Optimizing Data Alignment for Data Parallel Programs *

Hong Xu and Lionel M. Ni

Department of Computer Science
Michigan State University
East Lansing, MI 48824-1027
{xuh,ni}@cps.msu.edu

Abstract

Data decomposition across processors is critical to the performance of data parallel programs on distributed-memory machines. The data decomposition problem involves data alignment and data distribution. This paper addresses the data alignment phase which can be classified into slope alignment and offset alignment. We propose a data reference graph (DRG) model. Based on the DRG model, a slope alignment heuristic algorithm and an offset alignment heuristic algorithm are proposed for the purpose of minimizing interprocessor communication. Such a DRG-based data alignment framework makes our work unique from other related work. The time complexity of both proposed algorithms are in the linear order of distinct references given in a program structure.

1 Introduction

The data parallel programming model is considered as one of the most appropriate programming models for distributed-memory machines, especially massively parallel computers. A data parallel program typically decomposes global data arrays across processor local memories and each local processor only works on the data it owns [1]. Based on such an owner-computes rule, data decomposition determines both processor workload balance and interprocessor communication overhead. Data decomposition across processors is critical to the performance of data parallel programs on distributed-memory machines.

The data decomposition problem involves *data distribution*, which deals with how data arrays should be distributed, and *data alignment*, which deals with how data arrays should be aligned with respect to one another. The process of data alignment can be classified into two steps: *slope alignment* and *offset alignment*. Slope alignment determines the slope of parallel hyperplanes in which each array is partitioned. Offset alignment determines the offset of parallel hyperplanes in various arrays with respect to the local processor memory. A major task of data alignment is to minimize interprocessor communication generated by a data parallel program. The emphasis of this paper is on data alignment. Issues of data distribution and processor workload balance are beyond the scope of this paper [2].

The data alignment problem has been shown to be NP-complete [3]. Several heuristics [3, 4, 5, 6, 7] have been proposed to automatically determine an appropriate data alignment scheme. A survey of some of these heuristics and their limitations and drawbacks can be found in [2].

In this paper, we propose a *data reference graph* (DRG) model which can be used to resolve both slope alignment and offset alignment problems. This DRG-based data alignment framework makes our work unique from other related work. The alignment-distribution graph (ADG) model [5] is developed to optimize data alignment in the program structure which can be represented by directed acyclic graph. The DRG model, nevertheless, is proposed for global optimizations with regard to arrays referenced in parallelizable loops. In the ADG model, multi-dimensional arrays are restricted to be partitioned based on each dimension. However, the DRG model can deal with more general cases where multi-dimensional arrays may be partitioned into a family of parallel hyperplanes. Our DRG model is also different from the data-computation decomposition algorithms given in [7]. Wile the algorithms in [7] can only resolve the slope alignment problem, our DRG model can find both efficient slope alignment and efficient offset alignment.

The rest of this paper is organized as follows. Section 2 describes the program model and alignment specification used in this paper. Section 3 uses an example to illustrate the issues involved in slope alignment and offset alignment. Section 4 defines the DRG. Section 5 and Section 6 present the new slope alignment and offset alignment algorithms, respectively. Section 7 concludes the paper.

*This work was supported in part by NSF grants CDA-9121641 and MIP-9204066, and DOE grant DE-FG02-93ER25167.

2 Assumptions

2.1 Program Model

This paper only concerns distributed-memory systems in which a message transmission must be invoked by a remote data access. Arrays are distributed across different local processor memories. The owner-computes rule is followed in the way that no array element is allowed to be replicated on two processor local memories and each processor only works on the left-hand-side (LHS) array elements it owns. The program model used in this paper can be formalized as follows:

- Only the parallel construct, *forall* structure [8], is considered.
- Each array variable is referenced once in an statement, named as a *single-occurrence* statement.
- Only two-dimensional data arrays are considered. The size of each array is assumed to be $n \times n$.
- Suppose an array element $A(a_{11}i + a_{12}j + a_{10}, a_{21}i + a_{22}j + a_{20})$ is referenced in an assignment statement (on either the LHS or the RHS). It is assumed that $\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$ is unimodular.[2]

The syntax and semantics of a *forall* structure can be found in HPF [8]. A *forall* structure is free of loop-carried dependence. An example of a *forall* structure is shown in Figure 1. Note that in Figure 1, the loop boundary should be written as "(i=1:n-1;j=1:n-1;WHERE(i+j≤n+1))". However, for ease of presentation, the loop boundary conditions are neglected for the examples used in this paper.

```
     FORALL (i=1:n;j=1:n)
s1:     Z(i+1,j)=Y(j,i)
s2:     Y(i,j)=X(i,i+j-1)+Z(i+1,j+1)
     END FORALL
```

Figure 1. Example 1

Any assignment statement can be transformed to the equivalent single-occurrence statements by using extra temporary array variables. For example, the statement

```
FORALL (i=1:n,j=1:n)
   A[i,j]=A[i+1,j-1]
END FORALL
```

can be transformed into the following equivalent single-occurrence statements.

[2]An integer matrix is unimodular if it is square and the absolute value of its determinant is one.

```
FORALL (i=1:n,j=1:n)
   T1[i,j]=A[i+1,j-1]
   A[i,j]=T1[i,j]
END FORALL
```

There is no concern of memory space here because these temporary arrays are only necessary for the sake of alignment analysis and only original arrays will take place in the code generation.

The assumptions for two-dimensional arrays and unimodular subscript matrices are for the ease of explanation. The proposed slope alignment and offset alignment algorithms can be easily extended to the general case of m-dimensional data arrays and invertible subscript matrices [2].

Definition 1 *If array A is referenced at the LHS and array B is referenced at the RHS in statement s_k, the def/use relationship between A and B is called a* reference *and is represented by the symbolic form "$A \leftarrow B@s_k$".*[3]

In Example 1, we have "$Z \leftarrow Y@s_1$", "$Y \leftarrow X@s_2$", and "$Y \leftarrow Z@s_2$".

Definition 2 *If array element $A(a_{11}i + a_{12}j + a_{10}, a_{21}i + a_{22}j + a_{20})$ is referenced in statement s_k, $F_{A,k} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$ is called* access matrix, *and $f_{A,k} = \begin{pmatrix} a_{10} \\ a_{20} \end{pmatrix}$ is called* offset vector. *The array element can be written as $A(F_{A,k}\begin{pmatrix} i \\ j \end{pmatrix} + f_{A,k})$ and $F_{A,k}\begin{pmatrix} i \\ j \end{pmatrix} + f_{A,k}$ is called* access function.

In Example 1, $F_{Z,1} = F_{Z,2} = F_{Y,2} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$, $F_{Y,1} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, $F_{X,2} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$, $f_{Y,1} = f_{Y,2} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$, $f_{Z,1} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$, $f_{Z,2} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$, and $f_{X,2} = \begin{pmatrix} 0 \\ -1 \end{pmatrix}$.

2.2 Alignment Specification

Elements in a $n \times n$ array have one-to-one correspondence to grids in a $n \times n$ grid space. For this reason, the grid space which represents a 2D array is called the *data domain*.

In this paper, the following alignment primitive is used:

ALIGN $A(a_1 i_A + a_2 j_A)$ WITH $B(b_1 i_B + b_2 j_B + d_{B,A})$

where i_A and j_A are indices of data domain A, i_B and j_B are indices of data domain B, and a_i $(i = 1, 2)$,

[3]The notation "$A \leftarrow B@s_k$" is not ambiguous based on the assumption of the single occurrence statement.

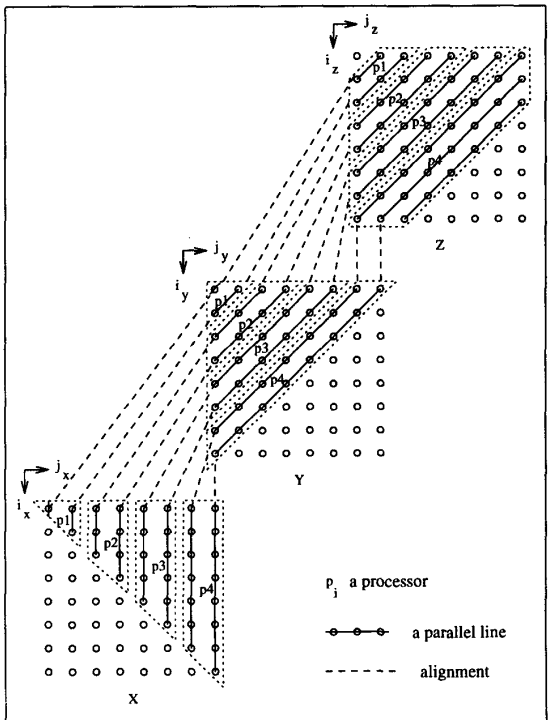

Figure 2. Decomposition of arrays X, Y, and Z in Example 1

b_j $(j = 1, 2)$ and $d_{B,A}$ are integers. This alignment primitive can be interpreted as follows. Array A is partitioned in a family of parallel lines[4] whose equations are $a_1 i_A + a_2 j_A = c_1$. Array B is partitioned in a family of parallel lines whose equations are $b_1 i_B + b_2 j_B = c_2$. The line in A whose equation is $a_1 i_A + a_2 j_A = c$ and the line in B whose equation is $b_1 i_B + b_2 j_B + d_{B,A} = c$ are aligned with each other. Here c, c_1, and c_2 can be any integer numbers. For simplicity, we denote $D_A = (a_1, a_2)$ and $D_B = (b_1, b_2)$. D_A and D_B are called *alignment slopes* and $d_{B,A}$ is called *alignment offset* of B with respect to A, or *alignment offset* for short. The proposed alignment primitive is an extension to HPF specification [8]. However, in this paper it is only used for the sake of alignment analysis and not for the purpose of a language directive.

In Example 1(Figure 2), alignment slopes $D_X = (0, 1)$, $D_Y = (1, 1)$, $D_Z = (1, 1)$ are specified for arrays X, Y, and Z, respectively. Moreover, alignment offsets $d_{X,Y} = 1$ and $d_{Z,Y} = -1$ are specified when array X and array Z are aligned with array Y, respectively. Overall, data alignment in Example 1 can be formalized as follows.

[4] A hyperplane in a 2D space is usually called a line.

ALIGN Y$(i_Y + j_Y)$ WITH X$(j_X + 1)$
ALIGN Y$(i_Y + j_Y)$ WITH Z$(i_Z + j_Z - 1)$

In Figure 2, the size of each array is declared as 8×8. Each array element is represented by a circle. The parallel line partition on each array is represented by a family of solid lines. Array X is partitioned in columns. Arrays Y and Z are partitioned along diagonal. Some circles are not covered by solid lines since the corresponding array elements are beyond loop boundaries. Two solid lines in different array domains are aligned with each other if they are connected by a dashed line. The line in X whose equation is $j_X + 1 = c$, the line in Y whose equation is $i_Y + j_Y = c$, and the line in Z whose equation is $i_Z + j_Z - 1 = c$ are aligned with one another, where c can be any integer number.

3 An Example

In this section, we study how to obtain the alignment results for Example 1. Details of slope alignment and offset alignment algorithms will be presented in Section 5 and Section 6.

3.1 Alignment and Access Function

In order to eliminate interprocessor communication, we first study the relationship between alignment and access function. Consider reference "$Y \leftarrow X@s_2$" in Example 1 (Figure 1). Suppose $Y(i_Y, j_Y)$ and $X(i_X, j_X)$ are referenced in iteration $\begin{pmatrix} i \\ j \end{pmatrix}$. We have

$$\begin{pmatrix} i_Y \\ j_Y \end{pmatrix} = F_{Y,2} \begin{pmatrix} i \\ j \end{pmatrix} + f_{Y,2}$$
$$\begin{pmatrix} i_X \\ j_X \end{pmatrix} = F_{X,2} \begin{pmatrix} i \\ j \end{pmatrix} + f_{X,2}$$

Since $F_{Y,2}$ and $F_{X,2}$ are invertible, the above equations can be rewritten as

$$\begin{pmatrix} i \\ j \end{pmatrix} = F_{Y,2}^{-1} \begin{pmatrix} i_Y \\ j_Y \end{pmatrix} - F_{Y,2}^{-1} f_{Y,2}$$
$$\begin{pmatrix} i \\ j \end{pmatrix} = F_{X,2}^{-1} \begin{pmatrix} i_X \\ j_X \end{pmatrix} - F_{X,2}^{-1} f_{X,2}$$

where $F_{Y,2}^{-1}$ is the inverted matrix for $F_{Y,2}$ and $F_{X,2}^{-1}$ is the inverted matrix for $F_{X,2}$. By substitution, we get

$$F_{Y,2}^{-1} \begin{pmatrix} i_Y \\ j_Y \end{pmatrix} - F_{Y,2}^{-1} f_{Y,2} = F_{X,2}^{-1} \begin{pmatrix} i_X \\ j_X \end{pmatrix} - F_{X,2}^{-1} f_{X,2}$$

It can be rewritten as

$$\begin{pmatrix} i_Y \\ j_Y \end{pmatrix} = F_{Y,2} F_{X,2}^{-1} \begin{pmatrix} i_X \\ j_X \end{pmatrix} + f_{Y,2} - F_{Y,2} F_{X,2}^{-1} f_{X,2} \quad (1)$$

Let D_Y be the alignment slope for Y. By applying D_Y to both sides of Equation (1), we get

$$D_Y \begin{pmatrix} i_Y \\ j_Y \end{pmatrix} = D_Y F_{Y,2} F_{X,2}^{-1} \begin{pmatrix} i_X \\ j_X \end{pmatrix} + D_Y f_{Y,2} - D_Y F_{Y,2} F_{X,2}^{-1} f_{X,2} \quad (2)$$

By the definition of alignment specification, for a fixed integer c, Y elements on line $D_Y \begin{pmatrix} i_Y \\ j_Y \end{pmatrix} = c$ are owned by a local processor. Hence, reference "$Y \leftarrow X@s_2$" is communication-free if X elements on line $D_Y F_{Y,2} F_{X,2}^{-1} \begin{pmatrix} i_X \\ j_X \end{pmatrix} + D_Y f_{Y,2} - D_Y F_{Y,2} F_{X,2}^{-1} f_{X,2} = c$ are owned by the same local processor. This implies that $Y(D_Y \begin{pmatrix} i_Y \\ j_Y \end{pmatrix})$ should be aligned with $X(D_X \begin{pmatrix} i_X \\ j_X \end{pmatrix} + d_{X,Y})$ where

$$D_X = D_Y F_{Y,2} F_{X,2}^{-1} \quad (3)$$

$$d_{X,Y} = D_Y f_{Y,2} - D_Y F_{Y,2} F_{X,2}^{-1} f_{X,2} \quad (4)$$

Arrays X and Y are *slope aligned* if Equation (3) holds. *Organization communication* occurs if two arrays specified by a reference are not slope aligned. Similarly, arrays X and Y are *offset aligned* if Equation (3) holds but Equation (4) does not. *Neighboring communication* occurs if two arrays specified by a reference are not offset aligned.

3.2 Slope Alignment

By the definition of alignment specification, array X is partitioned into a family of parallel lines whose equations are $D_X \begin{pmatrix} i_X \\ j_X \end{pmatrix} = c_1$. If X and Y are not slope aligned, i.e.$D_X \neq D_Y F_{Y,2} F_{X,2}^{-1}$, X elements in line $D_Y F_{Y,2} F_{X,2}^{-1} \begin{pmatrix} i_X \\ j_X \end{pmatrix} + D_Y f_{Y,2} - D_Y F_{Y,2} F_{X,2}^{-1} f_{X,2} = c$ must be distributed across different processors. This implies that in Equation (2), to write a Y element, the local processor almost always requires a remote access to the corresponding X element. Assume array elements are evenly distributed across processors. The number of remote array elements referenced by a local processor is approximately equal to $\frac{n^2}{p}$ where p is the number of processors.

Definition 3 *The communication cost generated by a reference is $\frac{n^2}{p}$ if two $n \times n$ arrays specified by the reference are not slope aligned.*

Equation (3) shows that alignment slopes D_X and D_Y are only a function of access matrices $F_{X,2}$ and $F_{Y,2}$. Therefore, slope alignment analysis can be carried out without considering offset vectors.

Similar to Equation (3), in Example 1, references "$Z \leftarrow Y@s_1$", "$Y \leftarrow X@s_2$", and "$Y \leftarrow Z@s_2$" are organization communication-free if the following equations have non-trivial solutions for D_X, D_Y, and D_Z.

$$D_Y = D_Z F_{Z,1} F_{Y,1}^{-1} \quad (5)$$

$$D_X = D_Y F_{Y,2} F_{X,2}^{-1} \quad (6)$$

$$D_Z = D_Y F_{Y,2} F_{Z,2}^{-1} \quad (7)$$

Replacing D_Y in Equation (7) by Equation (5), we get

$$D_Z = D_Z F_{Z,1} F_{Y,1}^{-1} F_{Y,2} F_{Z,2}^{-1}$$

It can be rewritten as

$$D_Z (I - F_{Z,1} F_{Y,1}^{-1} F_{Y,2} F_{Z,2}^{-1}) = 0 \quad (8)$$

Substituting the values of $F_{Z,1}$, $F_{Y,1}^{-1}$, $F_{Y,2}$, and $F_{Z,2}^{-1}$, we get

$$D_Z \left(\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} - \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \right) = 0$$

which conducts $D_Z = (1, 1)$. The result of D_Z is determined such that two integer elements in vector D_Z are prime to each other. Substituting the value of D_Z into Equation (5), we have $D_Y = D_Z F_{Z,1} F_{Y,1}^{-1} = (1, 1)$. Substituting the value of D_Y into Equation (6), we have

$$D_X = D_Y F_{Y,2} F_{X,2}^{-1} = (1, 1) \begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix} = (0, 1)$$

The above solutions of D_X, D_Z, and D_Y guarantee that three references are organization communication-free.

3.3 Offset Alignment

Neighboring communication occurs when two arrays X and Y are slope aligned but not offset aligned. Neighboring communication can be significantly reduced by *block* distribution in which parallel lines in a data domain are consecutively assigned to each processor. As a result, the neighboring communication cost depends on the number of parallel lines which are mismatched in two corresponding data domains. This can be formally described as follows:

Definition 4 *Two $n \times n$ matrices X and Y are specified by a reference in statement s_k. $D_X = D_Y F_{Y,k} F_{X,k}^{-1}$. The neighboring communication cost generated by the reference is $n|d_{X,Y} - (D_Y f_{Y,k} - D_X f_{X,k})|$.*

In the above definition, n is the maximum number of data elements in a line in a $n \times n$ data domain. The neighboring communication cost is calculated based

on the maximum number of remote data elements a local processor requires to access. Neighboring communication is only considered for references which are organization communication-free.

In Figure 2, we assume that the number of available processors is four. Parallel lines in arrays X, Y, and Z are distributed to four processors in *block* fashion, respectively. Block distribution overlaps neighboring communication. For example, in reference "$Y \leftarrow Z@s_2$", to write a Y element in line $i_Y + j_Y = 6$, a Z element in line $i_Z + j_Z = 8$ is required to be accessed. Though these two lines are not aligned, they are owned by the same processor p_3 due to *block* distribution. However, line $i_Y + j_Y = 7$ and line $i_Z + j_Z = 9$ are mismatched and the neighboring communication cost for reference "$Y \leftarrow Z@s_2$" is 6, the number of data elements in the line $i_Y + j_Y = 7$.

The overall neighboring communication cost can be obtained by accumulating neighboring communication cost generated by each reference. In Example 1, the overall cost is equal to

$$\begin{aligned} & n|d_{X,Y} - (D_Y f_{Y,2} - D_X f_{X,2})| \\ & +n|d_{Z,Y} - (D_Y f_{Y,1} - D_Z f_{Z,1})| \\ & +n|d_{Z,Y} - (D_Y f_{Y,2} - D_Z f_{Z,2})| \end{aligned} \quad (9)$$

Substituting values of access matrices and offset vectors into Equation (9), the neighboring communication cost can be minimized if a solution of $d_{X,Y}$ and $d_{Z,Y}$ can minimizes

$$|d_{X,Y} - 1| + |d_{Z,Y} + 1| + |d_{Z,Y} + 2|$$

An optimal solution is $d_{X,Y} = 1$ and $d_{Z,Y} = -1$. As a result, references "$Y \leftarrow X@s_2$" and "$Z \leftarrow Y@s_1$" are neighboring communication-free but reference "$Y \leftarrow Z@s_2$" is not.

4 Data Reference Graph

Given a *forall* structure, a *data reference graph* (DRG) is constructed as follows. An array referenced in the *forall* structure is represented by a node. For an array which has multiple instances referenced in the *forall* structure body, there is only one corresponding node in the DRG. There is a distinct edge connecting two nodes for each reference between the corresponding two arrays. A DRG is undirected.

Figure 3 shows the DRG for Example 1. In Figure 3, nodes x, y, and z correspond to arrays X, Y, and Z, respectively. In this paper, arrays are represented by characters in upper case while nodes are represented by the corresponding lower case characters. Edge e_1 connects nodes x and y due to reference "$Y \leftarrow X@s_2$". Edges e_2 and e_3 are incident with the same two nodes y and z due to references "$Z \leftarrow Y@s_1$" and "$Y \leftarrow Z@s_2$".

Definition 5 *Two or more edges which are incident with the same two nodes are called* parallel edges. *A node is parallel to the orther if there exist parallel edges between them.*

In Figure 3, edge e_2 and e_3 are parallel edges. Node y is parallel to node z, and vice verse.

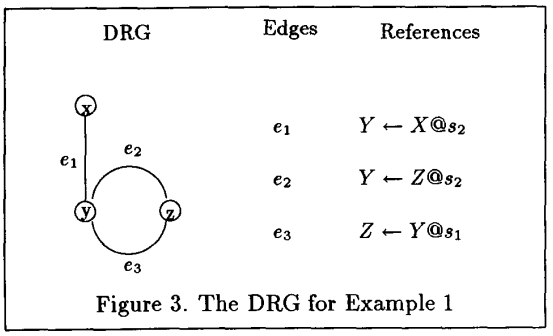

Figure 3. The DRG for Example 1

If a DRG is a tree, the optimal slope alignment and offset alignment can be found such that every reference is organization communication-free and neighboring communication-free. However, if a DRG contains a cycle, some reference(s) may not be communication-free. In a DRG with a cycle, there exist at least two disjoint paths between any two nodes in the cycle. As a result, the communication-free alignment requirements imposed by two paths may conflict with each other. In Figure 3, there exists a cycle $y \xrightarrow{e_2} z \xrightarrow{e_3} y$. By analysis in Section 3, arrays Y and Z have to be partitioned along diagonal in order to eliminate organization communication. Nevertheless, neighboring communication can not be avoided due to the conflict requirements imposed by references "$Z \leftarrow Y@s_1$" and "$Y \leftarrow Z@s_2$".

In the next two sections, two heuristic algorithms are presented to reduce organization communication and neighboring communication for a general DRG, respectively. Example 2 (Figure 4) will be used to illustrate the proposed algorithms.

```
     FORALL (i=1:n;j=1:n)
s1:     B[i,j+1]=C[i,j]
s2:     A[i,j]=Q[i+j+1,j]+B[j+1,i+2]
s3:     C[i,j]=B[i-1,j]+X[i+1,j+1]
s4:     Y[i+1,j]=X[i+1,j]+Z[j,i]
s5:     Z[i,j]=B[i+1,i+j]+A[i,j]+X[i-1,j]
s6:     C[i+j,2i+j]=Y[i,j+1]
     END FORALL
```

Figure 4. Example 2

5 Slope Alignment Algorithm

Definition 6 *Given a path L in a DRG, the function ψ on L can be defined by the following rules.*

1. *If edge* $e = (u, v)$ *represents reference* "$V \leftarrow U@s_k$",

$$\psi(u \xrightarrow{e} v) = F_{U,k}F_{V,k}^{-1}$$

2. *If* $\psi(u \xrightarrow{e} v)$ *is defined in step (1),*

$$\psi(v \xrightarrow{e} u) = \psi^{-1}(u \xrightarrow{e} v)$$

3. *For path* $L = u_1 \xrightarrow{e_1} u_2 \xrightarrow{e_2} \ldots \xrightarrow{e_{m-1}} u_m$,

$$\psi(L) = \psi(u_1 \xrightarrow{e_1} u_2)\psi(u_2 \xrightarrow{e_2} \ldots \xrightarrow{e_{m-1}} u_m)$$

The value of $\psi(L)$ *is a matrix.* $\psi^{-1}(L)$ *refers to the inverted matrix of* $\psi(L)$. $\psi(L_1)\psi(L_2)$ *refers to the matrix multiplication of* $\psi(L_1)$ *and* $\psi(L_2)$.

Though a DRG is undirect, the ψ function defined on each path depends on the source, the destination, and the order of intermediate vertices it traverses. The ψ function is well-defined if its value for each edge is well-defined. For Example 1 (Figure 3), $\psi(z \xrightarrow{e_3} y) = F_{Z,1}F_{Y,1}^{-1}$, $\psi(y \xrightarrow{e_1} x) = F_{Y,2}F_{X,2}^{-1}$, and $\psi(y \xrightarrow{e_2} z) = F_{Y,2}F_{Z,2}^{-1}$. The slope alignment problem can be well modeled by ψ function. Equation (5) can be represented by $D_Y = D_Z\psi(z \xrightarrow{e_3} y)$. Equation (6) can be represented by $D_X = D_Y\psi(y \xrightarrow{e_1} x)$. Equation (7) can be represented by $D_Z = D_Y\psi(y \xrightarrow{e_2} z)$. Equation (8) can be represented by $D_Z(I - \psi(z \xrightarrow{e_3} y \xrightarrow{e_2} z)) = 0$ where $\psi(z \xrightarrow{e_3} y \xrightarrow{e_2} z) = \psi(z \xrightarrow{e_3} y)\psi(y \xrightarrow{e_2} z)$.

Given a DRG G with the well-define ψ function, the slope alignment algorithm can be formalized by the following rules.

1 Select an arbitrary node, say u, in G.

2.a If there exists a node v such that v is a neighboring node of u and v is not parallel to u,

$$D_V = D_U\psi(u \xrightarrow{e} v)$$

where $e = (u, v)$ is named as a *slope alignment edge.*

2.b Else if all neighboring nodes of u are parallel to u, select an arbitrary neighboring node v. Suppose $e_1 = (u, v)$, $e_2 = (u, v)$, ..., and $e_m = (u, v)$ are parallel edges between nodes u and v. Let $\mathcal{D} = \{\psi(u \xrightarrow{e_k} v) \mid \text{rank}(\psi(u \xrightarrow{e_k} v)) = 1 \text{ where} 1 \leq k \leq m\}$. Find a $\psi(u \xrightarrow{e_\ell} v)$ such that it is the majority in $\mathcal{D}$.

$$D_V = D_U\psi(u \xrightarrow{e_\ell} v)$$

where $e_\ell = (u, v)$ is a *slope alignment edge.* Let $\mathcal{R}_v = \{\{\psi(u \xrightarrow{e_\ell} v), \psi(u \xrightarrow{e_k} v)\} \mid k \neq \ell \text{ and } \psi(u \xrightarrow{e_k} v) \in \mathcal{D}\}$

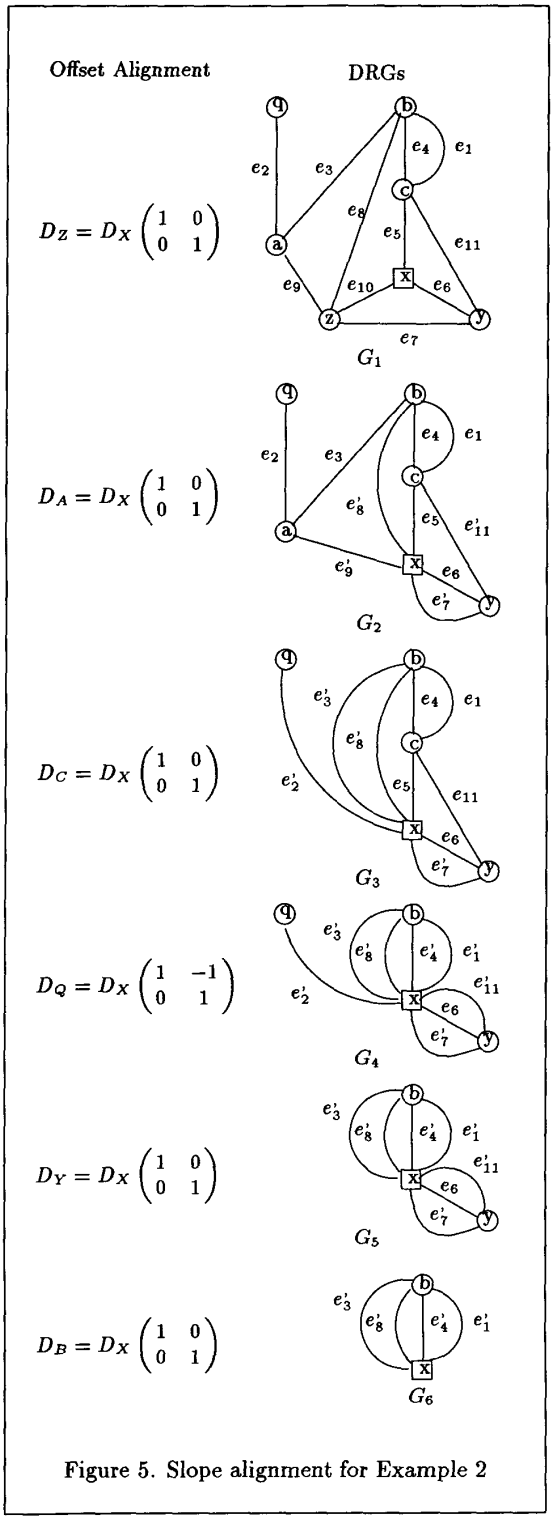

Figure 5. Slope alignment for Example 2

3 Suppose $e = (u, v)$ is the slope alignment edge obtained in step (2.a) or step (2.b). For each edge $e' = (v, w)$ such that node w is different from node u, create a new edge $e'' = (u, w)$ which connects w and u. The value of ψ function for new edge e' is defined as follows:

$$\psi(u \xrightarrow{e''} w) = \psi(u \xrightarrow{e} v \xrightarrow{e'} w)$$
$$\psi(w \xrightarrow{e''} u) = \psi^{-1}(u \xrightarrow{e''} w)$$

4 Delete node v and the edges which are incident with v.

5 Goto Step (2.a) until only node u is left.

6 Let $\{\psi(u \xrightarrow{e_\ell} v), \psi(u \xrightarrow{e_k} v)\}$ be the majority in the set union $\cup_{v \in G} \mathcal{R}_v$. Find a D_U such that

$$D_U(I - \psi(u \xrightarrow{e_\ell} v \xrightarrow{e_k} u)) = 0$$

Figure 5 shows offset alignment analysis for Example 2 by using the offset alignment algorithm above. The algorithm begins with the original DRG G_1. Steps (2.a)-(4) are executed for each construction from G_i to G_{i+1} $(1 \leq i \leq 5)$. The values of ψ function for edges in DRG G_1 are given as follows: $\psi(c \xrightarrow{e_1} b) = I$, $\psi(q \xrightarrow{e_2} a) = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$, $\psi(b \xrightarrow{e_3} a) = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, $\psi(b \xrightarrow{e_4} c) = I$, $\psi(x \xrightarrow{e_5} c) = I$, $\psi(x \xrightarrow{e_6} y) = I$, $\psi(z \xrightarrow{e_7} y) = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, $\psi(b \xrightarrow{e_8} z) = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$, $\psi(a \xrightarrow{e_9} z) = I$, $\psi(x \xrightarrow{e_{10}} z) = I$, and $\psi(y \xrightarrow{e_{11}} c) = \begin{pmatrix} -1 & 1 \\ 2 & -1 \end{pmatrix}$.

In step (1), node x is selected. Consider the construction from G_1 to G_2. In G_1, nodes c, y, and z are neighboring nodes of x and none of them is parallel to x. In step (2.a), node z is randomly selected. Since edge e_{10} connects x and z, $D_Z = D_X\psi(x \xrightarrow{e_{10}} z)$. In step (3), three new edges e_7', e_8', and e_9' are generated to replace edges $e_7 = (z, y)$, $e_8 = (z, b)$, and $e_9 = (z, a)$, respectively. The values of ψ function for new edges can be obtained by

$$\psi(x \xrightarrow{e_7'} y) = \psi(x \xrightarrow{e_{10}} z \xrightarrow{e_7} y) = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$
$$\psi(x \xrightarrow{e_8'} b) = \psi(x \xrightarrow{e_{10}} z \xrightarrow{e_8} b) = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$$
$$\psi(x \xrightarrow{e_9'} a) = \psi(x \xrightarrow{e_{10}} z \xrightarrow{e_9} a) = I$$

Edge e_7' represents reference "$Y \leftarrow Z@s_4$" which edge e_7 represents. Edge e_8' represents reference "$Z \leftarrow B@s_5$" which edge e_8 represents. Edge e_9' represents reference "$Z \leftarrow A@s_5$" which edge e_9 represents. In step (4), node z is collapsed and edges e_7, e_8, e_9, and e_{10} are deleted. In step (5), a new construction begins with G_2. Similar procedures are repeated and nodes a, c, and q are collapsed in the constructions of G_2, G_3, and G_4, respectively. The results of ψ function on new edges e_1', e_2', e_3', e_4', and e_{11}' are shown as follows: $\psi(x \xrightarrow{e_1'} b) = I$, $\psi(x \xrightarrow{e_2'} q) = \begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix}$, $\psi(x \xrightarrow{e_3'} b) = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, $\psi(x \xrightarrow{e_4'} b) = I$, and $\psi(x \xrightarrow{e_{11}'} y) = \begin{pmatrix} 1 & 1 \\ 2 & 1 \end{pmatrix}$.

Consider the construction from G_5 to G_6. In step (2.b), both of neighboring nodes b and y are parallel to x. Suppose node y is selected. Edges e_6, e_7', and e_{11}' are parallel edges incident with y and x. $\mathcal{D} = \{\psi(x \xrightarrow{e_6} y), \psi(x \xrightarrow{e_7'} y)\}$ because $\text{rank}(\psi(x \xrightarrow{e_{11}'} y)) = 2$. Edge e_6 is selected as the slope alignment edge. Therefore, $D_Y = D_X\psi(x \xrightarrow{e_6} y)$ and $\mathcal{R}_y = \{\psi(x \xrightarrow{e_6} y), \psi(x \xrightarrow{e_7'} y)\} = \{\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\}$.

Similar analysis is done in G_6. Edges e_1', e_3', e_4', and e_8' are parallel edges incident with b and x. $\mathcal{D} = \{\psi(x \xrightarrow{e_1'} b), \psi(x \xrightarrow{e_3'} b), \psi(x \xrightarrow{e_4'} b), \psi(x \xrightarrow{e_8'} b)\}$. Since $\psi(x \xrightarrow{e_1'} b) = I$ is the majority in $\mathcal{D}$, $D_B = D_X\psi(x \xrightarrow{e_1'} b)$. Therefore, $\mathcal{R}_b = \{\{\psi(x \xrightarrow{e_1'} b), \psi(x \xrightarrow{e_3'} b)\}, \{\psi(x \xrightarrow{e_1'} b), \psi(x \xrightarrow{e_8'} b)\}\} = \{\{\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\}, \{\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}\}\}$.

In step 6, $\{\psi(x \xrightarrow{e_1'} b), \psi(x \xrightarrow{e_3'} b)\}$ is the majority in $\mathcal{R}_y \cup \mathcal{R}_b$. To satisfy the equation

$$D_X(I - \psi(x \xrightarrow{e_1'} b \xrightarrow{e_3'} x)) = 0$$

D_X must be equal to $(1, 1)$. Alignment slopes for other arrays can be obtained by substituting the value of D_X into each equation shown in Figure 5.

6 Offset Alignment Algorithm

References which are organization communication-free compose a *slope-aligned component.* Offset alignment is only considered among references which are involved in the same slope-aligned component. All references in Example 1 are organization communication-free and thus are considered in offset alignment analysis. In Example 2, references "$Z \leftarrow B@s_5$" and "$C \leftarrow Y@s_6$" are not organization communication-free and thus are not considered in offset alignment for Example 2. Given a DRG, the subgraph induced by the slope-aligned component is called *component reference graph*, CRG for short. For Example 1, its CRG is identical to its DRG. However, the CRG for Example 2 does not include edges e_8 and e_{11}. For simplicity, it is assumed any node or edge mentioned in the rest of this section refers to a node or an edge appearing in a CRG. Alignment slope D_V is pre-determined for any array V.

Definition 7 *Given a path L in a CRG, the function ϕ on L can be defined by the following rules.*

1. *If edge* $e = (u, v)$ *represents reference "*$V \leftarrow U@s_k$*" and alignment slopes* D_U *and* D_V *are given,*

$$\phi(u \xrightarrow{e} v) = D_V f_{V,k} - D_U f_{U,k}$$

2. *If* $\phi(u \xrightarrow{e} v)$ *is defined in step (1),*

$$\phi(v, u) = -\phi(u, v)$$

3. *For path* $L = u_1 \xrightarrow{e_1} u_2 \xrightarrow{e_2} \ldots \xrightarrow{e_{m-1}} u_m$,

$$\phi(L) = \phi(u_1 \xrightarrow{e_1} u_2) + \phi(u_2 \xrightarrow{e_2} \ldots \xrightarrow{e_{m-1}} u_m)$$

The value of $\phi(L)$ *is an integer.* $-\phi(L)$ *refers to the negative value of* $\phi(L)$. $\phi(L_1) + \phi(L_2)$ *refers to the addition of* $\phi(L_1)$ *and* $\phi(L_2)$.

Similar to the ψ function, the value of the ϕ function defined on each path depends on the source, the destination, and the order of intermediate vertices it traverses. The offset alignment problem can be well modeled by ϕ function. In CRG of Example 1 (Figure 3), $\phi(z \xrightarrow{e_3} y) = -1$, $\phi(x \xrightarrow{e_1} y) = 1$, and $\phi(z \xrightarrow{e_2} y) = -2$. Equation (9) can be represented by

$$|d_{X,Y} - \phi(x \xrightarrow{e_1} y)| + |d_{Z,Y} - \phi(z \xrightarrow{e_3} y)| + |d_{Z,Y} - \phi(z \xrightarrow{e_2} y)|$$

Given a CRG G with well-defined ϕ function, the offset alignment algorithm can be formalized by the following rules:

1 Select an arbitrary node, say u, in G.

2.a If there exists a node v such that v is a neighboring node of u and v is not parallel to u,

$$d_{U,V} = \phi(u \xrightarrow{e} v)$$

where $e = (u, v)$ is named as an *offset alignment edge.*

2.b Else if all neighboring nodes of u are parallel to u, select an arbitrary neighboring node v. Suppose $e_1 = (u, v)$, $e_2 = (u, v)$, ..., and $e_m = (u, v)$ are parallel edges between nodes u and v. Without loss of generality, we further assume that $\phi(u \xrightarrow{e_k} v) \leq \phi(u \xrightarrow{e_{k+1}} v)$ where $1 \leq k \leq m - 1$.

$$d_{U,V} = \phi(u \xrightarrow{e_{\lceil \frac{m}{2} \rceil}} v)$$

where $e_{\lceil \frac{m}{2} \rceil} = (u, v)$ is named as an *offset alignment edge.*

3 Suppose $e = (u, v)$ is the offset alignment edge obtained in Step (2.a) or Step (2.b). For each edge $e' = (w, v)$ such that node w is different from node u, create a new edge $e'' = (w, u)$ which connects w and u. The value of ϕ function for new edge e'' is defined as follows:

$$\phi(u \xrightarrow{e''} w) = \phi(u \xrightarrow{e} v \xrightarrow{e'} w)$$
$$\phi(w \xrightarrow{e''} u) = -\phi(u \xrightarrow{e''} w)$$

4 Delete node v and the edges which are incident with v.

5 Goto Step (2.a) until only node u is left.

Figure 6 shows offset alignment analysis for Example 2. The algorithm begins with CRG G_1' which is a subgraph of the original DRG G_1 (Figure 5). Steps (2.a)-(4) are executed for each construction from G_i' to G_{i+1}' ($1 \leq i \leq 5$). The values of ϕ function for edges in CRG G_1' are given as follows: $\phi(c \xrightarrow{e_1} b) = 1$, $\phi(q \xrightarrow{e_2} a) = -1$, $\phi(b \xrightarrow{e_3} a) = -3$, $\phi(b \xrightarrow{e_4} c) = 1$, $\phi(x \xrightarrow{e_5} c) = -2$, $\phi(x \xrightarrow{e_6} y) = 0$, $\phi(z \xrightarrow{e_7} y) = 1$, $\phi(a \xrightarrow{e_9} z) = 0$, and $\phi(x \xrightarrow{e_{10}} z) = 1$.

In step (1), node x is selected. Consider the construction from G_1' to G_2'. In G_1', nodes c, y, and z are neighboring nodes of x and none of them is parallel to x. In step (2.a), node z is selected. Since edge e_{10} connects x and z, $d_{X,Z} = \phi(x \xrightarrow{e_{10}} z)$. In step (3), new edges e_7'' and e_9'' are generated to replace edges $e_7 = (z, y)$ and $e_9 = (z, a)$, respectively. The values of ϕ function for new edges can be obtained by

$$\phi(x \xrightarrow{e_7''} y) = \phi(x \xrightarrow{e_{10}} z \xrightarrow{e_7} y) = 1 + 1 = 2$$
$$\phi(x \xrightarrow{e_9''} a) = \phi(x \xrightarrow{e_{10}} z \xrightarrow{e_9} a) = 1 + 0 = 1$$

Edges e_7'' and e_9'' represent references "$Y \leftarrow Z@s_4$" and "$Z \leftarrow A@s_5$", respectively. In step (4), node z is collapsed and edges e_7, e_9, and e_{10} are deleted. In step (5), a new construction begins with G_2'. Similar procedures are repeated and nodes a, c, and q are collapsed in the constructions of G_2', G_3', and G_4', respectively. The results of ϕ function on new edges e_1'', e_2'', e_3'', and e_4'' are shown as follows: $\phi(x \xrightarrow{e_1''} b) = -1$, $\phi(x \xrightarrow{e_2''} q) = 2$, $\phi(x \xrightarrow{e_3''} b) = 4$, and $\phi(x \xrightarrow{e_4''} b) = -3$.

Consider the construction from G_5' to G_6'. In step (2.b), both of neighboring node b and y are parallel to x. Suppose node y is selected. Edges e_6 and e_7'' are parallel edges incident with nodes y and x. Since $\phi(x \xrightarrow{e_6} y) < \phi(x \xrightarrow{e_7''} y)$, edge e_6 is chosen as the offset alignment edge and $d_{X,Y} = \phi(x \xrightarrow{e_6} y)$. Similar analysis is done in G_6'. Edges e_1'', e_3'', and e_4'' are parallel edges incident with nodes b and x. Since $\phi(x \xrightarrow{e_4''} b) < \phi(x \xrightarrow{e_1''} b) < \phi(x \xrightarrow{e_3''} b)$, edge e_1'' is chosen as the alignment edge and $d_{X,B} = \phi(x \xrightarrow{e_1''} b)$.

A close inspection reveals that in both slope alignment and offset alignment algorithms a new generated

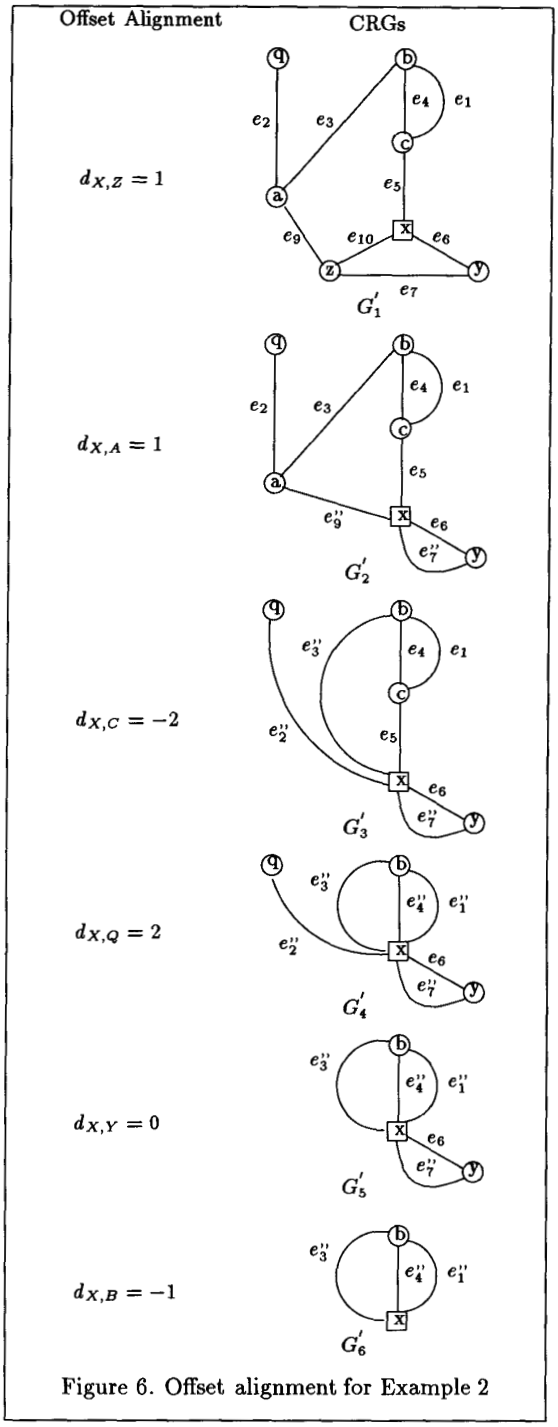

Figure 6. Offset alignment for Example 2

edge cannot be replaced by any other edge again. This property implies that the time complexity of both algorithms is in the linear order of the number of edges. The proof can be found in [2].

7 Future Work

This paper presented a DRG-based framework to resolve the data alignment problem. The concept of data reference graph can be enhanced by data flow analysis. Different remote accesses to the same context can be combined to further reduce interprocessor communication. Data reference graph model can be extended to represent data re-distribution and data re-alignment issues. In addition, since data arrays may be partitioned in a family of parallel lines with a general slope, an efficient data distribution descriptor is desired to handle the data allocation and reallocation across different processors.

References

1. S. Hiranandani, K. Kennedy, and C.-W. Tseng, "Compiling Fortran D for MIMD distributed-memory machines," *Communications of the ACM*, vol. 35, pp. 66–80, Aug. 1992.
2. H. Xu, "Optimizing data decomposition for data parallel programs," May 1994. (Ph.D thesis).
3. J. Li and M. Chen, "The data alignment phase in compiling programs for distributed-memory machines," *Journal of Parallel and Distributed Computing*, vol. 13, pp. 213–221, Oct. 1991.
4. K. Knob, J. D. Lukas, and G. L. Steel, "Data optimization:allocation of arrays to reduce communication on SIMD Machines," *Journal of Parallel and Distributed Computing*, vol. 2, pp. 102–118, Feb. 1990.
5. S. Chatterjee, J. R. Gilbert, R. Schreiber, and S. H. Teng, "Automatic array alignment in data-parallel programs," in *the Twentieth Annual ACM SIGACT/SIGPLAN Symposium on Principles of Programming Languages*, pp. 16–28, Jan. 1993.
6. J. Ramanujam and P. Sadayappan, "Compile-time techniques for data distribution in distributed memory machines," *IEEE Transactions on Parallel and Distributed Systems*, vol. 2, pp. 472–482, Oct. 1991.
7. J. M. Anderson and M. S. Lam, "Global optimizations for parallelism and locality on scalable parallel machines," in *Proceedings of the ACM SIGPLAN'93 Conference on Programming Language Design and Implementation*, pp. 112–125, June 1993.
8. High Performance Fortran Forum, "High Performance Fortran Language Specification (version 1.0)," May 1993.