The Journal of Supercomputing, 20, 67–83, 2001

Communication-Free Alignment for Array References with Linear Subscripts in Three Loop Index Variables or Quadratic Subscripts

WENG-LONG CHANG
Department of Management Information System
Kung Shan Technology University
Tainan, Taiwan 701, R.O.C.

CHIH-PING CHU AND JIA-HWA WU chucp@csie.ncku.edu.tw
Department of Computer Science and Information Engineering
National Cheng Kung University
Tainan, Taiwan 701, R.O.C.

Abstract. Bau et al. proposed an efficient and precise data alignment method to ascertain whether there is communication-free alignment of array reference function with linear subscripts in one loop index variable. Chu et al. presented an efficient and precise data alignment method to determine whether there is communication-free alignment of array reference function with linear subscripts in two loop index variables or quadratic subscripts ($ai^2 + bi + d$). However, for array reference function with linear subscripts in three loop index variables or quadratic subscripts ($ai^2 + bi + cj + d$), their methods cannot be applied. In this paper, we propose two new alignment functions for loop iteration space and array elements. The new alignment functions can be applied towards checking whether there is communication-free alignment of array reference function with linear subscripts in three loop index variables or quadratic subscripts. Experiments with benchmarks taken from Parallel loop and Vector loop showed that among the 7 nested loops tested, three of them had their data alignment improved by the method proposed.

Keywords: parallelizing compilers, communication-free alignment, parallel computing, loop optimization, data dependence analysis, load balancing

1. Introduction

Distributed memory multiprocessors and shared memory multiprocessors are the two most popular parallel computational models in high speed computing. However, distributed memory multiprocessors are superior to shared memory multiprocessors in scalability and latency tolerance and increasingly have been used for scientific and engineering applications. The major shortcoming of distributed memory multiprocessors is the difficulty in programming due to the lack of a shared memory space [5, 6]. Hence, programmers or compilers are responsible for distributing code and data over processors, and managing communication among tasks.

Over the last decade, many researchers have paid attention to maximizing parallelism and minimizing communication for a given program executed on a parallel machine [4, 7–14]. Kandemir et al. [8, 9, 12] proposed a linear algebra framework to automatically determine the optimal data layouts expressed as hyper-planes for each array referenced in a program. Lim et al. [10, 13] presented approaches to analyzing data reference patterns of a program with structure of nested loops so that the parallelized program is running on parallel machines in a communication-free manner with some constraints. Boudet et al. [11] proposed a method to solving the alignment problem by considering how to distribute and how to align data arrays while at the same time considering how to preserve parallelism on a given program. Shih et al. [14] took account of communication-free partitioning of statement-iteration spaces and data spaces along hyperplanes for multi-statements in perfect and imperfect loops. They offered the necessary and sufficient conditions for the feasibility of communication-free hyperplane partitions.

Bau et al. [4] employed elementary matrix methods to determine communication-free alignment of code and data and to solve the problem of replicating read-only data for eliminating communication. Their method is an efficient and precise data alignment method for checking whether there is communication-free alignment of array reference function with linear subscripts in one loop index variable. Chu et al. [7] presented an efficient and precise data alignment method to determine whether there is communication-free alignment of array reference function with linear subscripts in two loop index variables or quadratic subscripts $(ai^2 + bi + d)$. However, for array reference function with linear subscripts in three loop index variables or quadratic subscripts $(ai^2 + bi + cj + d)$, the two methods cannot be applied to manipulate them.

In this paper, we propose two alignment functions for loop iteration space and array elements. These alignment functions can be applied towards determining whether there is communication-free alignment of array reference function with linear subscripts in three loop index variables or quadratic subscripts. A theoretical analysis presented below explains that we take advantage of elementary linear algebra to determine communication-free alignment of array reference function with linear subscripts in three loop index variables or quadratic subscripts. The proposed alignment functions have been implemented and several benchmark experiments have also been performed. Quantitative results from these tests will be presented.

The rest of this paper is organized as follows. In Section 2, the problem of data communication and data computation is reviewed. In Section 3, the theoretical aspects and the worst-case time complexity of the presented method are described. Experimental results showing the advantages of the proposed method are given in Section 4. Finally, brief conclusions are drawn in Section 5.

2. Data communication and data computation

Data and computation placement is one important problem for generating code in distributed memory multiprocessors. It is obvious that the problem is to check what task each processor must finish and what data must be stored in local memory. It is well-known that the goal of placement is to determine parallelism by spreading the task across the processors, and to exploit locality by distributing data among processors so that memory accesses are local whenever possible. The problem of checking a good placement for a program is usually solved in two phases called *alignment and distribution*. The alignment phase maps data and computations to a set of virtual processors organized as a Cartesian grid of some dimension (a *template* in HPF Fortran terminology). The distribution phase folds the virtual processors into the physical processors. The advantage of separating alignment from distribution is that we can focus on the allocation problem (determining which iterations and data should be mapped to the same processor) without worrying about the load balancing problem. Our focus in this paper is alignment.

3. The new alignment functions proposed

The presented method contains three main steps. The first step is to ascertain the constraints on data and computation placement. The second step is to check which constraints should be left unsatisfied. The third step is to solve the remaining system of constraints for determining data and computation placement. In the first step, data accesses in the program are examined to ascertain a system of linear equations or a system of quadratic equations in which the unknowns are functions representing data and computation placement. Any solution to the system determines a so-called communication-free alignment. This is to say that all data required by a processor that executes the iterations mapped to it must reside in its local memory. Very often, the only communication-free alignment for a program is the trivial one where every iteration and data is mapped to a single processor. Intuitively, each equation in the system is a constraint on data and computation placement, and it is possible to over-constrain the system so that the trivial solution is the only one. If so, the second step ascertains which constraints must be left unsatisfied to retain parallelism in execution. The cost of leaving a constraint unsatisfied is that it introduces communication; therefore, the constraints left unsatisfied should be those that introduce as little communication as possible. In the last step, the remaining constraints are solved to determine data and computation placement. The proposed method can reduce the problem of solving remaining constraints to the standard linear algebra problem of determining a basis for the null space of a matrix. The presented method applies integer preserving Gaussian elimination to resolve this linear algebra problem [1]. In the following sections, the theoretical aspects and the worst-case time complexity of the proposed method are presented.

3.1. Array reference function with linear subscripts

It is assumed that there are m statements and q arrays with linear references in a general loop. The general loop is assumed to contain 3 common loops. A reference function to an array A_e for $1 \leq e \leq q$ in a general loop is $F_e = a_{e,1}I + b_{e,1}J + c_{e,1}K + d_{e,1}$, where I, J, and K are index variables of the general loop and $a_{e,1}, b_{e,1}, c_{e,1}$ and $d_{e,1}$ are all coefficients. It is postulated that M is a 4-component column vector and M is represented as $(I, J, K, 1)^T$, where T is transposition operator. If **i** is an iteration vector in the iteration space of a general loop, the alignment constraints require that the processor performing iteration **i** must own $A_e(F_eM)$ for $1 \le e \le q$. The alignment constraint for the iteration space of a general loop can be expressed formally as

$$[\vec{C}]_{4\times 4}[M]_{4\times 1} = \begin{bmatrix} c_{1,1} & c_{1,2} & c_{1,3} & c_{1,4} \\ c_{2,1} & c_{2,2} & c_{2,3} & c_{2,4} \\ c_{3,1} & c_{3,2} & c_{3,3} & c_{3,4} \\ c_{4,1} & c_{4,2} & c_{4,3} & c_{4,4} \end{bmatrix}_{4\times 4} \begin{bmatrix} I \\ J \\ K \\ 1 \end{bmatrix}_{4\times 1},$$

where $\vec{C}$ is a 4×4 matrix. Similarly, the alignment constraint for an array A_e for $1 \le e \le q$ in the general loop can also be, respectively, represented as

$$\begin{aligned} &[\vec{D}_{A_e}]_{4\times 4}[\vec{F}_{A_e}]_{4\times 4}[M]_{4\times 4} \\ &= \begin{bmatrix} w_{1,1} \\ w_{2,1} \\ w_{3,1} \\ w_{4,1} \end{bmatrix}_{4\times 1} * \left[a_{e,1}I + b_{e,1}J + c_{e,1}K + d_{e,1}\right]_{1\times 1} \\ &+ \begin{bmatrix} w_{1,2} \\ w_{2,2} \\ w_{3,2} \\ w_{4,2} \end{bmatrix}_{4\times 1} + \begin{bmatrix} w_{1,3} \\ w_{2,3} \\ w_{3,3} \\ w_{4,3} \end{bmatrix}_{4\times 1} * [I + J + K + 1]_{1\times 1} \\ &+ \begin{bmatrix} w_{1,4} \\ w_{2,4} \\ w_{3,4} \\ w_{4,4} \end{bmatrix}_{4\times 1} * [I + J + K + 1]_{1\times 1} \\ &= \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & w_{1,4} \\ w_{2,1} & w_{2,2} & w_{2,3} & w_{2,4} \\ w_{3,1} & w_{3,2} & w_{3,3} & w_{3,4} \\ w_{4,1} & w_{4,2} & w_{4,3} & w_{4,4} \end{bmatrix}_{4\times 4} \begin{bmatrix} a_{e,1} & b_{e,1} & c_{e,1} & d_{e,1} \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}_{4\times 4} \begin{bmatrix} I \\ J \\ K \\ 1 \end{bmatrix}_{4\times 1}. \end{aligned}$$

Therefore, the alignment problem can be expressed as follows: find $\vec{C}$ and $\vec{D}_{A_e}$ such that

$\forall\ (I, J, K) \in$ iteration space of loop:

$$[\vec{C}]_{4\times 4}[M]_{4\times 4} = [\vec{D}_{A_e}]_{4\times 4}[\vec{F}_{A_e}]_{4\times 4}[M]_{4\times 1}. \tag{Ex1}$$

To cancel the column vector M from both sides of equations, we need the following two lemmas. The following two lemmas are extensions of Lemma 1 in [4] and Lemmas 3.1 and 3.2 in [7].

Lemma 3.1 *Let* $\mathbf{T} \cdot \mathbf{t}$ *and* x *be a* $q \times 1$ *matrix, a q-component column vector and a scalar variable, respectively. Then*

$$\forall\ x[\mathbf{T} \quad \mathbf{t}]_{q\times 2}\begin{bmatrix} x \\ 1 \end{bmatrix}_{2\times 1} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}_{q\times 1} \Leftrightarrow \mathbf{T} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}_{q\times 1} \quad \textit{and} \quad \mathbf{t} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}_{q\times 1}.$$

Proof:
($\Rightarrow$) In particular, we can let x be equal to zero. This gives us:

$$[\mathbf{T}\quad \mathbf{t}]_{q\times 2}\begin{bmatrix}0\\1\end{bmatrix}_{2\times 1} = \mathbf{t} = \begin{bmatrix}0\\ \vdots\\ 0\end{bmatrix}_{q\times 1}.$$

So we can obtain $\mathbf{t}$ with a q-component zero vector. Now, for any x:

$$[\mathbf{T}\quad \mathbf{0}]_{q\times 2}\begin{bmatrix}x\\1\end{bmatrix}_{2\times 1} = [\mathbf{T}]_{q\times 1}\times x = \begin{bmatrix}0\\ \vdots\\ 0\end{bmatrix}_{q\times 1},$$

where $\mathbf{0}$ is a q-component zero vector,

which means that $\mathbf{T}$ is equal to a $q\times 1$ zero matrix. Therefore, we can conclude

$$\forall\, x[\mathbf{T}\quad \mathbf{t}]_{q\times 2}\begin{bmatrix}x\\1\end{bmatrix}_{2\times 1} = \begin{bmatrix}0\\ \vdots\\ 0\end{bmatrix}_{q\times 1} \Rightarrow \mathbf{T} = \begin{bmatrix}0\\ \vdots\\ 0\end{bmatrix}_{q\times 1} \text{ and } \mathbf{t} = \begin{bmatrix}0\\ \vdots\\ 0\end{bmatrix}_{q\times 1}. \tag{1}$$

($\Leftarrow$) Because $\mathbf{T} = \begin{bmatrix}0\\ \vdots\\ 0\end{bmatrix}_{q\times 1}$ and $\mathbf{t} = \begin{bmatrix}0\\ \vdots\\ 0\end{bmatrix}_{q\times 1}$, for any x we can gain

$$[\mathbf{T}\quad \mathbf{t}]_{q\times 2}\begin{bmatrix}x\\1\end{bmatrix}_{2\times 1} = \begin{bmatrix}0\\ \vdots\\ 0\end{bmatrix}_{q\times 1}. \tag{2}$$

From (1) and (2), we hence can derive

$$\forall\, x[\mathbf{T}\quad \mathbf{t}]_{q\times 2}\begin{bmatrix}x\\1\end{bmatrix}_{2\times 1} = \begin{bmatrix}0\\ \vdots\\ 0\end{bmatrix}_{q\times 1} \Leftrightarrow \mathbf{T} = \begin{bmatrix}0\\ \vdots\\ 0\end{bmatrix}_{q\times 1} \text{ and } \mathbf{t} = \begin{bmatrix}0\\ \vdots\\ 0\end{bmatrix}_{q\times 1}. \qquad \blacksquare$$

Lemma 3.2 *Let Q, R and S be $q\times 1$ matrices, and let t, x, y and z be, respectively, a q-component column vector and scalar variables. Then*

$$\forall\, xyz[Q\quad R\quad S\quad t]_{q\times 4}\begin{bmatrix}x\\y\\z\\1\end{bmatrix}_{4\times 1} = \begin{bmatrix}0\\ \vdots\\ 0\end{bmatrix}_{q\times 1} \Leftrightarrow Q = \begin{bmatrix}0\\ \vdots\\ 0\end{bmatrix}_{q\times 1},$$

$$R = \begin{bmatrix}0\\ \vdots\\ 0\end{bmatrix}_{q\times 1},\quad S = \begin{bmatrix}0\\ \vdots\\ 0\end{bmatrix}_{q\times 1} \quad \textit{and} \quad t = \begin{bmatrix}0\\ \vdots\\ 0\end{bmatrix}_{q\times 1}.$$

Proof: ($\Rightarrow$) In particular, we can let x, y and z be equal to zero. This gives us:

$$[Q \quad R \quad S \quad t\,]_{q\times 4}\begin{bmatrix}0\\0\\0\\1\end{bmatrix}_{4\times 1} = \mathbf{t} = \begin{bmatrix}0\\ \vdots\\0\end{bmatrix}_{q\times 1}.$$

So we can obtain $\mathbf{t}$ with a q-component zero vector. Similarly, we can let x and y be equal to zero. This gives us:

$$[Q \quad R \quad S \quad \mathbf{0}\,]_{q\times 4}\begin{bmatrix}0\\0\\z\\1\end{bmatrix}_{4\times 1} = [S]_{q\times 1} \times z = \begin{bmatrix}0\\ \vdots\\0\end{bmatrix}_{q\times 1},$$

where $\mathbf{0}$ is a q-component zero vector, which infers S to be a $q \times 1$ zero matrix. Similarly, we can let x be equal to zero. This gives us:

$$[Q \quad R \quad \mathbf{0} \quad \mathbf{0}\,]_{q\times 4}\begin{bmatrix}0\\y\\z\\1\end{bmatrix}_{4\times 1} = [R]_{q\times 1} \times y = \begin{bmatrix}0\\ \vdots\\0\end{bmatrix}_{q\times 1},$$

where $\mathbf{0}$ is a q-component zero vector, which infers R to be a $q \times 1$ zero matrix. Now, for any x, y and z:

$$[Q \quad \mathbf{0} \quad \mathbf{0} \quad \mathbf{0}\,]_{q\times 4}\begin{bmatrix}x\\y\\z\\1\end{bmatrix}_{4\times 1} = [Q]_{q\times 1} \times x = \begin{bmatrix}0\\ \vdots\\0\end{bmatrix}_{q\times 1},$$

where $\mathbf{0}$ is a q-component zero vector, which derives Q to be a $q \times 1$ zero matrix. Hence, we can conclude

$$\forall\, x\,y\,z\,[Q \quad R \quad S \quad t\,]_{q\times 4}\begin{bmatrix}x\\y\\z\\1\end{bmatrix}_{4\times 1} = \begin{bmatrix}0\\ \vdots\\0\end{bmatrix}_{q\times 1} \Rightarrow Q = \begin{bmatrix}0\\ \vdots\\0\end{bmatrix}_{q\times 1},$$

$$R = \begin{bmatrix}0\\ \vdots\\0\end{bmatrix}_{q\times 1},\ S = \begin{bmatrix}0\\ \vdots\\0\end{bmatrix}_{q\times 1} \text{ and } t = \begin{bmatrix}0\\ \vdots\\0\end{bmatrix}_{q\times 1}. \tag{3}$$

($\Leftarrow$) Because $Q = \begin{bmatrix}0\\ \vdots\\0\end{bmatrix}_{q\times 1}$, $R = \begin{bmatrix}0\\ \vdots\\0\end{bmatrix}_{q\times 1}$, $S = \begin{bmatrix}0\\ \vdots\\0\end{bmatrix}_{q\times 1}$ and $\mathbf{t} = \begin{bmatrix}0\\ \vdots\\0\end{bmatrix}_{q\times 1}$, for any x, y, and z we can obtain

$$[Q \quad R \quad S \quad t\,]_{q\times 4}\begin{bmatrix}x\\y\\z\\1\end{bmatrix}_{4\times 1} = \begin{bmatrix}0\\ \vdots\\0\end{bmatrix}_{q\times 1}. \tag{4}$$

From (3) and (4), therefore, we can infer

$$\forall\, x\,y\,z\, [Q \quad R \quad S \quad t]_{q\times 4} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}_{4\times 1} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}_{q\times 1} \Leftrightarrow Q = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}_{q\times 1},$$

$$R = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}_{q\times 1}, \; S = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}_{q\times 1} \text{ and } t = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}_{q\times 1}. \qquad \blacksquare$$

Using Lemmas 3.1 and 3.2, (Ex1) can be rewritten as

$$\left[\vec{C}\right]_{4\times 4} = \left[\vec{D}_{A_r}\right]_{4\times 4} \left[\vec{F}_{A_e}\right]_{4\times 4}. \tag{Ex2}$$

The system of equations (Ex2) can be converted into the following matrix equation:

$$\begin{aligned} &[\vec{C} \quad \vec{D}_{A_r} \quad \cdots \quad \vec{D}_{A_e}]_{4\times(4\times q+4)} \begin{bmatrix} 1 & \cdots & \mathbf{I} \\ -\vec{F}_{A_l} & \ddots & 0 \\ \vdots & \vdots & 0 \\ 0 & 0 & -\vec{F}_{A_q} \end{bmatrix}_{(4\times q+4)\times(4\times q)} \\ &= \begin{bmatrix} 0 & \cdots & 0 \\ 0 & \cdots & 0 \\ 0 & \cdots & 0 \end{bmatrix}_{4\times(4\times q)}, \end{aligned} \tag{Ex3}$$

where $\mathbf{I}$ and $\mathbf{0}$ are, respectively, a 3×3 identity matrix and a 3×3 zero matrix.

To resolve the matrix equation $[U]_{S\times M}[V]_{M\times N} = [\mathbf{0}]_{S\times N}$, we must get V into a 'rank-revealing' form. The idea cited from [1, 4] is to apply row and column operations as needed to get the matrix into a form in which its rank can be determined by inspection. There are many ways to finish this, and we describe one such. Suppose $V \in Z^{M\times N}$ and $rank(V) = r$. Then by integer preserving Gaussian elimination with full pivoting [1] we can establish the following factorization:

$$[H]_{M\times M}[V]_{M\times N}[P]_{N\times N} = \begin{bmatrix} R_{1,1} & R_{1,2} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}_{M\times N}, \tag{5}$$

where H is an $M \times M$ invertible matrix representing the row operations, P is an $N \times N$ unimodular matrix representing the column operations (which are permutations), $R_{1,1}$ is an $r \times r$ upper triangular invertible matrix. It is a property of this factorization that the last M–r rows of H span the null-space of V and hence give us the solution [1, 4]:

$$U = H(r+1 : M, 1 : M). \tag{6}$$

This also means that during the elimination we need to store only H, the composition of row operations.

```
do $I$=1 to 4
  do $J$=1 to 4
    do $K$=1 to 4
      $A(I+J+K) = B(I+J+K+1)$;
    enddo
  enddo
enddo
```

Figure 1. Array references with linear subscripts.

The following loop in Figure 1 is applied towards illustrating our points. If **i** is an iteration vector in the iteration space of the loop, the alignment constraints require the processor performing iteration **i** to own $A(F_1M)$ and $B(F_2M)$, where F_1 and F_2 are the following matrices:

$$F_1 = [1 \quad 1 \quad 1 \quad 0]_{1\times 4} \text{ and } F_2 = [1 \quad 1 \quad 1 \quad 1]_{1\times 4}.$$

The alignment constraint for the iteration space of the loop can be expressed formally as

$$[\vec{C}]_{4\times 4}[M]_{4\times 1} = \begin{bmatrix} c_{1,1} & c_{1,2} & c_{1,3} & c_{1,4} \\ c_{2,1} & c_{2,2} & c_{2,3} & c_{2,4} \\ c_{3,1} & c_{3,2} & c_{3,3} & c_{3,4} \\ c_{4,1} & c_{4,2} & c_{4,3} & c_{4,4} \end{bmatrix}_{4\times 4} \begin{bmatrix} I \\ J \\ K \\ 1 \end{bmatrix}_{4\times 1},$$

where $\vec{C}$ is a 4×4 matrix. Similarly, the alignment constraints for the two arrays A and B can also be, respectively, represented as

$$[\vec{D}_A]_{4\times 4}[\vec{F}_A]_{4\times 4}[M]_{4\times 1} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & w_{1,4} \\ w_{2,1} & w_{2,2} & w_{2,3} & w_{2,4} \\ w_{3,1} & w_{3,2} & w_{3,3} & w_{3,4} \\ w_{4,1} & w_{4,2} & w_{4,3} & w_{4,4} \end{bmatrix}_{4\times 4}$$
$$\times \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}_{4\times 4} \begin{bmatrix} I \\ J \\ K \\ 1 \end{bmatrix}_{4\times 1}, \quad \text{and}$$

$$[\vec{D}_B]_{4\times4}[\vec{F}_B]_{4\times4}[M]_{4\times1} = \begin{bmatrix} y_{1,1} & y_{1,2} & y_{1,3} & y_{1,4} \\ y_{2,1} & y_{2,2} & y_{2,3} & y_{2,4} \\ y_{3,1} & y_{3,2} & y_{3,3} & y_{3,4} \\ y_{4,1} & y_{4,2} & y_{4,3} & y_{4,4} \end{bmatrix}_{4\times4} \times \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}_{4\times4} \begin{bmatrix} I \\ J \\ K \\ 1 \end{bmatrix}_{4\times1} .$$

Therefore, the alignment problem can be expressed as follows: find $\vec{C}$, $\vec{D}_A$ and $\vec{D}_B$ such that

$$\forall\ (I, J, K) \in \text{iteration space of loop:} \begin{cases} [\vec{C}]_{4\times4} = [\vec{D}_A]_{4\times4}[\vec{F}_A]_{4\times4} \\ [\vec{C}]_{4\times4} = [\vec{D}_B]_{4\times4}[\vec{F}_B]_{4\times4} \end{cases} . \tag{Ex4}$$

The system of equations (Ex4) can be converted into the following matrix equation:

$$[\vec{C} \quad \vec{D}_A \quad \vec{D}_B]_{4\times12} \begin{bmatrix} 1 & 1 \\ -\vec{F}_A & 1 \\ 0 & -\vec{F}_B \end{bmatrix}_{12\times8} = \begin{bmatrix} 0 & \cdots & 0 \\ 0 & \cdots & 0 \\ 0 & \cdots & 0 \\ 0 & \cdots & 0 \end{bmatrix}_{4\times8} , \tag{Ex5}$$

where **I** and **0** are, respectively, a 4×4 identity matrix and a 4×4 zero matrix.

According to the method above, a solution matrix of (Ex5) is:

$$[\vec{C} \quad \vec{D}_A \quad \vec{D}_B]_{4\times12} = \begin{bmatrix} 1 & 1 & 1 & 1 & \frac{2}{3} & \frac{2}{3} & \frac{2}{3} & -\frac{1}{3} & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}_{4\times12} ,$$

which gives us:

$$[\vec{C}]_{4\times4} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}_{4\times4} , \quad [\vec{D}_A]_{4\times4} = \begin{bmatrix} \frac{2}{3} & \frac{2}{3} & \frac{2}{3} & -\frac{1}{3} \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}_{4\times4} , \quad \text{and}$$

$$[\vec{D}_B]_{4\times4} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}_{4\times4} .$$

Therefore, we can obtain the result of computational and data maps as follows

$$[\vec{C}]_{4\times4} \begin{bmatrix} I \\ J \\ K \\ 1 \end{bmatrix}_{4\times1} = \begin{bmatrix} I+J+K+1 \\ I+J+K+1 \\ 0 \\ 0 \end{bmatrix}_{4\times1} ,$$

$$[\vec{D}_A]_{4\times4}[\vec{F}_A]_{4\times4}\begin{bmatrix} I \\ J \\ K \\ 1 \end{bmatrix}_{4\times1} = \begin{bmatrix} I+J+K+1 \\ I+J+K+1 \\ 0 \\ 0 \end{bmatrix}_{4\times1} \quad \text{and}$$

$$[\vec{D}_B]_{4\times4}[\vec{F}_B]_{4\times4}\begin{bmatrix} I \\ J \\ K \\ 1 \end{bmatrix}_{4\times1} = \begin{bmatrix} I+J+K+1 \\ I+J+K+1 \\ 0 \\ 0 \end{bmatrix}_{4\times1} .$$

Iteration (I, J, K) is mapped to virtual processor $I+J+K+1$, and arrays A and B are also mapped to the same virtual processor. The **Align** statement provided by Fortran D is applied towards mapping arrays onto virtual processors. Arrays mapped to the same virtual processor are automatically aligned with each other. We employ the **Align** statement to describe the alignment relation of both arrays A and B. The **Align** statement is represented as

Align $A(I+J+K)$ with $T(I+J+K+1)$

Align $B(I+J+K+1)$ with $T(I+J+K+1)$,

where the virtual processors are assumed to be organized as a one-dimensional grid T.

3.2. *Array Reference function with quadratic subscripts*

It is assumed that there are m statements and q arrays with linear references in a general loop. The general loop is presumed to contain 2 common loops. A reference function to an array A_e for $1 \le e \le q$ in a general loop is $F_e = a_{e,1}I^2 + b_{e,1}I + c_{e,1}J + d_{e,1}$, where I and J are index variables of the general loop and $a_{e,1}, b_{e,1}, c_{e,1}, d_{e,1}$ are all coefficients. It is postulated that M is a 4-component column vector and M is represented as $(I^2, I, J, 1)^T$, where T is transposition operator. If **i** is an iteration vector in the iteration space of a general loop, the alignment constraints require that the processor performing iteration **i** must own $A_e(F_eM)$ for $1 \le e \le q$. The alignment constraint for the iteration space of a general loop can be expressed formally as

$$[\vec{C}]_{4\times4}[M]_{4\times1} = \begin{bmatrix} c_{1,1} & c_{1,2} & c_{1,3} & c_{1,4} \\ c_{2,1} & c_{2,2} & c_{2,3} & c_{2,4} \\ c_{3,1} & c_{3,2} & c_{3,3} & c_{3,4} \\ c_{4,1} & c_{4,2} & c_{4,3} & c_{4,4} \end{bmatrix}_{4\times4} \begin{bmatrix} I^2 \\ I \\ J \\ 1 \end{bmatrix}_{4\times1} ,$$

where $\vec{C}$ is a 4×4 matrix. Similarly, the alignment constraint for an array A_e for $1 \le e \le q$ in the general loop can also be, respectively, represented as

$$
\begin{aligned}
&[\vec{D}_{A_e}]_{4\times 4}[\vec{F}_{A_e}]_{4\times 4}[M]_{4\times 1} \\
&= \begin{bmatrix} w_{1,1} \\ w_{2,1} \\ w_{3,1} \\ w_{4,1} \end{bmatrix}_{4\times 1} * \left[a_{e,1}I^2 + b_{e,1}I + c_{e,1}J + d_{e,1}\right]_{1\times 1} + \begin{bmatrix} w_{1,2} \\ w_{2,2} \\ w_{3,2} \\ w_{4,2} \end{bmatrix}_{4\times 1} \\
&+ \begin{bmatrix} w_{1,3} \\ w_{2,3} \\ w_{3,3} \\ w_{4,3} \end{bmatrix}_{4\times 1} * \left[I^2 + I + J + 1\right] + \begin{bmatrix} w_{1,4} \\ w_{2,4} \\ w_{3,4} \\ w_{4,4} \end{bmatrix}_{4\times 1} * \left[I^2 + I + J + 1\right]_{1\times 1} \\
&= \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & w_{1,4} \\ w_{2,1} & w_{2,2} & w_{2,3} & w_{2,4} \\ w_{3,1} & w_{3,2} & w_{3,3} & w_{3,4} \\ w_{4,1} & w_{4,2} & w_{4,3} & w_{4,4} \end{bmatrix}_{4\times 4} \begin{bmatrix} a_{e1} & b_{e,1} & c_{e,1} & d_{e,1} \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}_{4\times 4} \begin{bmatrix} I^2 \\ I \\ J \\ 1 \end{bmatrix}_{4\times 1}.
\end{aligned}
$$

Therefore, the alignment problem can be expressed as follows: find $\vec{C}$ and $\vec{D}_{A_e}$ such that

$$
\begin{aligned}
&\forall\ (I, J) \in \text{ iteration space of loop: } \left[\vec{C}\right]_{4\times 4}[M]_{4\times 1} \\
&\quad = [\vec{D}_{A_e}]_{4\times 4}[\vec{F}_{A_e}]_{4\times 4}[M]_{4\times 1}. \qquad \text{(Ex6)}
\end{aligned}
$$

To cancel the column vector M from both sides of equations, we need the following lemma. Lemma 3.3 are an extension of Lemma 3.3 in the method proposed by Chu et al. in [7].

Lemma 3.3 Let Q, R and S be $q \times 1$ matrices, and let t, y and z be, respectively, a q-component column vector and scalar variables. Then

$$
\forall\ y\ z\left[Q \quad R \quad S \quad t\right]_{q\times 4} \begin{bmatrix} y^2 \\ y \\ z \\ 1 \end{bmatrix}_{4\times 1} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}_{q\times 1} \Leftrightarrow Q = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}_{q\times 1},
$$

$$
R = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}_{q\times 1}, \ S = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}_{q\times 1} \text{ and } t = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}_{q\times 1}.
$$

Proof: ($\Rightarrow$) In particular, we can let y and z be equal to zero. This gives us:

$$
\left[Q \quad R \quad S \quad t\right]_{q\times 4} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}_{4\times 1} = t = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}_{q\times 1}.
$$

So we can obtain t with a q-component zero vector. Similarly, we can let y be equal to zero. This gives us:

$$[Q \quad R \quad S \quad \mathbf{0}]_{q\times4}\begin{bmatrix}0\\0\\z\\1\end{bmatrix}_{4\times1} = [S]_{q\times1}\, z = \begin{bmatrix}0\\\vdots\\0\end{bmatrix}_{q\times1},$$

where $\mathbf{0}$ is a q-component zero vector. So we can obtain S with a q-component zero vector. Now, for any y and z:

$$\begin{aligned}[Q \quad R \quad \mathbf{0} \quad \mathbf{0}]_{q\times4}\begin{bmatrix}y^2\\y\\z\\1\end{bmatrix}_{4\times1} &= [Q \quad R]_{q\times2}\begin{bmatrix}y^2\\y\end{bmatrix}_{2\times1}\\ &= y\times[Q \quad R]_{q\times2}\begin{bmatrix}y\\1\end{bmatrix}_{2\times1} = \begin{bmatrix}0\\\vdots\\0\end{bmatrix}_{q\times1},\end{aligned}$$

where $\mathbf{0}$ is a q-component zero vector. According to Lemma 3.1, we can obtain

$$Q = \begin{bmatrix}0\\\vdots\\0\end{bmatrix}_{q\times1} \text{ and } R = \begin{bmatrix}0\\\vdots\\0\end{bmatrix}_{q\times1}.$$

We thus can conclude

$$\begin{aligned}\forall\, y\, z[Q \quad R \quad S \quad t]_{q\times4}\begin{bmatrix}y^2\\y\\z\\1\end{bmatrix}_{4\times1} = \begin{bmatrix}0\\\vdots\\0\end{bmatrix}_{q\times1} \Rightarrow Q = \begin{bmatrix}0\\\vdots\\0\end{bmatrix}_{q\times1},\\ R = \begin{bmatrix}0\\\vdots\\0\end{bmatrix}_{q\times1},\; S = \begin{bmatrix}0\\\vdots\\0\end{bmatrix}_{q\times1} \text{ and } t = \begin{bmatrix}0\\\vdots\\0\end{bmatrix}_{q\times1}.\end{aligned} \tag{7}$$

($\Leftarrow$) Because

$$Q = \begin{bmatrix}0\\\vdots\\0\end{bmatrix}_{q\times1}, R = \begin{bmatrix}0\\\vdots\\0\end{bmatrix}_{q\times1},\; S = \begin{bmatrix}0\\\vdots\\0\end{bmatrix}_{q\times1} \text{ and } t = \begin{bmatrix}0\\\vdots\\0\end{bmatrix}_{q\times1},$$

for any y and z we can acquire

$$[Q \quad R \quad S \quad t]_{q\times4}\begin{bmatrix}y^2\\y\\z\\1\end{bmatrix}_{4\times1} = \begin{bmatrix}0\\\vdots\\0\end{bmatrix}_{q\times1}. \tag{8}$$

According to (7) and (8), therefore, we can infer

$$\forall\, y\, z \begin{bmatrix} Q & R & S & t \end{bmatrix}_{q\times 4} \begin{bmatrix} y^2 \\ y \\ z \\ 1 \end{bmatrix}_{4\times 1} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}_{q\times 1} \Leftrightarrow Q = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}_{q\times 1},$$

$$R = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}_{q\times 1}, \; S = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}_{q\times 1} \text{ and } t = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}_{q\times 1}. \qquad \blacksquare$$

Using Lemma 3.3, (Ex6) can be rewritten as

$$[\vec{C}]_{4\times 4} = [\vec{D}_{A_e}]_{4\times 4}[\vec{F}_{A_r}]_{4\times 4}. \tag{Ex7}$$

The system of equations (Ex7) can be converted into the following matrix equation:

$$\begin{bmatrix} \vec{C} & \vec{D}_{A_1} & \cdots & \vec{D}_{A_q} \end{bmatrix}_{4\times(4\times q+4)} \begin{bmatrix} 1 & \cdots & I \\ -\vec{F}_{A_l} & \ddots & 0 \\ \vdots & \vdots & 0 \\ 0 & 0 & -\vec{F}_{A_q} \end{bmatrix}_{(4\times q+4)\times(4\times q)}$$

$$= \begin{bmatrix} 0 & \cdots & 0 \\ 0 & \cdots & 0 \\ 0 & \cdots & 0 \end{bmatrix}_{4\times(4\times q)}, \tag{Ex8}$$

where **1** and **0** are, respectively, a 3×3 identity matrix and a 3×3 zero matrix.

Consider the following loop with two one-dimensional arrays with quadratic subscripts in Figure 2. The loop is used to illustrate our points. If **i** is an iteration vector in the iteration space of the loop, the alignment constraints require the processor performing iteration **i** to own $A(F_1 M)$ and $B(F_2 M)$, where F_1 and F_2 and are the following matrices:

$$F_1 = [1 \quad -1 \quad 1 \quad 0]_{1\times 4} \text{ and } F_2 = [1 \quad -1 \quad 1 \quad 0]_{1\times 4}.$$

```
do I = 1, 3
    do J = 1, 3
        A(I^2-I+J) = B(I^2-I+J)
    enddo
enddo
```

Figure 2. Array references with quadratic subscripts.

The alignment constraints for the iteration space of the loop can be expressed formally as

$$[\vec{C}]_{4\times4}[M]_{4\times1} = \begin{bmatrix} c_{1,1} & c_{1,2} & c_{1,3} & c_{1,4} \\ c_{2,1} & c_{2,2} & c_{2,3} & c_{2,4} \\ c_{3,1} & c_{3,2} & c_{3,3} & c_{3,4} \\ c_{4,1} & c_{4,2} & c_{4,3} & c_{4,4} \end{bmatrix}_{4\times4} \begin{bmatrix} I^2 \\ I \\ J \\ 1 \end{bmatrix}_{4\times1},$$

where $\vec{C}$ is a 4×4 matrix. Similarly, the alignment constraints for the two arrays A and B can also be, respectively, represented as

$$[\vec{D}_A]_{4\times4}[\vec{F}_A]_{4\times4}[M]_{4\times1} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & w_{1,4} \\ w_{2,1} & w_{2,2} & w_{2,3} & w_{2,4} \\ w_{3,1} & w_{3,2} & w_{3,3} & w_{3,4} \\ w_{4,1} & w_{4,2} & w_{4,3} & w_{4,4} \end{bmatrix}_{4\times4} \times \begin{bmatrix} 1 & -1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}_{4\times4} \begin{bmatrix} I^2 \\ I \\ J \\ 1 \end{bmatrix}_{4\times1}, \text{ and}$$

$$[\vec{D}_B]_{4\times4}[\vec{F}_B]_{4\times4}[M]_{4\times1} = \begin{bmatrix} y_{1,1} & y_{1,2} & y_{1,3} & y_{1,4} \\ y_{2,1} & y_{2,2} & y_{2,3} & y_{2,4} \\ y_{3,1} & y_{3,2} & y_{3,3} & y_{3,4} \\ y_{4,1} & y_{4,2} & y_{4,3} & y_{4,4} \end{bmatrix}_{4\times4} \times \begin{bmatrix} 1 & -1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}_{4\times4} \begin{bmatrix} I^2 \\ I \\ J \\ 1 \end{bmatrix}_{4\times1}.$$

Therefore, the alignment problem can be expressed as follows: find $\vec{C}$, $\vec{D}_A$ and $\vec{D}_B$ such that

$$\forall\ (I, J) \in \text{iteration space of loop:} \begin{cases} [\vec{C}]_{4\times4} = [\vec{D}_A]_{4\times4}[\vec{F}_A]_{4\times4} \\ [\vec{C}]_{4\times4} = [\vec{D}_B]_{4\times4}[\vec{F}_B]_{4\times4} \end{cases}. \tag{Ex9}$$

The system of equations (Ex9) can be converted into the following matrix equation:

$$[\vec{C} \quad \vec{D}_A \quad \vec{D}_B]_{4\times12} \begin{bmatrix} \mathbf{I} & \mathbf{I} \\ -\vec{F}_A & \mathbf{I} \\ \mathbf{0} & -\vec{F}_B \end{bmatrix}_{12\times8} = \begin{bmatrix} 0 & \cdots & 0 \\ 0 & \cdots & 0 \\ 0 & \cdots & 0 \\ 0 & \cdots & 0 \end{bmatrix}_{4\times8}, \tag{Ex10}$$

where $\mathbf{I}$ and $\mathbf{0}$ are, respectively, a 4×4 identity matrix and a 4×4 zero matrix.

According to the method stated in Section 3.1, a solution matrix of (Ex10) is:

$$[\vec{C} \quad \vec{D}_A \quad \vec{D}_B]_{4\times 12} = \begin{bmatrix} 2 & 0 & 2 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 3 & 1 & 3 & 3 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{bmatrix}_{4\times 12},$$

which gives us:

$$[\vec{C}]_{4\times 4} = \begin{bmatrix} 2 & 0 & 2 & 1 \\ 1 & 1 & 1 & 1 \\ 3 & 1 & 3 & 3 \\ 0 & 0 & 0 & 0 \end{bmatrix}_{4\times 4}, \quad [\vec{D}_A]_{4\times 4} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}_{4\times 4},$$

$$\text{and} \quad [\vec{D}_B]_{4\times 4} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & -1 & 1 \end{bmatrix}_{4\times 4}.$$

Therefore, we can obtain the computational and data maps as follows

$$[\vec{C}]_{4\times 4} \begin{bmatrix} I^2 \\ I \\ J \\ 1 \end{bmatrix}_{4\times 1} = \begin{bmatrix} 2I^2 + 2J + 1 \\ I^2 + I + J + 1 \\ 3I^2 + I + 3J + 3 \\ 0 \end{bmatrix}_{4\times 1},$$

$$[\vec{D}_A]_{4\times 4} [\vec{F}_A]_{4\times 4} \begin{bmatrix} I^2 \\ I \\ J \\ 1 \end{bmatrix}_{4\times 1} = \begin{bmatrix} 2I^2 + 2J + 1 \\ I^2 + I + J + 1 \\ 3I^2 + I + 3J + 3 \\ 0 \end{bmatrix}_{4\times 1} \quad \text{and}$$

$$[\vec{D}_B]_{4\times 4} [\vec{F}_B]_{4\times 4} \begin{bmatrix} I^2 \\ I \\ J \\ 1 \end{bmatrix}_{4\times 1} = \begin{bmatrix} 2I^2 + 2J + 1 \\ I^2 + I + J + 1 \\ 3I^2 + I + 3J + 3 \\ 0 \end{bmatrix}_{4\times 1}.$$

Iteration (I, J) is mapped to virtual processor $2I^2 + 2J + 1$, and arrays A and B are also mapped to the same virtual processor. We employ the **Align** statement to describe the alignment relation both arrays A and B. The **Align** statement is represented as

Align $A(I^2 - I + J)$ with $T(2I^2 + 2J + 1)$

Align $B(I^2 - I + J)$ with $T(2I^2 + 2J + 1)$,

where the virtual processors are assumed to be organized as a one-dimensional grid T.

3.3. *Time complexity*

The proposed method reduces the problem of determining data and computation placement to the standard linear algebra problem of determining a basis for the null space of a matrix. It includes three main phases. The first phase is to set the value of each element in the matrix of constraints V. The second phase is to apply Gaussian elimination with full pivoting [1] to compute a basis U^T for the null-space of V, where T is a transposition operator. The third phase is to extract the solution matrices from U.

It is postulated that V is an M^*N matrix and its rank is r, where r is at most the minimal value between M and N. The worst-case time complexity of setting the value for every element of V is $O(M^*N)$. Gaussian elimination with full pivoting is employed towards figuring out a basis U^T for the null-space of V, and requires $O(r^*M^*N + r^3)$ *arithmetic* operations [1]. Therefore, the worst-case time complexity of determining a basis U^T for the null-space of V is immediately inferred to be $O(r^*M^*N + r^3)$. It is clear from (6) that $U = H(r + 1 : M, 1 : M)$. The worst-case time complexity of extracting the solution matrices from U thus is $O(M^2 - r^*M)$. Hence, the worst-case time complexity of the presented method is $O(M^*N + r^*M^*N + r^3 + M^2 - r^*M)$, similar to that of the method proposed by Bau et al. in [4].

4. Experimental results

Using hand computation, the proposed method was tested experimentally on codes from two numerical packages. Parallel Loop and Vector Loop [2, 3]. The codes included seven nested loops consisting of array access references with linear subscripts in three loop index variables or quadratic subscripts. The results are shown in Table 1. It is very clear from Table 1 that the proposed method solves the data alignment problem for array reference with linear subscripts in three loop index variables and quadratic subscripts.

5. Conclusions

We have presented a technique to improve data locality for arrays with complex subscripts in loop nests. Our technique can reduce the problem of determining data and computation placement to the standard linear algebra problem of determining a

Table 1. Results of application of proposed method on communication-free alignment of the seven tested nested loops

Benchmark	The number of loop tested	The number of communication-free alignment obtained
Parallel loop	1	1
Vector loop	6	2

basis for the null space of a matrix. It is therefore suggested that, depending on the application domain, our method can be used together with the methods proposed by Bau et al. [4] and Chu et al. [7] to enhance data locality in loop nests. If the corresponding lemmas can be derived with correct proofs, it would be possible to extend our method to deal with loops with linear subscripts in four or more loop index variables or quadratic subscripts.

References

1. Jack Edmonds. Systems of distinct representative and linear algebra, *Journal of Research of National Bureau of Standards,* Sect. B, 71(4):241–245, 1967.
2. J. Dongarra, M. Furtney, S. Reinhardt, and J. Russell. Parallel loops—a test suite for parallelizing compilers: Description and example results, *Parallel Computing* 17:1247–1255, 1991.
3. David Levine, David Callahan, and Jack Dongarra. A comparative study of automatic vectorizing compilers, *Parallel Computing* 17:1223–1244, 1991.
4. David Bau, Induprakas Kodukula, Vladimir Kotlyar, Keshav Pingali, and Paul Stodghill. Solving alignment using elementary linear algebra. In: *Conference Record of the 7th Workshop on Languages and Compilers for Parallel Computing,* pp. 46–60, August 1994.
5. Michael Wolfe. *High Performance Compilers for Parallel Computing.* Addison-Wesley Publishing Company, Redwood City, 1996.
6. John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach, Second Edition.* Morgan Kaufmann Publishers, Inc. San Francisco, California, 1996.
7. Chih-Ping Chu, Weng-Long Chang, Iwen Chen, and Peng-Sheng Chen. Communication-free alignment for array references with linear subscripts in two loop index variables or quadratic subscripts. *Proceedings of the Second IASTED International Conference on Parallel and Distributed Computing and Networks (PDCN'98)*, Australia, pp. 571–576.
8. M. Kandemir, A. Choudhary, N. Shenoy, P. Banerjee, and J. Ramanujam. A hyper-plane based approach for optimizing spatial locality in loop nests. In *Proc. 12th ACM Int. Conf. Supercomputing*, July 1998.
9. M. Kandemir, J. Ramanujam, A. Choudhary, and P. Banerjee. A loop transformation algorithm based on explicit dada layout representation for optimizing locality. In *Proc. 11th International Workshop, LCPC'98,* Chapel Hill, NC, USA, August 1998.
10. A. W. Lam and M. S. Lam. Maximizing parallelism and minimizing synchronization with affine partitions. *Parallel Computing,* 24(3–4):445–475, 1998.
11. V. Boudet, F. Rastello, and Y. Robert. Alignment and distribution is NOT (always) NP-hard. Proceeding of 1998 International Conference on Parallel and Distributed Systems, 5(9):648–657, 1998.
12. M. Kandemir, A. Choudhary, N. Shenoy, P. Banerjee, and J. Ramanujam. A linear algebra framework for automatic determination of optimal data layouts. *IEEE Transactions on Parallel and Distributed Systems,* 21(12):115–135, 1999.
13. A. W. Lam, G. I. Cheong, and M. S. Lam. An affine partitioning algorithm to maximize parallelism and minimize communication. *13th ACM International Conference on Supercomputing*, Rhodes, Greece, pp. 228–237, June 1999.
14. K.-P. Shih, J.-P. Sheu, and C.-H. Huang. Statement-level communication-free partitioning techniques for parallelizing compilers. *The Journal of Supercomputing*, 15(3):243–269, 2000.