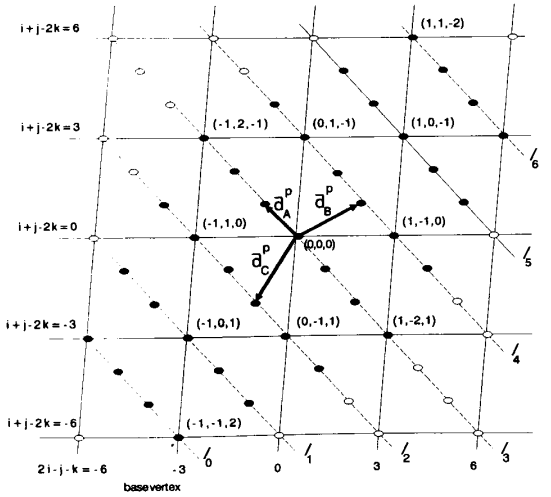


Fig. 5. The projected structure of Fig. 4.

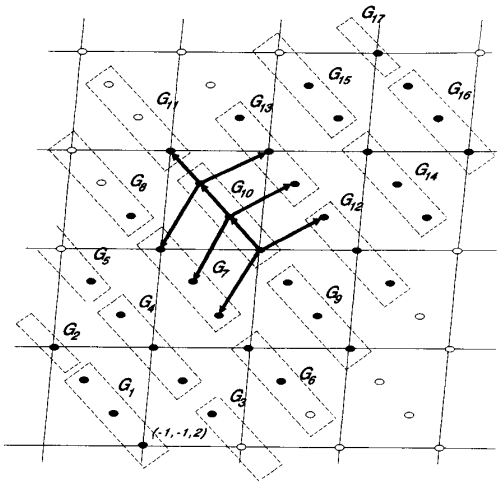


Fig. 6. Grouping the projected points of Fig. 5.

along $\bar{d}_B^p = (2/3, -1/3, -1/3)$ which is not a grouping or auxiliary grouping vector. Hence, there are $2 \times 3 - 2 = 4$ groups that depend on the group G_{10}. */

Step 6: The procedure partitions the nested loop into 17 partitioned groups. We use a node to represent a partitioned group, and if G_i will send data to G_j then there is a edge from G_i to G_j. The resulting graph is illustrated in Fig. 7. □

IV. Mapping the Partitioned Blocks of Nested Loops onto Hypercubes

When an algorithm is executed on a message-passing multiprocessor system, the overhead due to interprocessor communication/synchronization, and idle processors due to contention for shared hardware resources can lead to poor overall performance [19]. Therefore, we have to map the parallel component of the algorithm on the processors in such a way that minimizes the time required to perform necessary interprocessor communication and the amount of processor idle time. In Section III, a nested loop is partitioned into blocks without concern for the machine topology. The next step is to map the partitioned blocks of a nested loop onto a specific target machine. We choose the hypercube topology as the target machine because of the recent interest in this configuration. Hypercubes are loosely coupled parallel processors based on the binary n-cube interconnection network. An n-dimensional hypercube computer consists of $N = 2^n$ identical processors, each provided with its own local memory, and directly connected to n neighbors.

In general, the problem size is much larger than the machine size. Without loss of generality, we assume that the number of partitioned blocks of a nested loop is larger than the number of processors in the target machine. Therefore, we first divide these blocks into clusters which are suitable for mapping onto the target machine. After this processing, the clusters are assigned to processors by some mapping algorithm.

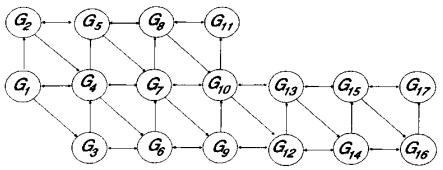


Fig. 7. Graphical representation of Fig. 6 with communication links.

First, we divide the total partitioned blocks into two clusters, of equal size, and divide every resulting cluster into two subclusters until there are N clusters generated. Then these N clusters can be mapped to an n-cube. Since the number of clusters increases by a factor of two after each dividing, we divide the partitioned blocks n times to generate N clusters. Our mapping approach is based on the concept illustrated above. Moreover, the partitioned blocks are first modeled by the Task Interaction Graph (TIG) model [19]. In the TIG model, the vertices represent blocks of the partitioning, and the edges of the TIG represent communication requirements between blocks.

Assume that there is a partitioning of a nested loop and it will be mapped onto an n-dimensional hypercube. The mapping algorithm proceeds in two phases:

1) Cluster formation: The TIG is partitioned into as many clusters as the number of processors. We start with the entire partitioning as a single cluster, and successively divide each cluster into two equal size of blocks n times.
2) Cluster allocation: The clusters generated in the first phase are numbered using the Gray code scheme. After this numbering, a cluster is allocated to the processor whose binary representation is identical to the binary number of the cluster. The mapping algorithm is described as follows.

Algorithm 2: (Hypercube Mapping)

Phase I: Cluster Formation.

/* Let $\Omega = \{\bar{g}_0, \cdots, \bar{g}_{\beta-1}\}$ be the set of grouping and auxiliary grouping vectors defined in Section III */

Let $j = 0$;

Do the following operations n times:

$i = j \bmod \beta$;

Select a vector $\bar{g}_i$ from Ω;

Partition every cluster generated in the last partitioning along the direction $\bar{g}_i$ into two equal size of subclusters;

$j = j + 1$;

/* After Phase I, the TIG is divided into $N = 2^n$ clusters. */

Phase II: Cluster Allocation.

/* Because of N clusters generated in Phase I, it is necessary to use n-bit binary codes to number these clusters. */

Step 1: Suppose the TIG have been divided along m different directions in Phase I. If $\beta \geq n$, i.e., there are n vectors chosen from Ω in Phase I, then $m = n$ else $m = \beta$. Assume the TIG is divided p_i times along the $\bar{g}_i$ direction. Because there are 2^n clusters generated in Phase I and 2^{p_i} partitions in the $\bar{g}_i$ direction, thus $n = p_1 + p_2 + \cdots + p_m$, i.e., $2^n = 2^{p_1} \cdot 2^{p_2} \cdots 2^{p_m}$. We use p_i-bit Gray code for

the $\overline{g}_i$ direction. Then every cluster has a unique n-bit binary representation which is obtained by concatenating the ith coordinate, for $1 \leq i \leq m$.

Step 2: Every cluster is allocated to the processor whose binary number is the same as that of the cluster. □

Algorithm 2 divides the partitioned blocks of a nested loop into clusters such that the neighboring blocks will be allocated to the same cluster. From the discussion of our partitioning method, most of the communication happens at neighboring blocks. Hence, the amount of data transmission of the clusters is reduced.

Example 3: Suppose a 4 × 4 mesh-like TIG as shown in Fig. 8(a) will be mapped onto a three-dimensional hypercube as shown in Fig. 8(b). Since there are $8 = 2^3$ processors, the TIG has to be divided 3 times. Let $\Psi = \{\overline{x}, \overline{y}\}$ be the grouping and auxiliary grouping vectors. Assume that the TIG is divided twice along $\overline{y}$ and once along $\overline{x}$. Choosing a 2-bit Gray code for the $\overline{y}$ direction and 1-bit Gray code for the $\overline{x}$ direction, we number each cluster where the binary representation is obtained by concatenating its binary $\overline{x}$ and $\overline{y}$ coordinates. Every cluster is assigned to the node whose number is identical to the binary representation of the cluster. For example, the blocks B_1 and B_2 are grouped within cluster C_0 whose binary representation is 000. Then the blocks in cluster C_0 are allocated to processor 000. Such a mapping is illustrated in Fig. 8(b). □

In the following, we use an example, the matrix-vector multiplication, to discuss the performance of our partitioning algorithm and mapping algorithm for hypercubes.

The matrix-vector multiplication is described as follows:

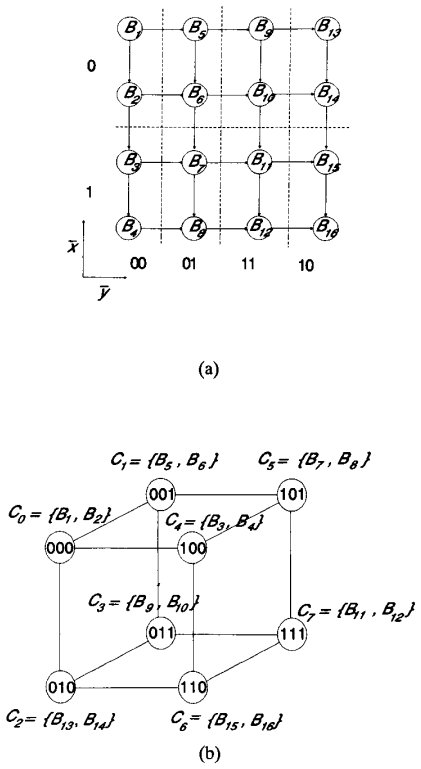
```
for i := 1 to M
  for j := 1 to M
    y[i] := y[i] + A[i,j] * x[j];                    (L4)
  end
end.
```

The above loop can be rewritten into the following equivalent form:

```
for i := 1 to M
  for j := 1 to M
    x^(i,j)[j] := x^(i-1,j)[j];
    y^(i,j)[i] := y^(i,j-1)[i] + A^(i,j)[i,j] * x^(i,j)[j];   (L5)
  end
end.
```

The set of dependence vectors is $D = \{\overline{d}_x = (1,0)^t, \overline{d}_y = (0,1)^t\}$ and the computational structure is shown in Fig. 9. We choose Π = (1, 1) as the time transformation and the nested loop will be executed on a hypercube with $N = 2^n$ transformation and the nested loop will be executed on a hypercube with $N = 2^n$ processors. In the machine, the time to perform a floating-point multiply or add is t_{calc}, and the time to communicate including two parts—t_{start}, which is the startup time for communication, and t_{comm} which is the time to transmit a single real word. Therefore, the time to transmit k real words between two processors is $t_{\text{start}} + kt_{\text{comm}}$.

In our partitioning and mapping algorithms, the two-dimensional computational structure is projected with the projection vector Π = (1, 1), and a one-dimensional projected structure is formed by the projected points and the projected dependence vectors. The set of projected dependence vectors is $D^p = \{\overline{d}_x^p = (1/2, -1/2)^t, (\overline{d}_y^p) = (-1/2, 1/2)^t\}$ and there are $2M - 1$ projected points. Suppose we choose the vector $\overline{d}_x^p$ as the grouping vector, then there are M groups and every one has two projected points except the one at boundary. That is each corresponding block contains two projection lines. Because these M blocks have to be mapped onto N processors, every processor consists of M/N blocks. The largest block is the one that contains the main diagonals of the computational structure. Therefore, the maximum number of index points assigned to a processor is $W = \sum_{i=1}^{M} i$, where $l = \lfloor \frac{N-2}{N} M \rfloor + 1$, and the computation time needed for the processor is $2Wt_{\text{calc}}$. Moreover, the communication time needed is $(2M-2)$ $(t_{\text{start}} + t_{\text{comm}})$. Hence, the maximum



(a)

(b)

Fig. 8. Mapping of a TIG into a three-dimensional hypercube.

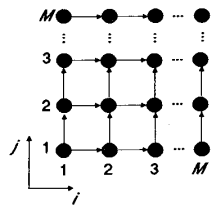


Fig. 9. The computational structure of loop $(L5)$.

execution time of the processors is

$$T_{\text{exec}}(N) = 2Wt_{\text{calc}} + (2M-2)(t_{\text{start}} + t_{\text{comm}}).$$

Table I illustrates the maximum execution time for parallel executing a nested loop, where the problem size M =1024. From Table I, we know that the communication time of our method is invariant when the machine size becomes larger. The reason is that the communication time is determined by the largest amount of interblock communication that occurred between two processors. The main diagonal of the computational structure is always the boundary points of a processor, that is, these index points must communicate with other adjacency index points. Therefore, the amount of interblock communication needed by the index points on main diagonal is $2(M-1)$ which is independent of the machine size. In our method, the ratio of communication time to computation time declines rapidly as the grain size grows. Thus, our method is suitable for medium- to coarse-grain computation.

TABLE I
THE EXECUTION TIME $T_{\text{exec}}(N)$ WITH $M = 1024$

$N = 1$	$2097152t_{\text{calc}}$
$N = 4$	$786944t_{\text{calc}}$+2046($t_{\text{comm}} + t_{\text{start}}$)
$N = 16$	$245888t_{\text{calc}}$+2046($t_{\text{comm}} + t_{\text{start}}$)
$N = 64$	$64544t_{\text{calc}}$+2046($t_{\text{comm}} + t_{\text{start}}$)
$N = 256$	$16328t_{\text{calc}}$+2046($t_{\text{comm}} + t_{\text{start}}$)
$N = 1024$	$4094t_{\text{calc}}$+2046($t_{\text{comm}} + t_{\text{start}}$)

V. CONCLUSION

In this paper, we consider the problem of executing nested loops in parallel on message-passing multiprocessor systems. We partition the nested loop into blocks which result in little interblock communication and then map these blocks efficiently onto the processors.

In the partitioning method, a given time transformation is used for deciding the execution ordering of all the index points in the index set. Using as the projection vector, we project the index points and dependence vectors along the direction which is perpendicular to Π. The projected points are grouped along the direction of projected dependence vectors into groups in the manner such that all index points that are projected to the projected points within one group will not be executed at the same time. The group size is as large as possible in order to reduce the communication overhead. Finally, we get every partitioned block by finding all the index points that are projected to the corresponding group.

After partitioning, the blocks of the nested loop are divided into clusters suitable for the topology and size of the target machine. Then, we can use techniques developed for the task allocation on multiprocessor systems to map the clusters onto machines. In this paper, we propose a heuristic mapping algorithm for hypercube machines by successively dividing the partitioning of a nested loop and numbering the resulting clusters using the Gray code sequence. Then every cluster is allocated to the processor whose binary number is the same as that of the cluster.

APPENDIX

In this Appendix, we prove Lemma 1, Lemma 2, and Lemma 3.

Lemma 1: Let $r \in \mathbb{Z}^+$ be the smallest positive integer such that $r\bar{d}_l^p \in \mathbb{Z}^n$. Let the projected points $v_0^p, \cdots, v_{r-1}^p$ be grouped into the same group where $v_{i+1}^p = v_i^p + \bar{d}_l^p$, for $0 \le i \le r-2$. Then, all the index points $\in J^n$ projected to v_i^p, for $0 \le i \le r-1$, will be executed at different time.

Proof: From the hyperplane method, two iterations can be executed at the same time if they lie on the same hyperplane. That is, the index points of J^n which lie on the same hyperplane, $\Pi\bar{x} = c$, where $\bar{x} \in \mathbb{R}^n$ can be executed simultaneously. Consider the projected points v_i^p and v_j^p chosen from $\{v_0^p, \cdots, v_{r-1}^p\}$, where $i < j$, i.e., $v_j^p = v_i^p + (j-i)\bar{d}_l^p$. The corresponding projection lines of v_i^p and v_j^p are $\bar{y} = v_i^p + t\Pi$ and $\bar{z} = v_j^p + s\Pi$, where t and $s \in \mathbb{R}$, respectively. Assume that two index points $\bar{i} = v_i^p + a\Pi$ and $\bar{j} = v_j^p + b\Pi$ belong to the same hyperplane $\Pi\bar{x} = c$, where $a, b \in \mathbb{R}$, i.e., $\Pi\bar{i} = \Pi\bar{j} = c$. We will prove that $\bar{i}$ and $\bar{j}$ do not belong to the index set J^n simultaneously. Since v_i^p and v_j^p lie on the zero-hyperplane, $\Pi\bar{x} = 0$, then

$$\Pi\bar{i} = \Pi(v_i^p + a\Pi) = \Pi v_i^p + a\Pi^2 = a\Pi^2 = c,$$
$$\text{and } \Pi\bar{j} = \Pi(v_j^p + b\Pi) = \Pi v_j^p + b\Pi^2 = b\Pi^2 = c.$$

It implies that $a = b$. Assume $\bar{i} = v_i^p + a\Pi \in \mathbb{Z}^n$. Then,

$$\bar{j} = v_j^p + a\Pi = v_i^p + (j-i)\bar{d}_l^p + a\Pi = v_i^p + a\Pi + (j-i)\bar{d}_l^p.$$

Because r is the smallest positive integer that makes $r\bar{d}_l^p \in \mathbb{Z}^n$ and $(j-i) < r$, the term $(j-i)\bar{d}_l^p \notin \mathbb{Z}^n$ and $v_i^p + a\Pi \in \mathbb{Z}^n$. Then, $j \notin \mathbb{Z}^n$, it means that $\bar{j} \notin J^n$. Hence, the index points $\in J^n$ that are projected to v_i^p and v_j^p do not belong to the same hyperplane, i.e., they will not be executed at the same time. □

Lemma 2: A group only depends on one group along the direction of the grouping vector and each of the auxiliary grouping vectors.

Proof: Let the vertices of G_i and G_j be ordered according to the projected dependence vector $\bar{d}_l^p$ as $(u_0^p, \cdots, u_{r-1}^p)$ and $(v_0^p, \cdots, v_{r-1}^p)$, respectively. That is, $u_j^p = u_0^p + j\bar{d}_l^p$ and $v_j^p = v_0^p + j\bar{d}_l^p$ for $1 \le j \le r-1$. Suppose G_j is the forward neighboring group of G_i as shown in Fig. 10(a), i.e., $v_0^p = u_0^p + r\bar{d}_l^p$. Since the grouping is along the direction $\bar{d}_l^p$, there is only one vertex v_0^p depending on u_{r-1}^p along $\bar{d}_l^p$. Hence, G_i only depends on G_j along the direction $\bar{d}_l^p$.

Assume that G_j is the forward neighboring group of G_i along $\bar{d}_k^p \in \Psi$ as shown in Fig. 10(b). Because the base vertices are chosen along the auxiliary grouping vector $\bar{d}_k^p$, then $v_0^p = u_0^p + \bar{d}_k^p$. For the vertices $u_j^p = u_0^p + j\bar{d}_l^p \in G_i$ and $v_j^p = v_0^p + j\bar{d}_l^p \in G_j$, we know that

$$v_j^p = v_0^p + j\bar{d}_l^p = u_0^p + \bar{d}_k^p + j\bar{d}_l^p = (u_0^p + j\bar{d}_l^p) + \bar{d}_k^p = u_j^p + \bar{d}_k^p.$$

That is to say, each vertex in G_j depends on a vertex of G_i along $\bar{d}_k^p$, respectively. Therefore, G_i only depends on one

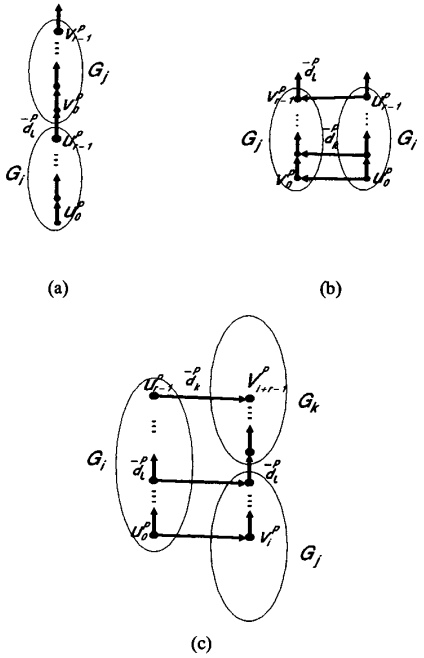


Fig. 10. (a) The dependence relation between groups along the direction of grouping vector $\bar{d}_l^p$. (b) The dependence relation between groups along the direction of auxiliary grouping vector $\bar{d}_k^p$. (c) The dependence relation between groups along the direction of vector $\in D^p - \left(\Psi \cup \left\{\bar{d}_l^{-p}\right\}\right)$.

group G_j along the direction $\bar{d}_k^p \in \Psi$. □

Lemma 3: A group depends on at most two groups along the direction of each projected dependence vector $\in D^p - (\Psi \cup \{\bar{d}_l^p\})$.

Proof: Let the group $G_i = \{u_0^p, \cdots, u_{r-1}^p\}$ and u_0^p be the base vertex of G_i. These vertices are ordered along the direction of grouping vector $\bar{d}_l^p$, i.e., $u_j^p = u_0^p + j\bar{d}_l^p$, for $1 \le j \le r-1$. Assume that there is a projected point v_i^p which depends on u_0^p along $\bar{d}_k^p \in D^p - (\Psi \cup \{\bar{d}_l^p\})$, that is, $v_i^p - u_0^p = \bar{d}_k^p$, and the projected points $v_{i+1}^p, \cdots, v_{i+r-1}^p$ are dependent on $u_1^p, \cdots, u_{r-1}^p$ along the direction of $\bar{d}_k^p$, respectively. The relationship between $u_0^p, \cdots, u_{r-1}^p$ and $v_i^p, \cdots, v_{i+r-1}^p$ is depicted in Fig. 10(c).

Since the groups are grouped along $\bar{d}_l^p$ and every group has r projected points except the groups at the boundary, these r projected points $v_i^p, \cdots, v_{i+r-1}^p$ will belong to at most two groups. Therefore, the group G_i sends data to at most two groups in the direction of $\bar{d}_k^p$ which is not a grouping or auxiliary grouping vector. □

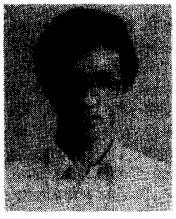
ACKNOWLEDGMENT

The authors would like to thank the anonymous referees for their helpful suggestions and comments. They have improved both the content and the presentation of this paper.

REFERENCES

[1] W. C. Athas and C. L. Seitz, "Multicomputers: Message-passing concurrent computers," *IEEE Comput. Mag.*, pp. 9–24, Aug. 1988.
[2] U. Banerjee, S. C. Chen, D. J. Kuck, and R. A. Towle, "Time and parallel processor bounds for Fortran-like loops," *IEEE Trans. Comput.*, vol. C-28, pp. 660–670, Sept. 1979.
[3] M. C. Chen, "A design methodology for synthesizing parallel algorithms and architectures," *J. Parallel Distributed Comput.*, vol. 3, pp. 461–491, 1986.
[4] ——, "The generation of a class of multipliers: Synthesizing highly parallel algorithms in VLSI," *IEEE Trans. Comput.*, vol. 37, pp. 329–338, Mar. 1988.
[5] E. H. D'Hollander, "Partitioning and labeling of index sets in do loops with constant dependence," in *Proc. 1989 Int. Conf. Parallel Processing*, vol. II, 1989, pp. 139–144.
[6] R. C. Gonzalez and P. Wintz, *Digital Image Processing*, 2nd ed. Reading, MA: Addison-Wesley, 1987.
[7] R. Gupta, "Synchronization and communication costs of loop partitioning on shared-memory multiprocessor systems," in *Proc. 1989 Int. Conf. Parallel Processing*, vol. II, Aug. 1989, pp. 23–30.
[8] C. T. King and L. M. Ni, "Pipelined data-parallel algorithms: Part II—design," *IEEE Trans. Parallel Distributed Syst.*, vol. 1, pp. 486–499, Oct. 1990.
[9] L. Lamport, "The parallel execution of Do loops," *Commun. ACM*, vol. 17, no. 2, pp. 83–93, Feb. 1974.
[10] S. Lang, *Linear Algebra*. Reading, MA: Addison-Wesley, 1986.
[11] P.-Z. Lee and Z. M. Kedem, "Synthesizing linear array algorithms from nested for loop algorithms," *IEEE Trans. Comput.*, vol. C-37, pp. 1578–1598, Dec. 1988.
[12] G. J. Li and B. W. Wah, "The design of optimal systolic arrays," *IEEE Trans. Comput.*, vol. C-34, pp. 66–77, Jan. 1985.
[13] L. S. Liu, C. W. Ho, and J. P. Sheu, "On the parallelism of nested for-loops using index shift method," in *Proc. 1990 Int. Conf. Parallel Processing*, vol. II, PA, Aug. 1990, pp. 119–123.
[14] W. L. Miranker and A. Winkler, "Spacetime representation of computational structures," *Computing*, vol. 32, pp. 93–114, 1984.
[15] 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.
[16] D. A. Padua, "Multiprocessor: Discussions of some theoretical and practical problems," Ph.D. dissertation, Univ. of Illinois at Urbana–Champaign, Urbana, IL, Nov. 1979.
[17] 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.
[18] J.-K. Peir and R. Cytron, "Minimum distance: A method for partitioning recurrences for multiprocessors," *IEEE Trans. Comput.*, vol. 38, pp. 1203–1211, Aug. 1989.
[19] P. Sadayappan and F. Ercal, "Nearest-neighbor mapping of finite element graphs onto processor meshes," *IEEE Trans. Comput.*, vol. C-36, pp. 1408–1424, Dec. 1987.
[20] W. Shang and J. A. B. Fortes, " Independent partitioning of algorithms with uniform dependencies," in *Proc. 1988 Int. Conf. Parallel Processing*, 1988, pp. 26–33.

Jang-Ping Sheu (S'85–M'86), for a photograph and biography, see the July 1991 issue of this TRANSACTIONS, p. 317.



Tsu-Huei Tai was born in Taiwan, Republic of China, on November 23, 1966. He received the B.S. degree in information and computer engineering from Chun Yuan University, and the M.S. degree in computer and electrical engineering from National Central University, Taiwan, in 1988 and 1990, respectively.

His current research interests are distributed systems and parallel processing.