Using Elementary Linear Algebra to Solve Data Alignment for Arrays with Linear or Quadratic References

Weng-Long Chang, Jih-Woei Huang, and Chih-Ping Chu

Abstract—Data alignment that facilitates data locality so that the data access communication costs can be minimized, helps distributed memory parallel machines improve their throughput. Most data alignment methods are devised mainly to align the arrays referenced using linear subscripts or quadratic subscripts with few (one or two) loop index variables. In this paper, we propose two communication-free alignment techniques to align the arrays referenced using linear subscripts or quadratic subscripts with multiple loop index variables. The experimental results from our techniques on Vector Loop and TRFD of the Perfect Benchmarks reveal that our techniques can improve the execution times of the subroutines in these benchmarks.

Index Terms—Parallel compiler, communication-free alignment, parallel computing, loop optimization, data dependence analysis, load balancing.

1 Introduction

Distributed memory multiprocessors systems have been increasingly used in scientific and engineering applications. The major shortcoming of this system is the difficulty in programming due to the lack of shared memory space [8]. The programmers or compilers in this computing architecture must be responsible for distributing the computations and data in a program over processors and managing communications among tasks. Carefully arranging the computations and data can assist the parallel system in improving throughput. This matter relates to determining which computations need to be distributed onto which processor and what data should be stored locally for the corresponding computations to access with little or no communication cost.

Over the past several years, many researchers have paid attention to maximizing parallelism and minimizing the communication cost for a given program executed on a parallel machine. Ramanujam and Sadayappan [3] considered the problem to communication-free partition data space along hyperplanes for distributed memory multiprocessors system. They presented a matrix-based formulation for this problem to determine the existence of communication-free partitions for data arrays. Feautrier [6] proposed an algorithm analogous to Gaussian elimination to determine a placement function for the problem of data and code distributions among the processors of a distributed memory supercomputer. Dion and Robert [10] introduced an access graph to model affine communications more adequately for the data and computation alignment problem when mapping affine loop nests onto distributed memory parallel computers. Lam et al. [14], [21] presented data access pattern analyzing approaches for a program with nested loops to enable the program to run on a parallel machine in a communication-free manner with some constraints. Shih et al. [22] examined the problem to communication-free partition statement-iteration and data spaces along hyperplanes for multistatements in perfect and imperfect loops. They offered the necessary and sufficient conditions for communication-free hyperplane partitions. Lee [11] showed that data redistribution is necessary for executing a sequence of do loops if the communication cost for performing this do loops sequence is larger than a threshold value. Hwang and Lee [20] proposed an expression-rewriting framework to generate communication sets for arrays in loops with block-cyclic distribution. Chung et al. [13], Liao and Chung, [19], and Hsu et al. [23] presented efficient methods to perform block-cyclic array redistribution that allow a processor not to construct send/receive data sets for a redistribution.

To properly allocate computations and data in a program over multiple processors usually involves two phases called *alignment* and *distribution*. First, the alignment phase intelligently maps computations and data onto a set of virtual processors organized into a Cartesian grid of some dimension (or a *template* in HPF Fortran term) to provide data locality in a program. The distribution phase then folds the virtual processors in the physical processors according to feasible distribution strategies. Our concern in this paper is the alignment.

Being a mapping, the alignment theoretically functions as transformation that symbolizes a linear algebra model. Kandemir et al. [15], [16] presented a linear algebra framework to automatically determine the optimal data layouts expressed by hyperplanes for each array reference in a program. Boudet et al. [18] proposed a method to solve the alignment problem by considering how to align and distribute data arrays while, at the same time, considering

- *W.-L. Chang is with the Department of Information Management, Southern Taiwan University of Technology, Tainan County, Taiwan 710, Republic of China.*
E-mail: changwl@{csie.ncku.edu.tw, mail.stut.edu.tw}.
- *J.-W. Huang and C.-P. Chu are with the Department of Computer Science and Information Engineering, National Cheng Kung University, Tainan, Taiwan 701, Republic of China.*
E-mail: {cwhuang, chucp}@csie.ncku.edu.tw.

Manuscript received 17 Oct. 2001; revised 21 Nov. 2002; accepted 19 June 2003.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 115212.

how to preserve parallelism for a given program. Bau et al. [7] proposed an alignment technique to align arrays referenced using linear subscripts with one loop index variable in a communication-free manner. Chu et al. [17] and Chang et al. [24] presented communication-free alignment methods to align the arrays referenced using linear subscripts with two and three loop index variables or quadratic subscripts with the $(aI^2 + bI + d)$ or $(aI^2 + bI + cJ + d)$ patterns [5]. However, for the arrays referenced using linear subscripts or quadratic subscripts with multiple index variables (more than three loop index variables), these alignment methods cannot be applied. Generally, array alignment constraints are induced by the characteristics of the computations in which the arrays are referenced. These array alignment constraints can be expressed mathematically through a set of algebraic formulation, or can be modeled with a weighted graph problem.

In this paper, we propose two alignment techniques to properly map the loop iteration space that implies the computation instances, and the array elements which are respectively referenced using linear subscripts or quadratic subscripts with multiple loop index variables, onto the virtual processors so that no communication cost for data accesses is produced. Our alignment techniques, based on elementary linear algebra, reduce the computations and array elements mapping problem into the problem of determining a null space basis for a matrix. By simplifying solving the null space basis, the proposed techniques can readily determine the desired mapping functions.

The rest of this paper is organized as follows: In Section 2, the primary data alignment notion is introduced. In Section 3, the theoretical explanations and practical applications of our data alignment techniques are presented. In Section 4, the experimental results to show that our techniques can improve the execution time of the subroutines in Vector Loop and TRFD of the Perfect Benchmarks [4], [12] are offered. Finally, a brief conclusion is drawn in Section 5.

2 Preliminary Data Alignment Notion

In terms of linear algebra, the entire alignment framework, in general, includes three main steps [7]. The first step is to determine the constraints on the computation and data mapping. This means that the data accesses in a program are inspected and formulated as a system of equations in which the unknowns can be utilized to determine the virtual processors for the computations and data to be mapped onto. Each equation in the system is indeed a constraint on the computation and data mapping. Any solution to the system determines a so-called communication-free alignment that enables the needed data elements for a processor to perform a computation to be mapped onto its local memory so that no communication cost owing to data accesses occurs and, thus, optimizes the data locality in a program. Various data access patterns, such as array subscript patterns, will result in unlike equation systems. Hence, the alignment framework needs to take data access patterns into account to achieve a communication-free alignment. Intuitively, mapping all of the computations and data in a program onto a single processor is the trivial communication-free alignment—this, however, implies no parallelism. The trivial solution quite possibly is the only communication-free alignment if the system of equations is constrained too far. Accordingly, the second step of the framework is to identify the constraints (or equations) that need to be intentionally left unsatisfied to retain parallelism in the computations. Allowing unsatisfied constraints will introduce communication costs. Thus, the constraints left unsatisfied should be those that result in as little communication as possible. Finally, the remaining constraints are solved to determine the computation and data mapping functions. In terms of linear algebra, solving the remaining constraints is identical to determining the null space basis for a matrix. Eventually, based on the alignment result, the programmer needs to provide codes ahead of the nested loops for replicating or broadcasting data onto the processors so that no further communication is needed within the nested loops.

3 The Proposed Alignment Techniques

Linear expressions are the most common subscript patterns for referenced arrays. Petersen and Padua [9] indicated that there are 6,503 nonlinear cases in the analyzed Perfect Benchmarks, which were obtained in counting the number of feasible directions of the potential dependences. With our counting criteria for the number of quadratic cases for data alignment, which is the number of the nested loops containing arrays with quadratic subscripts, we surveyed and found that the quadratic cases approximately account for 60 percent of all the nested loops that contain nonlinear array references in TRFD of the Perfect Benchmarks after induction variable substitution and/or scalar expansion transformations. These results imply that the number of the arrays with quadratic subscript might attain to certain extent. However, the data alignment for the arrays referenced using quadratic subscripts was scarcely discussed before. In this section, our communication-free alignment techniques for aligning the arrays referenced using generalized linear or quadratic subscripts (i.e., linear or quadratic subscripts with multiple loop index variables) are discussed. To avoid falling into complications, the illustration for our techniques is restricted to formulating and solving the equation system. The discussion related to unsatisfied constraints is not considered here. The features and time complexity analysis of our techniques are presented as well.

3.1 Arrays Referenced Using Linear Subscripts

Assume that there exist s statements containing q arrays, each with one or more (say m) dimensions, referenced using linear subscripts enclosed with a general n nested do loop. To align data elements for multidimensional arrays, a general approach is to employ one dimension among others for each array as the alignment basis. We consider the data alignment for multidimensional arrays as the data alignment simply for the adopted dimension of the arrays in the following discussions. Suppose a reference function for the adopted dimension of an array A_e for $1 \leq e \leq q$ in this common loop is $R_{A_e} = a_{e,1}I_1 + a_{e,2}I_2 + \ldots + a_{e,n}I_n + a_{e,0}$, where $I_1, I_2, \ldots, I_n$ are index variables of the general nested loop, and $a_{e,1}, a_{e,2}, \ldots, a_{e,n}$ are integer coefficients and $a_{e,0}$ is an integer constant. For an iteration vector $\mathbf{i}$ ($\mathbf{i} = [i_1 i_2 \ldots i_n]^T$, i_u is an index value of I_u for $1 \leq u \leq n$ and T is the transposition operation) in the iteration space of this general n nested do loop, the alignment constraints require the processor performing iteration $\mathbf{i}$, which stands for a

```
DO I= 1, 100
    DO J= 1, 100
        S1: B(I+J-1)=2*A(2I+3J+1)-1
        S2: D(I+J+5)=B(I+J-1)
    ENDDO
ENDDO
```

Fig. 1. Arrays referenced using linear subscripts.

computation instance, to own $A_e(R_{A_e})$. With our techniques, if there exist two or more *distinct* references (either read or write) to an array, each of the distinct references will be selected as the alignment constraints respectively for this array without considering their data dependences. For instance, there are two statements containing three different one-dimensional arrays (i.e., $q = 3$ and $m = 1$) in the example in Fig. 1. For an iteration vector $\mathbf{i}$ ($\mathbf{i} = [i_j]^T$), the alignment constraint demands that the processor performing iteration $\mathbf{i}$ must own $A(R_A)$, $B(R_B)$, and $D(R_D)$.

Below, we first give the mathematical derivation of our alignment methods. Suppose that C is the computation mapping function to map the loop iteration space onto virtual processors and D_{A_e} is the data mapping function to map the array elements of A_e onto virtual processors. The alignment problem can be described as: Find C and D_{A_e} such that $\forall\ \mathbf{i} \in$ iteration space of this loop:

$$C(\mathbf{i}) = D_{A_e}(R_{A_e}). \tag{1}$$

To map the computations and array elements in a communication-free manner, our alignment technique considers the array subscript patterns that are generalized linear subscripts here. Hence, C and D_{A_e} will be formulated using our technique in linear form as follows:

$$R_{A_e} = R'_{A_e}\mathbf{i} + a_{e,0}(R'_{A_e} = [a_{e,1}a_{e,2}\ldots a_{e,n}]), \tag{2}$$

$$C(\mathbf{i}) = C'\mathbf{i} + c_0, \tag{3}$$

and

$$D_{A_e}(R_{A_e}) = D'_{A_e}(R_{A_e}) + d_0 = D'_{A_e}(R'_{A_e}\mathbf{i} + a_{e,0}) + d_0. \tag{4}$$

From (3) and (4), (1) can be converted into the following equation:

$$C'\mathbf{i} + c_0 = D'_{A_e}(R'_{A_e}\mathbf{i} + a_{e,0}) + d_0. \tag{5}$$

Without loss of generality, (5) can be reduced to (6):

$$[\,C' \quad c_0\,]\begin{bmatrix}\mathbf{i}\\1\end{bmatrix} = [\,D'_{A_e} \quad d_0\,]\begin{bmatrix}R'_{A_e} & a_{e,0}\\0 & 1\end{bmatrix}\begin{bmatrix}\mathbf{i}\\1\end{bmatrix}. \tag{6}$$

Let

$$C = [\,C' \quad c_0\,], D_{A_e} = [\,D'_{A_e} \quad d_0\,], F_{A_e} = \begin{bmatrix}R'_{A_e} & a_{e,0}\\0 & 1\end{bmatrix}$$

and $\mathbf{i}' = \begin{bmatrix}\mathbf{i}\\1\end{bmatrix}$, (6) can be transformed into the following equation:

$$C\mathbf{i}' = D_{A_e}F_{A_e}\mathbf{i}'. \tag{7}$$

As a result, to determine C and D_{A_e} is to solve the equation $C = D_{A_e}F_{A_e}$ (or $C - D_{A_e}F_{A_e} = \mathbf{0}$) provided that $\mathbf{i}'$ can be cancelled. Such an equation can be rewritten, without loss of generality, in block matrix form [7] as follows:

$$[\,C \quad D_{A_e}\,]\begin{bmatrix}\mathbf{I}\\-F_{A_e}\end{bmatrix} = \mathbf{0}. \tag{8}$$

Here, $\mathbf{I}$ is an identity matrix, and $\mathbf{0}$ (zero matrix), C, D_{A_e} and F_{A_e} are square matrices with the same size as $\mathbf{I}$. By expressing (8) in the form of $UV = \mathbf{0}$ and determining a null space basis for V^T, the alignment problem is thus reduced to the standard linear problem of determining a null space basis for a matrix.

Clearly, to construct C, D_{A_e}, and F_{A_e} as square matrices for the nested do loops with different numbers of loop index variables, (3) and (4) need to be adjusted. For example, suppose that there is only one index loop variable I_1 for this do loop; that is, $R_{A_e} = a_{e,1}I_1 + a_{e,0}$, then our alignment technique originally formulates $C(\mathbf{i})$ and $D_{A_e}(R_{A_e})$ as follows: $C(\mathbf{i}) = c_1 i_1 + c_0$ and $D_{A_e}(R_{A_e}) = d_1(a_{e,1}i_1 + a_{e,0}) + d_0$. To construct C and D_{A_e} as square matrices, the above equations are adjusted as follows: $C(\mathbf{i}) = (c_{1,1} + c_{2,1})i_1 + (c_{1,2} + c_{2,2})$ and

$$D_{A_e}(R_{A_e}) = (d_{1,1} + d_{2,1})(a_{e,1}i_1 + a_{e,0}) + (d_{1,2} + d_{2,2}).$$

Using our technique, the above equations are reduced to the following equation:

$$\begin{bmatrix}c_{1,1} & c_{1,2}\\c_{2,1} & c_{2,2}\end{bmatrix}\begin{bmatrix}i_1\\1\end{bmatrix} = \begin{bmatrix}d_{1,1} & d_{1,2}\\d_{2,1} & d_{2,2}\end{bmatrix}\begin{bmatrix}a_{e,1} & a_{e,0}\\0 & 1\end{bmatrix}\begin{bmatrix}i_1\\1\end{bmatrix}.$$

This makes C and D_{A_e} 2×2 square matrices. However, for a nested do loop with two index loop variables I_1 and I_2; that is, $R_{A_e} = a_{e,1}I_1 + a_{e,2}I_2 + a_{e,0}$, our alignment technique originally formulates $C(\mathbf{i})$ and $D_{A_e}(R_{A_e})$ as follows: $C(\mathbf{i}) = c_1 i_1 + c_2 i_2 + c_0$ and $D_{A_e}(R_{A_e}) = d_1(a_{e,1}i_1 + a_{e,2}i_2 + a_{e,0}) + d_0$ However, in addition to constructing C and D_{A_e} as square matrices, our alignment technique also considers enabling C and D_{A_e} to be easily determined by constructing F_{A_e} in a form that allows the Gaussian elimination for echelon reduction to be carried out effortlessly to simplify the calculation for the aforementioned null space basis. The above equations are thus adjusted as follows:

$$\begin{aligned}C(\mathbf{i}) =& (c_{1,1} + c_{2,1} + c_{3,1})i_1 + (c_{1,2} + c_{2,2} + c_{3,2})i_2 +\\ & (c_{1,3} + c_{2,3} + c_{3,3})\end{aligned}$$

and

$$\begin{aligned}D_{A_e}(R_{A_e}) =& (d_{1,1} + d_{2,1} + d_{3,1})(a_{e,1}i_1 + a_{e,2}i_2 + a_{e,0}) +\\ & (d_{1,2} + d_{2,2} + d_{3,2}) + (d_{1,3} + d_{2,3} + d_{3,3})(i_1 + i_2 + 1).\end{aligned}$$

Using our technique, the above equations are reduced to the following equation:

$$\begin{bmatrix}c_{1,1} & c_{1,2} & c_{1,3}\\c_{2,1} & c_{2,2} & c_{2,3}\\c_{3,1} & c_{3,2} & c_{3,3}\end{bmatrix}\begin{bmatrix}i_1\\i_2\\1\end{bmatrix} =$$

$$\begin{bmatrix}d_{1,1} & d_{1,2} & d_{1,3}\\d_{2,1} & d_{2,2} & d_{2,3}\\d_{3,1} & d_{3,2} & d_{3,3}\end{bmatrix}\begin{bmatrix}a_{e,1} & a_{e,2} & a_{e,0}\\0 & 0 & 1\\1 & 1 & 1\end{bmatrix}\begin{bmatrix}i_1\\i_2\\1\end{bmatrix}.$$

This makes C and D_{A_e} 3×3 square matrices. Continuing with this notion, the alignment constraint for the iteration space of this n nested do loop can be formally expressed, using our technique, as:

$$C\mathbf{i}' = \begin{bmatrix} c_{1,1} \\ c_{2,1} \\ \vdots \\ c_{n+1,1} \end{bmatrix} [i_1] + \begin{bmatrix} c_{1,2} \\ c_{2,2} \\ \vdots \\ c_{n+1,2} \end{bmatrix} [i_2] + \ldots + \begin{bmatrix} c_{1,n} \\ c_{2,n} \\ \vdots \\ c_{n+1,n} \end{bmatrix} [i_n] + \begin{bmatrix} c_{1,n+1} \\ c_{2,n+1} \\ \vdots \\ c_{n+1,n+1} \end{bmatrix}$$
$$= \begin{bmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,n+1} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,n+1} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n+1,1} & c_{n+1,2} & \cdots & c_{n+1,n+1} \end{bmatrix} \begin{bmatrix} i_1 \\ \vdots \\ i_n \\ 1 \end{bmatrix}.$$

The alignment constraint for an array $A_e, 1 \leq e \leq q$ in the general loop can be represented as:

$$D_{A_e}F_{A_e}\mathbf{i}' = \begin{bmatrix} d_{1,1} \\ d_{2,1} \\ \vdots \\ d_{n+1,1} \end{bmatrix} [a_{e,1}i_1 + \ldots + a_{e,n}i_n + a_{e,0}] + \begin{bmatrix} d_{1,2} \\ d_{2,2} \\ \vdots \\ d_{n+1,2} \end{bmatrix} + \begin{bmatrix} d_{1,3} \\ d_{2,3} \\ \vdots \\ d_{n+1,3} \end{bmatrix} [i_1 + \ldots + i_n + 1] + \ldots + \begin{bmatrix} d_{1,n+1} \\ d_{2,n+1} \\ \vdots \\ d_{n+1,n+1} \end{bmatrix}$$
$$[i_1 + \ldots + i_n + 1] = \begin{bmatrix} d_{1,1} & \cdots & d_{1,n+1} \\ \vdots & \ddots & \vdots \\ d_{n+1,1} & \cdots & d_{n+1,n+1} \end{bmatrix}$$
$$\begin{bmatrix} a_{e,1} & a_{e,2} & \ldots & a_{e,n} & a_{e,0} \\ 0 & 0 & \ldots & 0 & 1 \\ 1 & 1 & \ldots & 1 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & 1 & \ldots & 1 & 1 \end{bmatrix} \begin{bmatrix} i_1 \\ \vdots \\ i_n \\ 1 \end{bmatrix}.$$

Therefore, the alignment problem can be restated as: Find C and D_{A_e} such that $\forall\, \mathbf{i} \in$ iteration space of this loop: $C\mathbf{i}' = D_{A_e}F_{A_e}\mathbf{i}'$. Here, $\mathbf{i}' = [\mathbf{i}\ 1]^T$, as mentioned. The above equation can be reduced to (8) to determine C and D_{A_e}, as described. This requires the column vector $\mathbf{i}'$ on both sides of the equation to be cancelled to make $(C - D_{A_e}F_{A_e})$ equal to $\mathbf{0}$ for any $\mathbf{i}'$. To do this, we need the following lemma.

Lemma 1. *Let Q_i be a $q \times 1$ matrix for $1 \leq i \leq n$, $\mathbf{t}$ a q-elements column vector, $\mathbf{0}$ a q-elements zero vector, and x_i a scalar variable for $1 \leq i \leq n$. Then,*

$$\forall x_i [Q_1 \ \ldots \ Q_n \ \mathbf{t}] \begin{bmatrix} x_1 \\ \vdots \\ x_n \\ 1 \end{bmatrix} = \mathbf{0} \Leftrightarrow Q_i = \mathbf{0} \text{ for } 1 \leq i \leq n \text{ and } \mathbf{t} = \mathbf{0}.$$

Proof.

$$\forall x_i [Q_1 \ \ldots \ Q_n \ \mathbf{t}] \begin{bmatrix} x_1 \\ \vdots \\ x_n \\ 1 \end{bmatrix} = \mathbf{0} \Leftrightarrow \forall x_i \begin{bmatrix} q_{1,1} & q_{2,1} & \cdots & q_{n,1} & t_1 \\ q_{1,2} & q_{2,2} & \cdots & q_{n,2} & t_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ q_{1,q} & q_{2,q} & \cdots & q_{n,q} & t_q \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \\ 1 \end{bmatrix} = \mathbf{0}$$

$$\Leftrightarrow \quad \forall x_i \begin{bmatrix} x_1q_{1,1} + x_2q_{2,1} + \ldots + x_nq_{n,1} + t_1 \\ x_1q_{1,2} + x_2q_{2,2} + \ldots + x_nq_{n,2} + t_2 \\ \vdots \\ x_1q_{1,q} + x_2q_{2,q} + \ldots + x_nq_{n,q} + t_q \end{bmatrix} = \mathbf{0}$$

$$\Leftrightarrow \quad \forall\ x_i x_1Q_1 + x_2Q_2 + \ldots + x_nQ_n + \mathbf{t} = \mathbf{0}.$$

We now show that

$$\forall\ x_i x_1Q_1 + x_2Q_2 + \ldots + x_nQ_n + \mathbf{t} = \mathbf{0} \Leftrightarrow$$
$$Q_1 = Q_2 = \ldots = Q_n = \mathbf{t} = \mathbf{0}.$$

$\Rightarrow$) Assume that there exists some Q_i's $\neq \mathbf{0}$ (e.g., Q_p, Q_r, and $Q_s \neq \mathbf{0}$) and $\mathbf{t} \neq \mathbf{0}$ and some x_i's $\neq 0$ (e.g., x_p, x_r, and $x_s \neq 0$) such that

$$x_pQ_p + x_rQ_r + x_sQ_s + \mathbf{t} = \mathbf{0}.$$

Since all x_is can be any value, we have

$$(x_p + 1)Q_p + x_rQ_r + x_sQ_s + \mathbf{t} = \mathbf{0}.$$

That is, $(x_pQ_p + x_rQ_r + x_sQ_s + \mathbf{t}) + Q_p = \mathbf{0}$. This implies $Q_p = \mathbf{0}$, which contradicts the assumption. Similarly, we have $Q_r = Q_s = \mathbf{0}$. This makes $\mathbf{t} = \mathbf{0}$. Therefore, $Q_1 = Q_2 = \ldots = Q_n = \mathbf{t} = \mathbf{0}$.

$\Leftarrow$) The (if part) is trivial. □

From Lemma 1, (7) can indeed be rewritten as:

$$C = D_{A_e}F_{A_e}. \tag{9}$$

For $1 \leq e \leq q$, the equation system of (9) can be converted into the following matrix equation (10):

$$[C\ D_{A_1} \ \ldots \ D_{A_q}] \begin{bmatrix} \mathbf{I} & \mathbf{I} & \ldots & \mathbf{I} \\ -F_{A_1} & \mathbf{0} & \ldots & \mathbf{0} \\ \mathbf{0} & \ddots & & \vdots \\ \vdots & & \ddots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \ldots & -F_{A_q} \end{bmatrix} = [\mathbf{0} \ \ldots \ \mathbf{0}].$$

Here, $\mathbf{I}$ is an $(n+1)\times(n+1)$ identify matrix, $\mathbf{0}$ is an $(n+1)\times(n+1)$ zero matrix, and $[\mathbf{0}\ \dots\ \mathbf{0}]$ is an $(n+1)\times((n+1)\times q)$ zero matrix.

To solve the matrix equation $[U]_{s\times m}[V]_{m\times n}=[0]_{s\times n}$ in which $[U]_{s\times m}$ is unknown and $[V]_{m\times n}$ is known, we can first transform V into a "rank-revealing" form by performing the required rank preserving operations—elementary row and column operations. The notion behind this is to get a matrix into a form in which its rank can be determined by inspection [1], [7]. One way to achieve this is to perform integer preserving Gaussian elimination [1], [2], whereby matrix rows or columns are systematically manipulated by elementary row or column operations to yield a matrix in echelon form, to enable us to obtain the following factorization (suppose that $V\in Z^{m\times n}$ and $rank(V)=r$):

$$[H]_{m\times m}[V]_{m\times n}[P]_{n\times n}=\begin{bmatrix} R_{1,1} & R_{1,2} \\ 0 & 0 \end{bmatrix}_{m\times n}.$$

Here, H is an $m\times m$ invertible matrix representing the row operations, P is an $n\times n$ unimodular matrix representing the column operations, and $R_{1,1}$ is an $r\times r$ upper triangular invertible matrix. It is a property of this factorization that the transposition of the last m-r rows of H spans the null space of V^T. Thus, we can then obtain the solution for $[U]_{s\times m}$ as follows: $U=H(r+1:m,\ 1:m)$. This means that only H, the composition of row operations, needs to be determined during the elimination.

We now go back to the example in Fig. 1 to illustrate our points. In this case, there are two references to the same array B. The reference to array B in S_1 updates the values of array B elements. The reference to array B in S_2 reads the values of array B elements. Because both references to array B use the same data access function, one of these references is selected as the alignment constraint for array B. Thus, the alignment constraint for the iteration space of this nested loop can be formally expressed as:

$$C\mathbf{i}'=\begin{bmatrix} c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,1} & c_{3,2} & c_{3,3} \end{bmatrix}\begin{bmatrix} i \\ j \\ 1 \end{bmatrix}.$$

The alignment constraints for the three arrays A, B, and D, can also be, respectively, represented as:

$$D_AF_A\mathbf{i}'=\begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} \\ x_{2,1} & x_{2,2} & x_{2,3} \\ x_{3,1} & x_{3,2} & x_{3,3} \end{bmatrix}\begin{bmatrix} 2 & 3 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}\begin{bmatrix} i \\ j \\ 1 \end{bmatrix},$$

$$D_BF_B\mathbf{i}'=\begin{bmatrix} y_{1,1} & y_{1,2} & y_{1,3} \\ y_{2,1} & y_{2,2} & y_{2,3} \\ y_{3,1} & y_{3,2} & y_{3,3} \end{bmatrix}\begin{bmatrix} 1 & 1 & -1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}\begin{bmatrix} i \\ j \\ 1 \end{bmatrix},\ \text{and}$$

$$D_DF_D\mathbf{i}'=\begin{bmatrix} z_{1,1} & z_{1,2} & z_{1,3} \\ z_{2,1} & z_{2,2} & z_{2,3} \\ z_{3,1} & z_{3,2} & z_{3,3} \end{bmatrix}\begin{bmatrix} 1 & 1 & 5 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}\begin{bmatrix} i \\ j \\ 1 \end{bmatrix}.$$

Therefore, the alignment problem can be described as follows: Find C, D_A, D_B, and D_D such that $\forall\,(i,j)\in$ iteration space of this loop:

$$\begin{cases} C=D_AF_A \\ C=D_BF_B\,. \\ C=D_DF_D \end{cases} \tag{11}$$

The system of equations (11) can be converted into the following matrix equation:

$$[C\quad D_A\quad D_B\quad D_D]\begin{bmatrix} \mathbf{I} & \mathbf{I} & \mathbf{I} \\ -F_A & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & -F_B & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & -F_D \end{bmatrix}=[\mathbf{0}\quad\mathbf{0}\quad\mathbf{0}]. \tag{12}$$

Here, $\mathbf{I}$ is a 3×3 identity matrix and $\mathbf{0}$ is a 3×3 zero matrix. According to the method described above, a solution matrix of (12) is:

$$\begin{bmatrix} C & D_A & D_B & D_D \end{bmatrix}=\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 1 & 1 & 2 & 0 & 1 & -4 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & -2 & 1 & -1 & 4 & 1 \end{bmatrix}.$$

This gives us:

$$C=\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix},\ D_A=\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix},\ D_B=\begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 0 \\ -1 & -2 & 1 \end{bmatrix},$$

$$\text{and } D_D=\begin{bmatrix} 1 & -4 & 0 \\ 0 & 1 & 0 \\ -1 & 4 & 1 \end{bmatrix}.$$

Therefore, we can obtain the computation and data mapping as follows:

$$C\mathbf{i}'=\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} i \\ j \\ 1 \end{bmatrix}=\begin{bmatrix} i+j+1 \\ 1 \\ 0 \end{bmatrix},$$

$$D_AF_A\mathbf{i}'=\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} 2 & 3 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}\begin{bmatrix} i \\ j \\ 1 \end{bmatrix}=\begin{bmatrix} i+j+1 \\ 1 \\ 0 \end{bmatrix},$$

$$D_BF_B\mathbf{i}'=\begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 0 \\ -1 & -2 & 1 \end{bmatrix}\begin{bmatrix} 1 & 1 & -1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}\begin{bmatrix} i \\ j \\ 1 \end{bmatrix}=\begin{bmatrix} i+j+1 \\ 1 \\ 0 \end{bmatrix},\ \text{and}$$

$$D_DF_D\mathbf{i}'=\begin{bmatrix} 1 & -4 & 0 \\ 0 & 1 & 0 \\ -1 & 4 & 1 \end{bmatrix}\begin{bmatrix} 1 & 1 & 5 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}\begin{bmatrix} i \\ j \\ 1 \end{bmatrix}=\begin{bmatrix} i+j+1 \\ 1 \\ 0 \end{bmatrix}.$$

Hence, using our alignment, iteration $(i,\ j)$ is mapped onto virtual processor $i+j+1$ and the corresponding array elements of A, B, and D are mapped onto the same virtual processor. In Fortran D, the **Align** statement can be applied to map the arrays onto the virtual processors. The array elements mapped onto the same virtual processor are automatically aligned with one another. We employ **Align** statements in this case to describe the alignment relation for the array elements of A, B, and D. They are represented as follows:

$$\begin{aligned} &\text{Align } A(2I+3J+1) && \text{with } T(I+J+1), \\ &\text{Align } B(I+J-1)_{\text{WRITE}} && \text{with } T(I+J+1), \\ &\text{Align } B(I+J-1)_{\text{READ}} && \text{with } T(I+J+1),\ \text{and} \\ &\text{Align } D(I+J+5) && \text{with } T(I+J+1). \end{aligned}$$

Here, the virtual processors are supposed to be organized as a one-dimensional template T, $B(I+J-1)_{\text{WRITE}}$ represents the reference to array B in S_1 and $B(I+J-1)_{\text{READ}}$ represents the reference to array B in S_2.

There exist loop-carried output-dependences for array B in S_1. From S_1 to S_2, there exist loop-carried true-dependences

```
L1: DO I1 = 1, N
    L2: DO I2 = 1, I1
            IA=I1*(I1 −1)/ 2 + I2
            IB=I2*(I2 −1)/ 2 + I1
    S:      A(IA) = B(IB)
        ENDDO
    ENDDO
```

Fig. 2. Arrays referenced using quadratic subscripts.

for array B. In S_2, there exist loop-carried output-dependences for array D. The alignment function for the one-dimensional template T and the reference functions for the written arrays, B and D, are in common form. Therefore, the iterations updating the same elements of array B or array D are mapped onto the same template element (see the proof for Theorem 1 in the Appendix). That is, the loop-carried output-dependences in this nested loop can be removed. Likewise, because those iterations respectively updating and reading the same elements of array B are mapped onto the same template element, the loop-carried true-dependences for array B from S_1 to S_2 can also be removed. The array A in S_1 is a read-only variable. Though the mapping of array A is not one-to-one, replicating multiple copies of the data elements of array A onto different processors will not alter the correctness of execution. Owing to that the required data elements (the written and read data elements of array B, the written data element of array D, and the read data element of array A) for a computation are mapped onto the same template element, this nested loop can be executed in parallel without inter-processor communication.

3.2 Arrays Referenced Using Quadratic Subscripts

Assume that there exist s statements containing q arrays of m-dimensions referenced using quadratic subscripts enclosed with a general n nested do loop. Suppose that a reference function for the adopted dimension of an array A_e for $1 \leq e \leq q$ in this general loop is

$$R_{A_e} = a_{e,1}I_1^2 + a_{e,2}I_2^2 + \ldots + a_{e,n}I_n^2 + b_{e,1}I_1 + b_{e,2}I_2 + \ldots + b_{e,n}I_n + f_e,$$

where $I_1, I_2, \ldots, I_n$ are index variables of the general loop and $a_{e,1}, a_{e,2}, \ldots, a_{e,n}, b_{e,1}, b_{e,2}, \ldots, b_{e,n}$ are coefficients, which, in general, are integers or fractions in the quadratic case, and f_e is an integer constant.

For instance, two different one-dimensional arrays (i.e., $q = 2$ and $m = 1$), referenced using quadratic subscripts, are enclosed with a depth-two nested loop in the example in Fig. 2. For an iteration vector $\mathbf{i}$ ($\mathbf{i} = [i_1\ i_2]^T$), the alignment constraints require the processor performing iteration $\mathbf{i}$ to own $A(R_A)$ and $B(R_B)$.

The primary notion of our technique to align arrays referenced using quadratic subscripts is the same as the previous linear technique, except that C and D_{A_e} are formulated in quadratic patterns. However, in dealing with the quadratic cases, more subtle but nontrivial adjustments on (3) and (4) are needed to construct C, D_{A_e}, and F_{A_e} as square matrices for the nested do loop with different numbers of loop index variables. Similar to the linear case, the alignment problem can be described as: Find C and D_{A_e} such that $\forall\ \mathbf{i} \in$ iteration space of this loop: $C\mathbf{i}' = D_{A_e}F_{A_e}\mathbf{i}'$. In this case, $\mathbf{i} = [i_1\ i_2\ \ldots\ i_n]^T$ and $\mathbf{i}' = [i_1^2\ i_2^2\ \ldots i_n^2\ i_1\ i_2\ \ldots\ i_n\ 1]^T$. The alignment constraint for the iteration space of this general loop can be formally expressed, using our alignment technique, as:

$$C\mathbf{i}' = \begin{bmatrix} c_{1,1} \\ c_{2,1} \\ \vdots \\ c_{2n+1,1} \end{bmatrix} [i_1^2] + \ldots + \begin{bmatrix} c_{1,n} \\ c_{2,n} \\ \vdots \\ c_{2n+1,n} \end{bmatrix} [i_n^2] + \begin{bmatrix} c_{1,n+1} \\ c_{2,n+1} \\ \vdots \\ c_{2n+1,n+1} \end{bmatrix} [i_1] + \ldots + \begin{bmatrix} c_{1,2n} \\ c_{2,2n} \\ \vdots \\ c_{2n+1,2n} \end{bmatrix} [i_n] + \begin{bmatrix} c_{1,2n+1} \\ c_{2,2n+1} \\ \vdots \\ c_{2n+1,2n+1} \end{bmatrix}$$

$$= \begin{bmatrix} c_{1,1} & \ldots & c_{1,n} & c_{1,n+1} & \ldots & c_{1,2n} & c_{1,2n+1} \\ c_{2,1} & \ldots & c_{2,n} & c_{2,n+1} & \ldots & c_{2,2n} & c_{2,2n+1} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ c_{2n+1,1} & \ldots & c_{2n+1,n} & c_{2n+1,n+1} & \ldots & c_{2n+1,2n} & c_{2n+1,2n+1} \end{bmatrix} \begin{bmatrix} i_1^2 \\ \vdots \\ i_n^2 \\ i_1 \\ \vdots \\ i_n \\ 1 \end{bmatrix}.$$

The alignment constraint for an array $A_e, 1 \leq e \leq q$, in the general loop can be represented as:

$$D_{A_e}F_{A_e}\mathbf{i}' = \begin{bmatrix} d_{1,1} \\ d_{2,1} \\ \vdots \\ d_{2n+1,1} \end{bmatrix} [a_{e,1}i_1^2 + \ldots + a_{e,n}i_n^2 + b_{e,1}i_1 + \ldots + b_{e,n}i_n + f_e] + \begin{bmatrix} d_{1,2} \\ d_{2,2} \\ \vdots \\ d_{2n+1,2} \end{bmatrix} + \begin{bmatrix} d_{1,3} \\ d_{2,3} \\ \vdots \\ d_{2n+1,3} \end{bmatrix} [i_1^2 + \ldots + i_n^2 + i_1 + \ldots + i_n + 1] + \ldots + \begin{bmatrix} d_{1,2n+1} \\ d_{2,2n+1} \\ \vdots \\ d_{2n+1,2n+1} \end{bmatrix} [i_1^2 + \ldots + i_n^2 + i_1 + \ldots + i_n + 1] =$$

$$\begin{bmatrix} d_{1,1} & \ldots & d_{1,2n+1} \\ d_{2,1} & \ldots & d_{2,2n+1} \\ \vdots & \ddots & \vdots \\ d_{2n+1,1} & \ldots & d_{2n+1,2n+1} \end{bmatrix} \begin{bmatrix} a_{e,1} & \ldots & b_{e,n} & f_e \\ 0 & \ldots & 0 & 1 \\ 1 & \ldots & 1 & 1 \\ \vdots & \ddots & \vdots & \vdots \\ 1 & \ldots & 1 & 1 \end{bmatrix} \begin{bmatrix} i_1^2 \\ \vdots \\ i_n^2 \\ i_1 \\ \vdots \\ i_n \\ 1 \end{bmatrix}.$$

Similar to the discussion in the previous section, the column vector $\mathbf{i}'$ on both sides of equation, $C\mathbf{i}' = D_{A_e}F_{A_e}\mathbf{i}'$, is required to be cancelled for any $\mathbf{i}'$. To do this, we need the following lemma.

Lemma 2. *Let Q_i be a $q \times 1$ matrix for $1 \leq i \leq 2n$, $\mathbf{t}$ a q-elements column vector, $\mathbf{0}$ a q-elements zero vector, and x_i a scalar variable for $1 \leq i \leq 2n$. Then,*

$$\forall x_i \; [Q_1 \; \dots \; Q_n \; Q_{n+1} \; \dots \; Q_{2n} \; \mathbf{t}] \begin{bmatrix} x_1^2 \\ \vdots \\ x_n^2 \\ x_1 \\ \vdots \\ x_n \\ 1 \end{bmatrix} = \mathbf{0} \Leftrightarrow Q_i = \mathbf{0}$$

for $1 \leq i \leq 2n,$ and $\mathbf{t} = \mathbf{0}$.

Proof.

$$\forall x_i \; [Q_1 \; \dots \; Q_n \; Q_{n+1} \dots Q_{2n} \; t] \begin{bmatrix} x_1^2 \\ \vdots \\ x_n^2 \\ x_1 \\ \vdots \\ x_n \\ 1 \end{bmatrix} = \mathbf{0} \Leftrightarrow$$

$$\forall x_i \begin{bmatrix} q_{1,1} & \cdots & q_{n,1} & q_{n+1,1} & \cdots & q_{2n,1} & t_1 \\ q_{1,2} & \cdots & q_{n,2} & q_{n+1,2} & \cdots & q_{2n,2} & t_2 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ q_{1,q} & \cdots & q_{n,q} & q_{n+1,q} & \cdots & q_{2n,q} & t_q \end{bmatrix} \begin{bmatrix} x_1^2 \\ \vdots \\ x_n^2 \\ x_1 \\ \vdots \\ x_n \\ 1 \end{bmatrix} = \mathbf{0} \Leftrightarrow$$

$$\forall x_i \begin{bmatrix} x_1^2 q_{1,1} + \dots + x_n^2 q_{n,1} + x_1 q_{n+1,1} + \dots + x_n q_{2n,1} + t_1 \\ x_1^2 q_{1,2} + \dots + x_n^2 q_{n,2} + x_1 q_{n+1,2} + \dots + x_n q_{2n,2} + t_2 \\ \vdots \\ x_1^2 q_{1,q} + \dots + x_n^2 q_{n,q} + x_1 q_{n+1,q} + \dots + x_n q_{2n,q} + t_q \end{bmatrix} = \mathbf{0}$$

$$\Leftrightarrow \quad \forall x_i \; x_1^2 Q_1 + \dots + x_n^2 Q_n + x_1 Q_{n+1} + \dots + x_n Q_{2n} + \mathbf{t} = \mathbf{0}.$$

We now show that

$\forall x_i \; x_1^2 Q_1 + \dots + x_n^2 Q_n + x_1 Q_{n+1} + \dots + x_n Q_{2n} + \mathbf{t} = \mathbf{0} \;\; \Leftrightarrow$
$Q_1 = \dots = Q_n = Q_{n+1} = \dots = Q_{2n} = \mathbf{t} = \mathbf{0}$.

$\Rightarrow$) Assume that there exists some Q_is $\neq \mathbf{0}$ (e.g., Q_p, Q_r, Q_s, and $Q_t \neq \mathbf{0}$) and $\mathbf{t} \neq \mathbf{0}$ and some x_is $\neq 0$ (e.g., x_p, x_r, x_s, and $x_t \neq 0$) such that $x_p^2 Q_p + x_r^2 Q_r + x_s Q_s + x_t Q_t + \mathbf{t} = \mathbf{0}$. Since all x_is can be any value, we have $(x_p + 1)^2 Q_p + x_r^2 Q_r + x_s Q_s + x_t Q_t + \mathbf{t} = \mathbf{0}$. That is,

$$(x_p^2 Q_p + x_r^2 Q_r + x_s Q_s + x_t Q_t + \mathbf{t}) + (2x_p + 1) Q_p = \mathbf{0}.$$

This implies $Q_p = \mathbf{0}$, which contradicts the assumption. Similarly, we have $Q_r = \mathbf{0}$. This gives us $x_s Q_s + x_t Q_t + \mathbf{t} = \mathbf{0}$. From Lemma 1, we can obtain $Q_s = Q_t = \mathbf{t} = \mathbf{0}$. Therefore, $Q_1 = \dots = Q_n = Q_{n+1} = \dots = Q_{2n} = \mathbf{t} = \mathbf{0}$.
$\Leftarrow$) The (if part) is trivial. □

Again, similar to the linear equation, the equation system in this case can eventually be converted into the following matrix equation:

$$[C \quad D_{A_1} \quad \dots \quad D_{A_q}] \begin{bmatrix} \mathbf{I} & \dots & \mathbf{I} \\ -F_{A_1} & \dots & \mathbf{0} \\ \vdots & \ddots & \vdots \\ \mathbf{0} & \dots & -F_{A_q} \end{bmatrix} = [\mathbf{0} \quad \dots \quad \mathbf{0}].$$

Here, $\mathbf{I}$ is a $(2n+1) \times (2n+1)$ identity matrix, $\mathbf{0}$ is a $(2n+1) \times (2n+1)$ zero matrix, and $[\mathbf{0} \quad \dots \quad \mathbf{0}]$ is a $(2n+1) \times ((2n+1) \times q)$ zero matrix. Note that, in solving the above matrix equation, each row with fraction elements in each F_{A_e} can be first multiplied by a factor to make it have only integer elements.

In the example in Fig. 2, the alignment constraint for the iteration space of this loop can be formally expressed as:

$$C\mathbf{i}' = \begin{bmatrix} c_{1,1} & c_{1,2} & c_{1,3} & c_{1,4} & c_{1,5} \\ c_{2,1} & c_{2,2} & c_{2,3} & c_{2,4} & c_{2,5} \\ c_{3,1} & c_{3,2} & c_{3,3} & c_{3,4} & c_{3,5} \\ c_{4,1} & c_{4,2} & c_{4,3} & c_{4,4} & c_{4,5} \\ c_{5,1} & c_{5,2} & c_{5,3} & c_{5,4} & c_{5,5} \end{bmatrix} \begin{bmatrix} i_1^2 \\ i_2^2 \\ i_1 \\ i_2 \\ 1 \end{bmatrix}.$$

The alignment constraints for arrays A and B can be, respectively, represented as:

$$D_A F_A \mathbf{i}' = \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} & x_{1,5} \\ x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} & x_{2,5} \\ x_{3,1} & x_{3,2} & x_{3,3} & x_{3,4} & x_{3,5} \\ x_{4,1} & x_{4,2} & x_{4,3} & x_{4,4} & x_{4,5} \\ x_{5,1} & x_{5,2} & x_{5,3} & x_{5,4} & x_{5,5} \end{bmatrix} \begin{bmatrix} 1/2 & 0 & -1/2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} i_1^2 \\ i_2^2 \\ i_1 \\ i_2 \\ 1 \end{bmatrix}$$

and

$$D_B F_B \mathbf{i}' = \begin{bmatrix} y_{1,1} & y_{1,2} & y_{1,3} & y_{1,4} & y_{1,5} \\ y_{2,1} & y_{2,2} & y_{2,3} & y_{2,4} & y_{2,5} \\ y_{3,1} & y_{3,2} & y_{3,3} & y_{3,4} & y_{3,5} \\ y_{4,1} & y_{4,2} & y_{4,3} & y_{4,4} & y_{4,5} \\ y_{5,1} & y_{5,2} & y_{5,3} & y_{5,4} & y_{5,5} \end{bmatrix} \begin{bmatrix} 0 & 1/2 & 1 & -1/2 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} i_1^2 \\ i_2^2 \\ i_1 \\ i_2 \\ 1 \end{bmatrix}.$$

The alignment problem can be expressed as follows: Find C, D_A, and D_B such that $\forall \; (i_1, \; i_2) \in$ iteration space of this loop:

$$\begin{cases} C = & D_A F_A \\ C = & D_B F_B \end{cases}. \tag{13}$$

The equations system of (13) can be converted into the following matrix equation:

$$[C \quad D_A \quad B_B] \begin{bmatrix} \mathbf{I} & \mathbf{I} \\ -F_A & \mathbf{0} \\ \mathbf{0} & -F_B \end{bmatrix} = [\mathbf{0} \quad \mathbf{0}]. \tag{14}$$

Here, **I** is a 5×5 identity matrix and **0** is a 5×5 zero matrix. According to the method stated in Section 3.1, a solution matrix of (14) is:

$$[C \quad D_A \quad D_B] = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 2 & -1 & 0 & -2 & -1 & 1 & 0 & 0 & 2 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

This gives us

$$C = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 2 & -1 & 0 \end{bmatrix}, D_A = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ -2 & -1 & 1 & 0 & 0 \end{bmatrix}, \text{and}$$

$$D_B = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

We can obtain the mappings of computations and data as follows:

$$C\mathbf{i}' = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 2 & -1 & 0 \end{bmatrix} \begin{bmatrix} i_1^2 \\ i_2^2 \\ i_1 \\ i_2 \\ 1 \end{bmatrix} = \begin{bmatrix} i_1^2 + i_2^2 + i_1 + i_2 + 1 \\ i_1^2 + i_2^2 + i_1 + i_2 + 1 \\ i_1^2 + i_2^2 + i_1 + i_2 + 1 \\ 1 \\ i_2^2 + 2i_1 - i_2 \end{bmatrix},$$

$$D_A F_A \mathbf{i}' = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ -2 & -1 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1/2 & 0 & -1/2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} i_1^2 \\ i_2^2 \\ i_1 \\ i_2 \\ 1 \end{bmatrix} = \begin{bmatrix} i_1^2 + i_2^2 + i_1 + i_2 + 1 \\ i_1^2 + i_2^2 + i_1 + i_2 + 1 \\ i_1^2 + i_2^2 + i_1 + i_2 + 1 \\ 1 \\ i_2^2 + 2i_1 - i_2 \end{bmatrix}, \text{ and}$$

$$D_B F_B \mathbf{i}' = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1/2 & 1 & -1/2 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} i_1^2 \\ i_2^2 \\ i_1 \\ i_2 \\ 1 \end{bmatrix} = \begin{bmatrix} i_1^2 + i_2^2 + i_1 + i_2 + 1 \\ i_1^2 + i_2^2 + i_1 + i_2 + 1 \\ i_1^2 + i_2^2 + i_1 + i_2 + 1 \\ 1 \\ i_2^2 + 2i_1 - i_2 \end{bmatrix}.$$

Hence, using our alignment, iteration (i_1, i_2) is mapped onto virtual processor $(i_1^2 + i_2^2 + i_1 + i_2 + 1)$ and the corresponding A and B array elements are mapped onto the same virtual processor. The **Align** statements adopted to describe the alignment relation for the array elements of both A and B are represented as follows:

Align $A(1/2\ I_1^2 - 1/2\ I_1 + I_2)$ with $T(I_1^2 + I_2^2 + I_1 + I_2 + 1)$ and
Align $B(1/2\ I_2^2 + I_1 - 1/2\ I_2)$ with $T(I_1^2 + I_2^2 + I_1 + I_2 + 1)$.

Here, the virtual processors are supposed to be organized as a one-dimensional template T. Because no loop-carried output-dependences exist for array A and the required data elements (the written data element of array A and the corresponding read data element of array B) for a computation are mapped onto the same template element, this nested loop can be executed in parallel without interprocessor communication.

3.3 Features of the Proposed Techniques

In principle, our techniques have the following features:

Theorem 1. *If the alignment function of the template and the reference functions of the written arrays are in common form, i.e., multiple of the nonconstant items of the former is equal to multiple of the nonconstant items of the latter, then there are no distributed data updates for writing these arrays.*

Proof. Please refer to the Appendix. □

It is clear that our proposed techniques are not one-to-one mappings but many-to-one mappings, because dependent iterations with the properties described in Theorem 1 will be mapped onto the same template element. For instance, in Fig. 1, dependent iterations [2 3], [3 2], [1 4], and [4 1] will be mapped onto the same template array element $T(6)$. Thus, the size of the template array could have a smaller order than the iteration space. On the other hand the mappings of the read-only arrays will not inflect the correctness of execution, it is generally helpful to replicate multiple copies of a read-only data element onto the processors requiring that data element for computation, so that no further interprocessor communication is needed within the nested loops.

As mentioned, our proposed alignment techniques do not consider the data dependences. In this way, there are two principal advantages. First, the difficulty of alignment can be decreased, enabling our techniques to be applicable more broadly than other methods. Second, the original data dependences (if existed) are likely eliminated or reduced by the resulted alignment function because the dependent iterations and their needed data could be mapped onto the same temple element. This fact might benefit the exploitation of parallelism in distribution phase.

3.4 Time Complexity

The proposed alignment techniques reduce the problem of mapping the computations and data in a program to the standard linear algebra problem of determining a null space basis for a matrix. This includes three main phases. The first phase involves determining, according to the alignment constraints, the value for each element in the matrix V. The second phase involves applying Gaussian elimination to

TABLE 1
The Extracted Code Segment and the Data Alignments with Our Technique

DO 90 MI= 1,$MORB$ DO 80 MJ=1,MI	Align $XRSIJ(\frac{1}{2}MI^2 - \frac{1}{2}MI + MJ)$
80 $XRSIJ(\frac{1}{2}MI^2 - \frac{1}{2}MI + MJ)=XIJ(MJ)$	with $T(MI^2+MJ^2+MI+MJ+1)$, Align $XIJ(MJ)$
90 CONTINUE	with $T(MI^2+MJ^2+MI+MJ+1)$.

(For simplicity, the induction variable MRSIJ in the source code is substituted with $1/2\ MI^2 - 1/2\ MI + MJ$.*)*

determine a basis for U^T, the null space of V^T. The third phase involves extracting the solution matrices U.

Suppose that V is an $m \times n$ matrix and its rank is r, where r is at most the minimal value of m and n. The worst-case time complexity to determine the values for all of the elements of V is $\mathcal{O}(m \times n)$. Determining a null space basis for V^T using Gaussian elimination needs $\mathcal{O}(r \times m \times n + r^3)$ *arithmetic* operations. The worst-case time complexity to determine a basis for U^T is accordingly $\mathcal{O}(r \times m \times n + r^3)$. Due to $U = H(r + 1 : m, 1 : m)$, the worst-case time complexity to extract the solution matrices U from H is clearly $\mathcal{O}(m^2 - r \times m)$. Hence, the worst-case time complexity for the presented techniques is $\mathcal{O}(m \times n + r \times m \times n + r^3 + m^2 - r \times m)$, which is similar to that for the method proposed by Bau et al. in [7].

Generally, a single nested loop is the main program section to be parallelized, the data alignment, which reduces the interprocessor communication to avoid undermining the benefits of parallelism, is often discussed on a single nested loop basis [3], [6], [10]. In the next section, a code segment with a single nested loop extracted from a benchmark is executed with our alignment method. On the other hand, to reduce the communication for a program with multiple parallelizable nested loops, a data alignment focusing on the whole program may be needed. However, the alignment entirely based on the whole program is extremely complicated because many factors, e.g., data dependences across the nested loops, the iteration spaces for different nested loops and the data access patterns for each nested loop, etc., need to be considered. Alternatively, an intuitive way for aligning data arrays over multiple nested loops is to align the data arrays with the proposed techniques for each individual single nested loop. It is possible that large amounts of communication will occur in this way because the data arrays might need to be remapped onto the template elements and redistributed over the processors across the nested loops. Nevertheless, whether the time to run the program in the above way is beneficial might depend on the characteristics of the program. For example, if the program computation is enormous, the communication cost might become relatively less significant for the program execution. The executed program perhaps can obtain effective speedup. In the following section, a code segment with multiple parallelizable nested loops extracted from a benchmark is executed in the above way. The experimental result shows that the tested code segment can obtain effective speedup when its computation is vast.

4 Experimental Results

We have experimented with the proposed alignment techniques on some codes extracted, respectively, from TRFD of the Perfect Benchmarks and Vector Loop [4], [12] in our PC-cluster environment. Our PC-cluster includes a master, a PC with one P4 (Pentium 4) 1.8 GHz CPU and 256 MB main memory, and 10 slaves, each a PC with one P4 1.5GHz CPU and 128 MB main memory. The operation environment was the RedHat Linux 7.1 with the installed parallel software package-MPI-1.2.2.2. We hand-coded these extracted code segments in MPI (Message Passing Interface) with C language and executed them sequentially and in parallel in our MPI environment, respectively.

The code segment extracted from TRFD of the Perfect Benchmarks contains arrays referenced using quadratic subscripts, as shown in Table 1. This real code segment has no data dependence such that it can intrinsically be executed in parallel. Our proposed method can align the arrays of this code segment in a communication-free manner that does not cause interprocessor communication. The corresponding sequential and parallel run times for this code segment are shown in Fig. 3. Fig. 3 shows that the difference between sequential and parallel run time is insignificant. This is because that this tested code segment is not computation-intensive (only one statement with simple operation) that the cost for initially distributing the aligned data and finally receiving the computed results appears more significant. Generally, a computation-intensive nested loop that can be aligned in a communication-free manner should well benefit from parallelism.

On the other hand, Vector Loop consists of a main program, drivers, and a collection of subroutines that contain one or more nested loops of which the outer loop is used to increase the granularity of the calculation. By and large, this benchmark can be considered as consisting of a

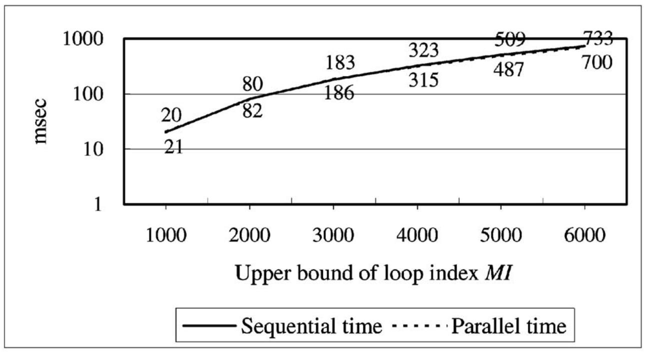

Fig. 3. The sequential and parallel runtimes for the extracted code segment.

TABLE 2
The Extracted Code Segment and the Data Alignments with Our Technique

Nested Loop 1 (SUB s422)	DO $J = 1, N1$ DO I=1,30 $X(I)$=$ARRAY(I$+8)+$A(I)$ ENDDO ENDDO	Align $X(I)$ with $T(I$+8), Align $ARRAY(I$+8) with $T(I$+8), and Align $A(I)$ with $T(I$+8).
Nested Loop 2 (SUB s423)	DO $J = 1, N1$ DO I=1,30 $ARRAY(I$+1)=$X(I)$+$A(I)$ ENDDO ENDDO	Align $X(I)$ with $T(I)$, Align $ARRAY(I$+1) with $T(I)$, and Align $A(I)$ with $T(I)$.
Nested Loop 3 (SUB s424)	DO $J = 1, N1$ DO I=1, 30 $X(I$+1)=$ARRAY(I)$+$A(I)$ ENDDO ENDDO	Align $X(I$+1) with $T(I)$, Align $ARRAY(I)$ with $T(I)$, and Align $A(I)$ with $T(I)$.

(In this experiment, the upper bound for loop index variable I in each nested loop was defined as 30, which is nonconstant in the original code.)

sequence of nested loops. In this experiment, we extracted a code segment from this benchmark that included three parallelizable nested loops, as shown in the Table 2.

In the parallel mode, the iterations and their needed data for each individual nested loop were remapped using our techniques onto the corresponding template elements and redistributed over the processors. The corresponding overall sequential and parallel run times for this code segment are shown in Fig. 4. It was not difficult to find that the code segment executed in parallel with each single nested loop aligned individually could obtain effective speedup when the code segment computation is vast. Therefore, when the program computation is enormous, the cost for remapping and redistributing become less significant for program execution.

5 Conclusions

Linear expressions are the most common subscript patterns for the referenced arrays and most data alignment methods were devised mainly to align the arrays referenced using linear subscripts with few loop index variables. According to Petersen and Padua's results [9] and our survey, the number of the arrays with quadratic subscript might attain to certain extent. However, the data alignments for the arrays referenced using quadratic subscripts were scarcely discussed before. In this paper, we offer two alignment techniques to properly map, in a communication-free manner, computations and arrays referenced using generalized linear subscripts or generalized quadratic subscripts onto the virtual processors. Our alignment techniques, based on elementary linear algebra, reduce the alignment problem to the problem of determining a null space basis for a matrix. By simplifying solving the null space basis, the proposed techniques can easily determine the desired mapping functions. Obviously, many different mapping functions can be obtained by different linear combinations of the null space basis. Additionally, because dependent iterations with the properties described in Theorem 1 will be mapped onto the same template element, the proposed techniques are not one-to-one mappings.

Nevertheless, to align arrays referenced using exponential subscripts with multiple loop index variables (e.g., subscripts with the pattern of $b_1 a_1^{I_1} + \ldots + b_n a_n^{I_n} + c_1 I_1 + \ldots + c_n I_n + d$), our techniques cannot be straightforwardly applied. Constructing a general communication-free alignment technique to achieve this would be our future research work.

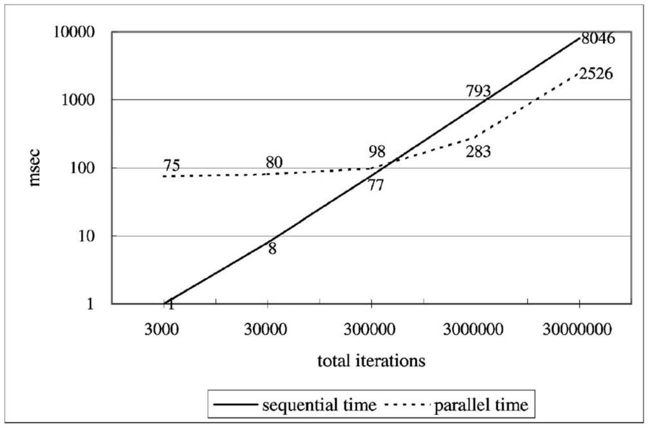

Fig. 4. The overall sequential and parallel runtimes for the extracted code segment.

Appendix

Theorem 1

Given that the alignment function of the template and the array reference functions of the written arrays are in the common form, i.e., multiple of the nonconstant items of the former is equal to multiple of the nonconstant items of the latter, then there is no distributed data updates for writing these arrays.

Proof. We give the proof for linear the cases, the proof for the quadratic cases can be obtained similarly. We first present the condition for two dependent iterations to be mapped onto the same template element. Let the alignment constraint for the iteration space of an n nested do loop be expressed as:

$$C\mathbf{i}' = \begin{bmatrix} c_{1,1} & c_{1,2} & \dots & c_{1,n+1} \\ c_{2,1} & c_{2,2} & \dots & c_{2,n+1} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n+1,1} & c_{n+1,2} & \dots & c_{n+1,n+1} \end{bmatrix} \begin{bmatrix} i_1 \\ \vdots \\ i_n \\ 1 \end{bmatrix} = \begin{bmatrix} c_{1,1}i_1 + c_{1,2}i_2 + \dots + c_{1,n}i_n + c_{1,n+1} \\ c_{2,1}i_1 + c_{2,2}i_2 + \dots + c_{2,n}i_n + c_{2,n+1} \\ \vdots \\ c_{n+1,1}i_1 + c_{n+1,2}i_2 + \dots + c_{n+1,n}i_n + c_{n+1,n+1} \end{bmatrix},$$

and the alignment constraint for the written array A_e in the general loop be represented as:

$$D_{A_e}F_{A_e}\mathbf{i}' = \begin{bmatrix} d_{1,1} & \dots & d_{1,n+1} \\ \vdots & \ddots & \vdots \\ d_{n+1,1} & \dots & d_{n+1,n+1} \end{bmatrix} \begin{bmatrix} a_{e,1} & a_{e,2} & \dots & a_{e,n} & a_{e,0} \\ 0 & 0 & \dots & 0 & 1 \\ 1 & 1 & \dots & 1 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & 1 & \dots & 1 & 1 \end{bmatrix} \begin{bmatrix} i_1 \\ \vdots \\ i_n \\ 1 \end{bmatrix} = \begin{bmatrix} d_{1,1} & \dots & d_{1,n+1} \\ \vdots & \ddots & \vdots \\ d_{n+1,1} & \dots & d_{n+1,n+1} \end{bmatrix} \begin{bmatrix} a_{e,1}i_1 + \dots + a_{e,n}i_n + a_{e,0} \\ 1 \\ i_1 + \dots + i_n + 1 \\ \vdots \\ i_1 + \dots + i_n + 1 \end{bmatrix},$$

and $C\mathbf{i}' = D_{A_e}F_{A_e}\mathbf{i}'$. Take any element of $C\mathbf{i}'$, e.g., $c_{k,1}i_1 + c_{k,2}i_2 + \dots + c_{k,n}i_n + c_{k,n+1}$ for $1 \leq k \leq n+1$, as the template alignment function by which the template elements are determined for the corresponding iterations (computations) and their needed data to be mapped onto, then two dependent iterations $\mathbf{i}_1 = [i_{1,1}i_{1,2}\dots i_{1,n}]$ and $\mathbf{i}_2 = [i_{2,1}\ i_{2,2}\dots i_{2,n}]$, which both access to the same element of array A_e, will be mapped onto the same template array element if and only if $i_{1,1} + i_{1,2} + \dots + i_{1,n} + 1 = i_{2,1} + i_{2,2} + \dots + i_{2,n} + 1$. It is clear that two iterations $\mathbf{i}_1 = [i_{1,1}\ i_{1,2}\dots i_{1,n}]$ and $\mathbf{i}_2 = [i_{2,1}\ i_{2,2}\dots i_{2,n}]$ can be mapped onto the same template element if and only if

$$c_{k,1}i_{1,1} + c_{k,2}i_{1,2} + \dots + c_{k,n}i_{1,n} + c_{k,n+1} = c_{k,1}i_{2,1} + c_{k,2}i_{2,2} + \dots + c_{k,n}i_{2,n} + c_{k,n+1}$$

for $1 \leq k \leq n+1$. Due to that $C\mathbf{i}' = D_{A_e}F_{A_e}\mathbf{i}'$, for two iterations $\mathbf{i}_1 = [i_{1,1}i_{1,2}\dots i_{1,n}]$ and $\mathbf{i}_2 = [i_{2,1}\ i_{2,2}\dots i_{2,n}]$, we have:

$$\begin{aligned} &c_{k,1}i_{1,1} + c_{k,2}i_{1,2} + \dots + c_{k,n}i_{1,n} + c_{k,n+1} \\ &= d_{k,1}(a_{e,1}i_{1,1} + \dots + a_{e,n}i_{1,n} + a_{e,0}) + d_{k,2} + \\ &\quad d_{k,3}(i_{1,1} + \dots + i_{1,n} + 1) + \dots + d_{k,n+1}(i_{1,1} + \dots + i_{1,n} + 1) \\ &= d_{k,1}(a_{e,1}i_{1,1} + \dots + a_{e,n}i_{1,n} + a_{e,0}) + d_{k,2} + \\ &\quad (d_{k,3} + \dots + d_{k,n+1})(i_{1,1} + \dots + i_{1,n} + 1), \end{aligned}$$

and

$$\begin{aligned} &c_{k,1}i_{2,1} + c_{k,2}i_{2,2} + \dots + c_{k,n}i_{2,n} + c_{k,n+1} \\ &= d_{k,1}(a_{e,1}i_{2,1} + \dots + a_{e,n}i_{2,n} + a_{e,0}) + d_{k,2} + \\ &\quad d_{k,3}(i_{2,1} + \dots + i_{2,n} + 1) + \dots + d_{k,n+1}(i_{2,1} + \dots + i_{2,n} + 1) \\ &= d_{k,1}(a_{e,1}i_{2,1} + \dots + a_{e,n}i_{2,n} + a_{e,0}) + d_{k,2} + \\ &\quad (d_{k,3} + \dots + d_{k,n+1})(i_{2,1} + \dots + i_{2,n} + 1). \end{aligned}$$

From the above equations, it is easy to find that, if

$$a_{e,1}i_{1,1} + \dots + a_{e,n}i_{1,n} + a_{e,0} = a_{e,1}i_{2,1} + \dots + a_{e,n}i_{2,n} + a_{e,0},$$

then

$$c_{k,1}i_{1,1} + c_{k,2}i_{1,2} + \dots + c_{k,n}i_{1,n} + c_{k,n+1} = c_{k,1}i_{2,1} + c_{k,2}i_{2,2} + \dots + c_{k,n}i_{2,n} + c_{k,n+1}$$

if and only if $i_{1,1} + \dots + i_{1,n} + 1 = i_{2,1} + \dots + i_{2,n} + 1$. For two dependent iterations $\mathbf{i}_1 = [i_{1,1}i_{1,2}\dots i_{1,n}]$ and $\mathbf{i}_2 = [i_{2,1}i_{2,2}\dots i_{2,n}]$, they access to the same data element of array A_e; that is,

$$a_{e,1}i_{1,1} + \dots + a_{e,n}i_{1,n} + a_{e,0} = a_{e,1}i_{2,1} + \dots + a_{e,n}i_{2,n} + a_{e,0}.$$

Thus, we have that two dependent iterations $\mathbf{i}_1 = [i_{1,1}\ i_{1,2}\dots i_{1,n}]$ and $\mathbf{i}_2 = [i_{2,1}\ i_{2,2}\dots i_{2,n}]$ will be mapped onto the same template array element if and only if $i_{1,1} + i_{1,2} + \dots + i_{1,n} + 1 = i_{2,1} + i_{2,2} + \dots + i_{2,n} + 1$. Similarly, for the quadratic cases, two dependent iterations $\mathbf{i}_1 = [i_{1,1}\ i_{1,2}\dots i_{1,n}]$ and $\mathbf{i}_2 = [i_{2,1}\ i_{2,2}\dots i_{2,n}]$ will be mapped onto the same template array element if and only if

$$i_{1,1}^2 + i_{1,2}^2 + \dots + i_{1,n}^2 + i_{1,1} + i_{1,2} + \dots + i_{1,n} + 1 = i_{2,1}^2 + i_{2,2}^2 + \dots + i_{2,n}^2 + i_{2,1} + i_{2,2} + \dots + i_{2,n} + 1.$$

Now, suppose that $c_{k,1}i_1 + c_{k,2}i_2 + \dots + c_{k,n}i_n + c_{k,n+1}$ for $1 \leq k \leq n+1$ is the template alignment function and $a_{e,1}i_1 + a_{e,2}i_2 + \dots + a_{e,n}i_n + a_{e,0}$ is the access function of the written array A_e and by the assumption,

$$m \times (c_{k,1}i_1 + c_{k,2}i_2 + \dots + c_{k,n}i_n) = n \times (a_{e,1}i_1 + a_{e,2}i_2 + \dots + a_{e,n}i_n),$$

m and n are nonzero positive integers. Due to the fact that $C\mathbf{i}' = D_{A_e}F_{A_e}\mathbf{i}'$, for two iterations $\mathbf{i}_1 = [i_{1,1}\ i_{1,2}\dots i_{1,n}]$ and $\mathbf{i}_2 = [i_{2,1}\ i_{2,2}\dots i_{2,n}]$, we have:

$$\begin{aligned} &d_{k,1}(a_{e,1}i_{1,1} + \dots + a_{e,n}i_{1,n} + a_{e,0}) + d_{k,2} + \\ &d_{k,3}(i_{1,1} + \dots + i_{1,n} + 1) + \dots + d_{k,n+1}(i_{1,1} + \dots + i_{1,n} + 1) \\ &= c_{k,1}i_{1,1} + c_{k,2}i_{1,2} + \dots + c_{k,n}i_{1,n} + c_{k,n+1} \\ &= (n/m) \times (a_{e,1}i_{1,1} + a_{e,2}i_{1,2} + \dots + a_{e,n}i_{1,n}) + c_{k,n+1}, \end{aligned}$$

and

$$\begin{aligned} &d_{k,1}(a_{e,1}i_{2,1} + \dots + a_{e,n}i_{2,n} + a_{e,0}) + d_{k,2} + \\ &d_{k,3}(i_{2,1} + \dots + i_{2,n} + 1) + \dots + d_{k,n+1}(i_{2,1} + \dots + i_{2,n} + 1) \\ &= c_{k,1}i_{2,1} + c_{k,2}i_{2,2} + \dots + c_{k,n}i_{2,n} + c_{k,n+1} \\ &= (n/m) \times (a_{e,1}i_{2,1} + a_{e,2}i_{2,2} + \dots + a_{e,n}i_{2,n}) + c_{k,n+1}. \end{aligned}$$

From the above equations, it is easy to find that, if

$$a_{e,1}i_{1,1} + \dots + a_{e,n}i_{1,n} = a_{e,1}i_{2,1} + \dots + a_{e,n}i_{2,n},$$

then

$$\begin{aligned} &d_{k,1}(a_{e,1}i_{1,1} + \dots + a_{e,n}i_{1,n} + a_{e,0}) + d_{k,2} + \\ &d_{1,3}(i_{1,1} + \dots + i_{1,n} + 1) + \dots + d_{k,n+1}(i_{1,1} + \dots + i_{1,n} + 1) = \\ &d_{k,1}(a_{e,1}i_{2,1} + \dots + a_{e,n}i_{2,n} + a_{e,0}) + d_{k,2} + \\ &d_{k,3}(i_{2,1} + \dots + i_{2,n} + 1) + \dots + d_{k,n+1}(i_{2,1} + \dots + i_{2,n} + 1), \end{aligned}$$

i.e.,

$$i_{1,1} + \dots + i_{1,n} + 1 = i_{2,1} + \dots + i_{2,n} + 1.$$

For two dependent iterations $\mathbf{i}_1 = [i_{1,1}\ i_{1,2} \ldots i_{1,n}]$ and $\mathbf{i}_2 = [i_{2,1}\ i_{2,2} \ldots i_{2,n}]$, we have

$$a_{e,1}i_{1,1} + a_{e,2}i_{1,2} + \ldots + a_{e,n}i_{1,n} + a_{e,0} = a_{e,1}i_{2,1} + a_{e,2}i_{2,2} + \ldots + a_{e,n}i_{2,n} + a_{e,0},$$

i.e.,

$$a_{e,1}i_{1,1} + a_{e,2}i_{1,2} + \ldots + a_{e,n}i_{1,n} = a_{e,1}i_{2,1} + a_{e,2}i_{2,2} + \ldots + a_{e,n}i_{2,n}$$

and, thus, $i_{1,1} + \ldots + i_{1,n} + 1 = i_{2,1} + \ldots + i_{2,n} + 1$. Thus, from above, two dependent iterations which both write the same element of array A_e will be mapped onto the same template array element to avoid the distributed data updates for writing the same data element of array A_e. □

REFERENCES

[1] J. Edmonds, "Systems of Distinct Representative and Linear Algebra," *J. Research of Nat'l Bureau of Standards, Section B,* vol. 71, no. 4, pp. 241-245, 1967.

[2] D.G. Luenberger, *Linear and Nonlinear Programming.* Addison-Wesley Publishing Company, 1984.

[3] J. Ramanujam and P. Sadayappan, "Compile-Time Techniques for Data Distributed in Distributed Memory Machines," *IEEE Trans. Parallel and Distributed Systems,* vol. 2, no. 4, pp. 472-482, Apr. 1991.

[4] D. Levine, D. Callahan, and J. Dongarra, "A Comparative Study of Automatic Vectorizing Compilers," *Parallel Computing,* vol. 17, pp. 1223-1244, 1991.

[5] J. Dongarra, M. Furtney, S. Reinhardt, and J. Russell, "Parallel Loops—A Test Suite for Parallelizing Compilers: Description and Example Results," *Parallel Computing,* vol. 17, pp. 1247-1255, 1991.

[6] P. Feautrier, "Toward Automatic Partitioning of Arrays on Distributed Memory Computers," *Proc. ACM Int'l Conf. Supercomputing,* pp. 175-184, 1993.

[7] D. Bau, I. Kodukula, V. Kotlyar, K. Pingali, and P. Stodghill, "Solving Alignment Using Elementary Linear Algebra," *Proc. Conf. Record Seventh Workshop Languages and Compilers for Parallel Computing,* pp. 46-60, Aug. 1994.

[8] M. Wolfe, *High Performance Compilers for Parallel Computing.* Redwood City: Addison-Wesley Publishing Company, 1996.

[9] P.M. Petersen and D.A. Padua, "Static and Dynamic Evaluation of Data Dependence Analysis Techniques," *IEEE Trans. Parallel and Distributed Systems,* vol. 7, no. 11, pp. 1121-1132, Nov. 1996.

[10] M. Dion and Y. Robert, "Mapping Affine Loop Nests: New Results," *Parallel Computing,* vol. 22, no. 10, pp. 1373-1397, Dec. 1996.

[11] P. Lee, "Efficient Algorithms for Data Distribution on Distributed Memory Parallel Computers," *IEEE Trans. Parallel and Distributed Systems,* vol. 8, no. 8, pp. 825-839, Aug. 1997.

[12] R. Eigenmann, J. Hoeflinger, and D. Padua, "On the Automatic Parallelization of the Perfect Benchmarks," *IEEE Trans. Parallel and Distributed Systems,* vol. 9, no. 1, pp. 5-23, Jan. 1998.

[13] Y.-C. Chung, C.-H. Hsu, and S.-W. Bai, "A Basic-Cycle Calculation Technique for Efficient Dynamic Data Redistribution," *IEEE Trans. Parallel and Distributed Systems,* vol. 9, no. 4, pp. 359-377, Apr. 1998.

[14] A.W. Lam and M.S. Lam, "Maximizing Parallelism and Minimizing Synchronization with Affine Partitions," *Parallel Computing,* vol. 24, nos. 3-4, pp. 445-475, May 1998.

[15] M. Kandemir, A. Choudhary, N. Shenoy, P. Banerjee, and J. Ramanujam, "A Hyperplane Based Approach for Optimizing Spatial Locality in Loop Nests," *Proc. 12th ACM Int'l Conf. Supercomputing,* pp. 69-76, July 1998.

[16] M. Kandemir, J. Ramanujam, A. Choudhary, and P. Banerjee, "A Loop Transformation Algorithm Based on Explicit DADA Layout Representation for Optimizing Locality," *Proc. 11th Int'l Workshop Languages and Compilers for Parallel Computing,* pp. 34-50, Aug. 1998.

[17] C.-P. Chu, W.-L. Chang, I. Chen, and P.-S. Chen, "Communication-Free Alignment for Array References with Linear Subscripts in Two Loop Index Variables or Quadratic Subscripts," *Proc. Second IASTED Int'l Conf. Parallel and Distributed Computing and Networks,* pp. 571-576, 1998.

[18] V. Boudet, F. Rastello, and Y. Robert, "Alignment and Distribution is NOT (Always) NP-Hard," *Proc. Int'l Conf. Parallel and Distributed Systems,* vol. 5, no. 9, pp. 648-657, Dec. 1998.

[19] C.-J. Liao and Y.-C. Chung, "Tree-Based Load Balancing Methods for Solution-Adaptive Finite Element Graphs on Distributed Memory Multicomputers," *IEEE Trans. Parallel and Distributed Systems,* vol. 10, no. 4, pp. 360-370, Apr. 1999.

[20] G.-H. Hwang and J.K. Lee, "An Expression-Rewriting Framework to Generate Communication Sets for HPF Programs with Block-Cyclic Distribution," *Parallel Computing,* vol. 25, pp. 1105-1139, 1999.

[21] A.W. Lam, G.I. Cheong, and M.S. Lam, "An Affine Partitioning Algorithm to Maximize Parallelism and Minimize Communication," *Proc. 13th ACM Int'l Conf. Supercomputing,* pp. 228-237, June 1999.

[22] K.-P. Shih, J.-P. Sheu, and C.-H. Huang, "Statement-Level Communication-Free Partitioning Techniques for Parallelizing Compilers," *J. Supercomputing,* pp. 243-269, vol. 15, no. 3, Feb. 2000.

[23] C.-H. Hsu, S.-W. Bai, Y.-C. Chung, and C.-S. Yang, "A Generalized Basic-Cycle Calculation Method for Array Redistribution," *IEEE Trans. Parallel and Distributed Systems,* vol. 11, no. 12, pp. 1201-1216, Dec. 2000.

[24] W.-L. Chang, C.-P. Chu, and J.-H. Wu, "Communication-Free Alignment for Array References with Linear Subscripts in Three Loop Index Variables or Quadratic Subscripts," *J. Supercomputing,* vol. 20, no. 1, pp. 67-83, Aug. 2001.

Weng-Long Chang received the BS and MS degrees in computer science and information engineering from Feng Chia University and National Cheng Kung University, Taiwan, in 1988 and 1994, respectively. In 1999, he received the PhD degree in computer science and information engineering from National Cheng Kung University. He is currently an assistant professor in the Department of Information Management, Southern Taiwan University of Technology, Tainan County, Taiwan. His research interests include languages and compilers for parallel computing and molecular computing.

Jih-Woei Huang received the BS degree in computer engineering from Feng Chia University, Taiwan, Republic of China, in 1986, and the MS degree in computer engineering from the National Chiao Tung University, Taiwan, Republic of China, in 1991. He is currently working toward the PhD degree in the Department of Computer Science and Information Engineering, National Cheng Kung University, Taiwan, Republic of China. His research interest is mainly in paralleling compilers.

Chih-Ping Chu received the BS degree in agricultural chemistry from National Chung Hsing University, Taiwan, the MS degree in computer science from the University of California, Riverside, and the PhD degree in computer science from Louisiana State University. He is currently a professor in the Department of Computer Science and Information Engineering at National Cheng Kung University, Taiwan. His research interests include parallelizing compilers, parallel computing, parallel processing, Internet computing, and software engineering.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** http://computer.org/publications/dlib.