A New Approach for Finding Loop Transformation Matrices

Hua Lin Mi Lu

Electrical Engineering Department
Texas A&M University, College Station, TX 77843

Jesse Z. Fang

Hewlett-Packard Lab.
P.O. Box 10490, Palo Alto, CA 94303

Abstract

Traditional approach for generating loop transformation matrix, which is based upon the computation of distance vectors or direction vectors, does not work for those nested loops whose distance vectors are uncomputable and direction vectors contain no useful information. In this paper, we present a new technique for generating transformation matrix that is based upon identifying certain types of linear equations or inequalities of distance vectors. Two issues related to this technique are discussed in this paper:

1) Given a nested loop how to identify these linear equations or inequalities;
2) Given a such linear equation or inequality how to generate a legal and unimodular transformation matrix for the purpose of loop parallelization.

1 Introduction

Loop transformation, such as loop interchange, reversal and skewing, is one of the most important techniques for automatically transforming sequential programs into parallel forms, which are extremely desired for the beneficiary of sequential programs for application areas such as aerospace, defense, geophysics and other intensive task [1][3][10][13] and it also has important applications in enhancing the performance of memory hierarchies of both sequential and parallel machines [6][12][4]. Traditional approach for finding the transformation matrix for a given nested loop is based on identifying distance vectors or direction vectors of the loop [2][12]. Unfortunately, this approach may not work for some nested loops. One of the main reasons is that the number of distance vectors of a nested loop may be of infinite, thus distance vectors may be uncomputable; on the other hand, direction vectors may not contain useful information, thus missing the chance of applying loop transformation. To illustrate this point, consider the nested loop in Example 1.

Example 1:

\[
\begin{align*}
\text{do } I_1 &= 0, n \\
\text{do } I_2 &= 0, n \\
\text{do } I_3 &= 0, n \\
\text{enddo}
\end{align*}
\]

Many loop-carried data dependences exist in this nested loop when \( n \geq 1 \). Suppose \( n = 4 \) for instance. Iteration \((0, 2, 0)\) is flow dependent on iteration \((0, 0, 1)\), and the corresponding distance vector is \((0, 2, -1)\); iteration \((1, 0, 2)\) is anti-dependent on iteration \((0, 4, 0)\), and the corresponding distance vector is \((1, -4, 2)\). Notice that the number of distinct distance vectors increases with \( n \). Therefore, distance vectors are not capable to describe the dependence information. On the other hand, using direction vectors to approximate data dependence, we can find the following direction vectors: \((=, <, >), (>, =, =), (<, <, >), \) and \((>, <, <)\). Apparently, there is no way to tell based on these information whether a linear transformations matrix exists to make the middle loop or the outermost loop of the transformed code parallelizable. Nevertheless, such transformations do exist. For example, through the following transformation:

\[
\begin{pmatrix}
I'_1 \\
I'_2 \\
I'_3
\end{pmatrix} = \begin{pmatrix}
5 & 1 & 0 \\
1 & 0 & 0 \\
0 & 0 & 1
\end{pmatrix}\begin{pmatrix}
I_1 \\
I_2 \\
I_3
\end{pmatrix}
\]

the original loop can be transformed into this form:

\[
\begin{align*}
\text{do } I'_1 &= \ldots \\
\text{do } I'_2 &= \ldots \\
\text{do } I'_3 &= \ldots \\
S[I'_1 - 4I'_2 + I'_3 + 1, 2I'_1 - 9I'_2 + 3I'_3 + 1] &= \cdots \\
\cdots &= h(S[I'_1 - 4I'_2 + I'_3, 2I'_1 - 9I'_2 + 3I'_3]) \\
\text{enddo}
\end{align*}
\]

Since there is no data dependence carried by loop \( I'_3 \), it can be parallelized. Similarly, through the following transformation matrix:

\[
\begin{pmatrix}
I'_1 \\
I'_2 \\
I'_3
\end{pmatrix} = \begin{pmatrix}
0 & 1 & 2 \\
1 & 0 & 0 \\
0 & 1 & 1
\end{pmatrix}\begin{pmatrix}
I_1 \\
I_2 \\
I_3
\end{pmatrix}
\]

the outermost loop of the transformed code, can be parallelized.

To deal with this type of situation, we present a new technique in this paper for finding legal linear transformation matrices for a given nested loop, which does
not require computing distance vectors and direction vectors. The basic idea of our approach is to identify linear equations or inequalities of distance vectors (called \( \alpha \)-equations and \( \alpha \)-inequalities, respectively) and construct desired transformation matrices based on these equations or inequalities. Consider Example 1, for instance. Under our approach, we first discover the linear \( \alpha \)-equation \( \Sigma_1 \) from \( \alpha \)-equations and inequalities. The rest of this paper is organized as follows. Section 2 discusses program model and background. Section 3 focuses on how to generate \( \alpha \)-equation from index functions, ignoring the constrains imposed by loop limits; Section 4 discusses how to find loop transformation matrix based on the information provided by \( \alpha \)-equations or inequalities to generate transformation matrices. We should point out that there are loops whose distance vectors may not satisfy any linear equations or inequalities, and our approach would not work for these loops.

In this paper we discuss two important issues regarding this approach: First, given a nested loop how to find proper \( \alpha \)-equations or inequalities if such equations or inequalities are exist. Second, how to generate a desired linear transformation matrix based on these equations and inequalities. The rest of this paper is organized as follows. Section 2 discusses program model and background. Section 3 focuses on how to generate \( \alpha \)-equation from index functions, ignoring the constrains imposed by loop limits; Section 4 discusses how to find loop transformation matrix based on the information provided by \( \alpha \)-equations. In section 5 we continue the topic of section 3 and section 4 considering both index functions and loop limits. Due to the limit of space, we omit all the proofs; readers may refer to [8].

2 Preliminaries

For the sake of simplicity, we restrict our attention to perfect nested sequential loops, though this is not a necessary condition for applying our approach. The general structure of an \( r \)-deep nested loop with \( m \)-dimensional array references has the following form:

\[
\text{do } I_1 = L_1(n), U_1(n) \\
\quad \ldots \\
\quad \text{do } I_r = L_r(I_1, \ldots, I_{r-1}, n), U_r(I_1, \ldots, I_{r-1}, n) \\
\quad \text{A}(f_1(I), \ldots, f_m(I)) = \ldots \\
\quad S_1 \\
\quad \quad \quad = A(g_1(I), \ldots, g_n(I)) \\
\quad \text{enddo} \\
\quad \ldots
\]

enddo

The iteration space, denoted by \( R \) in this paper, is defined by lower/upper limits functions \( L_k \)'s and \( U_k \)'s. They are functions of loop index \( i \) and loop-invariant variable \( n \) which is used to control the size of iteration space. We assume that index functions \( f_k(I) \)'s and \( g_k(I) \)'s are linear.

For a pair of dependent iterations, say \( i \) and \( i' \) with \( i' > i \), distance vector \( d \) is defined as \( d = i' - i \). There exists data dependence carried by loop \( l \) if there exists a distance vector \( d = (d_1, d_2, \ldots, d_r) \) such that \( d_j = 0 \) for \( 1 \leq j \leq l - 1 \) and \( d_l > 0 \). In this paper by \( d > 0 \) we mean the distance vector satisfying this condition.

By conducting linear transformations over iteration space, a nested loop can be converted into a new one, called transformed nested loop. Let matrix \( T \) of size \( r \times r \) be the transformation matrix for a given nested loop. Each iteration \( i_0 \) in the new iteration space \( R_{n_{\text{new}}} \) is a mapping from an iteration \( i \) in \( R \) through \( i_{n_{\text{new}}} = Ti_0 \). Readers may refer to [2] and [12] for more detail about linear loop transformation. A transformation \( T \) is legal if the transformed nested loop preserves the data dependency of the original loop, which is essential for guaranteeing the correctness of the output of the transformed loop.

Unimodular transformation, in which transformation matrix \( T \) is an unimodular matrix (i.e., \( det(T) = \pm 1 \)), is in particular important because it guarantees an one-to-one mapping and has the property that \( Ti_0 \) is an integer vector if \( i \) is an integer vector. We will concentrate on unimodular transformation in this paper.

A loop of a nested loop is parallelizable if the iterations of this loop can be executed simultaneously without violating the partial order imposed by the data dependence. Clearly, a loop is parallelizable if and only if there is no dependences carried by that loop. In other words, for an \( r \)-deep nested loop, its \( k \)-th loop \( L_k \) is parallelizable if there is no \( d \) with \( d > 0 \).

3 Preparing \( \alpha \)-equations for loop transformations

3.1 \( \alpha \)-equations and inequalities

Let \( D_f \) and \( D_a \) denote the set of distance vectors of flow-dependence and the set of distance vectors of anti-dependence of a nested loop, \( L \), respectively. Linear equations (or inequalities) \( S_f \) and \( S_a \) are called \( \alpha \)-equations \( \text{(or } \alpha \text{-inequalities)} \) of \( L \) if any \( d \in D_f \) satisfies \( S_f \) and any \( d \in D_a \) satisfies \( S_a \). Further, \( \alpha \)-equations (inequalities) are divided into two types: those in which the value of constant term is of nonzero are considered as Type I and those in which the value of constant term is of zero are considered as Type II. For instance, \( \alpha \)-equation \( d_1 + d_2 + d_3 = 1 \), \( d_1 + 2d_2 + 3d_3 = 1 \) and \( 2d_1 + d_2 + 2d_3 = 0 \) in Example 1 are of Type I while \( \alpha \)-equation \( d_1 + d_2 + d_3 = 0 \) is of

1 A vector \( x \) is lexicographically positive, denoted as \( x > 0 \), if there exists \( k \) such that \( x_k > 0 \) for any \( 1 \leq j \leq k-1 \). Vector \( y \) is lexicographically greater than vector \( x \), denoted as \( y > x \), if \( y - x \) is lexicographically positive.
Type II. Furthermore, we use Type $I_k$ and Type $II_k$ to denote those $\alpha$-equations (or inequalities) in which the coefficient of $d_k$ is of nonzero while the coefficients of $d_l$‘s, $k < l \leq r$, are of zero.

Generating $\alpha$-equations or inequalities for a given nested loop is the key of our technique. To find an $\alpha$-equation or inequality of a given nested loop, in general, we should consider both index functions and loop limits. The focus of this section will be on those nested loop whose $\alpha$-equations can be found from index functions; more complicated case in which $\alpha$-equations or inequalities can only be found based on both index functions and loop limits will be discussed later.

3.2 Generating a specific type of $\alpha$-equation

The technique we present below bases on manipulating the coefficients of index function. Consider an $r$-deep perfectly nested loop with $m$-dimensional array reference:

$$S[f_1, \ldots, f_m] = h(S[g_1, \ldots, g_m])$$

where

$$f_k(I) = a_k,0 + \sum_{j=1}^{k-1} a_{k,j}I_j$$

and

$$g_k(I) = b_k,0 + \sum_{j=1}^{k-1} b_{k,j}I_j$$

for $1 \leq k \leq m$. Let $A$ and $B$ denote the coefficients of index functions:

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,r} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,r} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,r} \end{bmatrix}$$

$$B = \begin{bmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,r} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,r} \\ \cdots & \cdots & \cdots & \cdots \\ b_{m,1} & b_{m,2} & \cdots & b_{m,r} \end{bmatrix}$$

Let $F$ denote the matrix of $A - B$, and $F_k$ denote matrix $(F; A_1, A_{r-1}; \cdots; A_k)$, where $A_j = (a_{1,j}, a_{2,j}, \ldots, a_{m,j})^T$. For the sake of simplicity, sometime by $F^{+1}$ we refer to $F$. Let $v$ denote column vector $(a_{1,0} - b_{1,0}, a_{2,0} - b_{2,0}, \ldots, a_{m,0} - b_{m,0})^T$, where $a_{1,0}, b_{1,0}, \ldots, a_{m,0}, b_{m,0}$ are the constant terms of the index functions. As an example, the coefficient matrices $v, F^1, F^2$ and $F^3$ for Example 2 are listed below:

<table>
<thead>
<tr>
<th>$v$</th>
<th>$F^1$</th>
<th>$F^2$</th>
<th>$F^3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\begin{bmatrix} 2 \ 1 \ 1 \ 1 \end{bmatrix}$</td>
<td>$\begin{bmatrix} 1 &amp; 1 &amp; 1 &amp; 2 &amp; 1 &amp; 2 \ 0 &amp; 1 &amp; 1 &amp; 1 &amp; 2 &amp; 3 \end{bmatrix}$</td>
<td>$\begin{bmatrix} 1 &amp; 1 &amp; 1 &amp; 1 &amp; 2 &amp; 1 \ 1 &amp; 1 &amp; 1 &amp; 2 &amp; 2 &amp; 2 \end{bmatrix}$</td>
<td>$\begin{bmatrix} 1 &amp; 1 &amp; 1 &amp; 2 \ 1 &amp; 1 &amp; 1 &amp; 1 \end{bmatrix}$</td>
</tr>
</tbody>
</table>

Example 2:

```plaintext```
```
do I1 = 0, n - 1
  do I2 = 0, n - 1
    do I3 = 0, n - 1
      A[I1 + I2 + I3 + 2, I1 + I2 + I3 + 1] = 
      3I1 + 2I2 + I3
    enddo
  enddo
enddo
```

Here is the conditions that test the existence of $\alpha$-equation of Type $I$ and Type $II$.

**Theorem 3.1:** Let $L$ be an $r$-deep nested loop with $m$-dimensional array reference. A Type $I_q$ $\alpha$-equation of $L$ with form $\sum_{j=1}^{r} c_{j}d_j = c_0$, where $1 \leq q \leq r$, can be found from index functions if rank($F^{q+1}$) < rank($F^q$) and rank($F^{q+1}$) < rank($F^q$; $v$) hold. A Type $II_q$ $\alpha$-equation of $L$ with form $\sum_{j=1}^{r} c_{j}d_j = 0$, where $1 \leq q \leq r$, can be found from index functions if rank($F^{q+1}$) < rank($F^q$) and rank($F^{q+1}$) = rank($F^q$; $v$) hold. Furthermore, the coefficients $c_j$’s can be computed as follows: $c_j = e^T A_j$ for $1 \leq j < q$, $c_0 = e^T v$, where $e^T$ is a row vector such that $e^T F = 0$, $e^T v \neq 0$, $e^T A_j = 0$ for $q + 1 \leq j \leq r$, and $e^T A_r = 0$.

We now discuss the computation issue. Echelon reduction [3] can be used to compute the ranks of coefficient matrices and vector $e^T$ as well. For a given $m_1 \times m_2$ matrix $W$, let $w_i$, be the first nonzero element in the $k$-th row if this row is not a zero row. $W$ is an echelon matrix if there exists $q$ such that: 1) rows 1 through $q$ are nonzero rows and rows $q + 1$ through $m_2$ are zero rows; 2) $l_1 < l_2 < \cdots < l_q$. Echelon reduction, given a $m_1 \times m_2$ matrix $P$, finds an $m_1 \times m_1$ unimodular matrix $Q$ and an $m_1 \times m_2$ echelon matrix $W$ such that $QP = W$. Such reduction can be done by applying a series of row elementary operations to $P$ and an $m_1 \times m_1$ unit matrix, and the transformed matrix of the unit matrix is $Q$.

The rank of $F^q$ can be efficiently computed by performing echelon reduction on $F^q$. Let $E(F^q)$ be the echelon form of $F^q$. Since the first $2r - q$ columns of $F^q$ form matrix $F^{q+1}$, the first $2r - q$ columns of $E(F^q)$ form the echelon form of $F^{q+1}$. So, the rank of $F^{q+1}$ can be obtained directly from $E(F^q)$. The computation of rank($F^{q+1}$; $v$) and $e^T$ is a by-product of the echelon reduction: When doing the reduction on $F^q$ we apply the same elementary row operations to $v$. Let $U$ and $v'$ be the corresponding transformed matrix of the unit matrix $U$ and $v$. rank($F^{q+1}$; $v'$) can be obtained by simply checking the rank of ($E(F^{q+1})$; $v'$) because elementary row operations do not change the rank of a matrix. $e^T$ can be obtained from $U$ as follows: Suppose that rank($F^{q+1}$) < rank($F^q$) = $l$, we choose the $l$-th row of $U^T$, $u_i$, as $e^T$. Recall Theorem 3.1, $e^T$ is required such that $e^T F = 0$, $e^T A_j \neq 0$, and $e^T A_j = 0$ for $q + 1 \leq j \leq r$. The $l$-th row of $U^T$ satisfies these conditions. Indeed, since rank($F^q$) = $l$ and rank($F^{q+1}$) < $l$, we have $u_i F \neq 0$ and $u_i A_j = 0$ for $q + 1 \leq j \leq r$. Since $F^{q+1} = (F; A_r; \cdots; A_{q+1})$, we have $u_i F = 0$ and $u_i A_j = 0$ for $q + 1 \leq j \leq r$. Since $F^q = (F^{q+1}; A_q)$, we have $u_i A_q \neq 0$.

Let us use Example 2 to illustrate the process of the computation described above. Suppose we want
know if an \( a \)-equation of Type \( I_3 \) can be found from the index functions. According to Theorem 3.1, we need
to check if \( \text{rank}(\mathbf{F}^4) < \text{rank}(\mathbf{F}^3) \) and \( \text{rank}(\mathbf{F}^4) < \text{rank}(\mathbf{F}^4; \nu) \), where \( \mathbf{F}^4 \) is denoted to \( \mathbf{F} \) as mentioned before; moreover, if these conditions hold we need to
find \( \mathbf{e}^i \) and compute the coefficients of the \( a \)-equation.

The reader should notice that in the following process these two jobs are to be done in one pass.

Initially, we have

\[
\mathbf{U} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}, \quad \mathbf{v} = \begin{bmatrix} 2 \\ 1 \\ 1 \\ 0 \end{bmatrix}
\]

After performing a series of elementary row operations on \( \mathbf{F}^3 \) as well as on \( \mathbf{U} \) and \( \mathbf{v} \), we obtain \( E(\mathbf{F}^3) \), \( \mathbf{U}' \) and \( \mathbf{v}' \):

\[
\mathbf{U}' = \begin{bmatrix} 1 & 0 & 1 & 0 \\ -1 & 0 & 1 & 0 \\ 2 & -2 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}, \quad \mathbf{E}(\mathbf{F}^3) = \begin{bmatrix} 1 & 1 & 1 & 2 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{v}' = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix}
\]

The first three columns of \( E(\mathbf{F}^3) \) form the echelon form of matrix \( \mathbf{F}^4 \). It is clear \( \text{rank}(\mathbf{F}^3) = 2 \),
\( \text{rank}(\mathbf{F}^4) = 1 \), and \( \text{rank}(\mathbf{F}^4; \nu) = 2 \). Hence, an \( a \)-equation of Type \( I_3 \) can be found from index functions.

We now compute the coefficients of this \( a \)-equation. Since \( \text{rank}(\mathbf{F}^3) = 2 \) we choose the second row of \( \mathbf{U} \) as \( \mathbf{e}^i \), which is \((-1, 0, 1)\). Recall

\[
\mathbf{A}_1 = \begin{bmatrix} 2 \\ 3 \end{bmatrix}, \quad \mathbf{A}_2 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \quad \mathbf{A}_3 = \begin{bmatrix} 2 \\ 1 \end{bmatrix}
\]

we have \( c_1 = \mathbf{e}^i \mathbf{A}_1 = 1 \), \( c_2 = \mathbf{e}^i \mathbf{A}_2 = 1 \), \( c_3 = \mathbf{e}^i \mathbf{A}_3 = -1 \) and \( c_0 = \mathbf{e}^i \mathbf{v} = -2 \). Hence, we have obtained an \( a \)-equation \( d_1 + d_2 - d_3 = 2 \) for flow dependence and \( a \)-equation \( d_1 + d_2 - d_3 = 0 \) for anti-dependence.

As for the time complexity of the above process, we notice that the largest matrix we may deal with is \( \mathbf{F}^4 \), which has a size of \( m \times 2r \). Therefore, it takes \( O(m^2r) \) time to find a specific type of \( a \)-equation. In applications, we may want to know all possible types of equation. An elegant algorithm is presented in [8] which does this with the same time complexity.

### 4 Finding transformation matrix through \( a \)-equation

In this section we discuss how to find transformation matrices for the purpose of loop parallelization given \( a \)-equations. In the follows we provide two theorems, one dealing with Type I \( a \)-equations and the other dealing with Type II \( a \)-equations.

**Theorem 4.1:** Let \( L \) be an \( r \)-deep nested loop with \( m \)-dimensional array reference, and \( \sum_{j=1}^r c_j d_j \pm c_0 = 0 \) is an \( a \)-equation of Type \( I_q \), where \( 1 \leq q \leq r \).

1. If \( q < r \), the innermost \( r - q \) loops of \( L \) are parallelizable;

2. If \( 2 \leq q \leq r \), through unimodular transformation matrix \( \mathbf{T} = \begin{bmatrix} \mathbf{U}_{q-2} & \mathbf{0} \\ \mathbf{0} & \mathbf{U}_{r-q} \end{bmatrix} \) the innermost \( r - q + 1 \) loops of the transformed code are parallelizable, where \( \mathbf{U}_k \) is a \( k \times k \) identity matrix and \( \mathbf{T}_0 = \begin{bmatrix} \mu & 1 \\ 1 & 0 \end{bmatrix} \) with \( \mu = [(c_{r-1} + c_0) / c_r] + 1 \) if \( c_0 / c_r > 0 \) or \( \mu = [-c_{r-1} / c_r] \).

**Theorem 4.2:** Let \( L \) be an \( r \)-deep nested loop with \( m \)-dimensional array reference, and \( \sum_{j=1}^r c_j d_j = 0 \) is the \( a \)-equation of Type \( II_q \), where \( 1 \leq q \leq r \).

1. The \( q \)-th loop of \( L \) is parallelizable;

2. The unimodular transformation matrix \( \mathbf{T} \) generated by a procedure called getT() which is shown below produces a parallelizable outermost loop.

In procedure getT() we assume \( c_r \neq 0 \); the case in which \( c_1, c_{i=1}, \ldots, c_r = 0 \) can be handled easily (we will show this through an example later). Before explaining how getT() works, we first introduce several functions used in getT(). Function \( \text{gcd}(x_1, x_2, \ldots, x_q) \) returns an integer, say \( v \), such that \( [v] \) is the greatest common divisor of integer numbers \( x_1, x_2, \ldots, x_q \) and that \( v \) is positive if the nonzero element of input with the smallest subscript is positive otherwise it is negative. As a special case, \( \text{gcd}(x_1) \) returns \( x_1 \). For example, \( \text{gcd}(2, 4, 6) \) returns 2, \( \text{gcd}(0, -3, 6) \) returns 3, and \( \text{gcd}(0, 3, -9) \) returns 3. Function \( \text{int}(z_1, z_2) \) takes integer numbers \( z_1 \) and \( z_2 \) as input and output integer \( (y_1, y_2) \) such that \( z_1 y_2 - z_2 y_1 = 1 \) if \( z_1 z_2 > 0 \) or \( z_1 y_2 - z_2 y_1 = 1 \) if \( z_1 z_2 < 0 \). For example, one possible return of \( \text{int}(2, 3) \) is \((1, 1)\); one possible return of \( \text{int}(2, 3, -7) \) is \((1, -2)\), and the return of \( \text{int}(2, 0, 1) \) is \((1, 0)\).

```plaintext
procedure getT(c_1, c_2, \ldots, c_q)
/* let \( k,j \) be the entry of \( \mathbf{T} \) on the \( k \)-th row and \( j \)-th column */
step 1.1 \( z \leftarrow \text{gcd}(c_1, c_2, \ldots, c_q) \);
step 1.2 \( k, j \leftarrow c_1 / z \) for \( 1 \leq j \leq r \);
step 2.1 \( k, j \leftarrow 0 \) for \( 1 \leq j \leq 2 - 2 \);
step 2.2 \( k, j \leftarrow \text{gcd}(c_{k+1}, c_k, \ldots, c_q) \);
step 2.3 \( k, j \leftarrow \text{gcd}(c_k, c_{k+1}, \ldots, c_q) \);
step 2.4 \( (y_1, y_2) \leftarrow \text{int}(z_1, z_2) \);
step 2.5 \( t_{k, 0} \leftarrow y_1 \) and \( t_{k, j} \leftarrow y_2 z^{1+j} \) for \( 1 \leq j \leq r \).
endfor
```

In illustration of getT(), we consider a nested loop that has the following \( a \)-equation:

\[
2d_1 + 3d_3 - 3d_4 + 6d_5 = 0
\]

and want to find a unimodular transformation \( \mathbf{T} \) that make the outermost loop parallelizable. In Step 1.1, we compute \( \text{gcd}(2, 0, 3, -3, 6) \) and get \( z = 1 \); following Step 1.2 we get the first row of \( \mathbf{T} \). We now go to the second stage, and consider \( k = 2 \) first. Since \( k - 2 = 0 \)
we do nothing in Step 2.1; in Step 2.2 we compute 
\( z_1 \leftarrow \gcd (2, 0, 3, -3, 6) = 1 \); in Step 2.3 compute 
\( z_2 \leftarrow \gcd (0, 3, -3, 6) = 1 \); in Step 2.4 compute 
\( (y_1, y_2) \leftarrow \text{int2}(2, 3) = (1, 1) \); then in Step 2.5 \( t_{3,1} \) is assigned 
with \( y_1 \) and \( t_{2,j} \)'s assigned with \( y_2 z_1 c_j / z_2 \) for \( 2 \leq j \leq 5 \), respectively, which are 0, 1, -1, and 2.

\[
\begin{pmatrix}
2 & 0 & 3 & -3 & 6 \\
1 & 0 & 1 & -1 & 2 \\
0 & 0 & 0 & 0 & 0
\end{pmatrix}
\]

Now consider \( k = 3 \). In Step 2.1 \( t_{3,1} \leftarrow 0 \); in Step 
2.2 \( z_1 \leftarrow \gcd (0, 3, -3, 6) = 3 \); in Step 2.3 \( z_2 \leftarrow \gcd (3, -3, 6) = 3 \); in Step 2.4 \( (y_1, y_2) \leftarrow \text{int2}(0, 1) = 
(1, 0) \); then in Step 2.5 \( t_{3,2} \) is assigned with 1, and 
\( t_{3,j} \)'s assigned with \( y_2 z_1 c_j / z_2 \) for \( 3 \leq j \leq 5 \), respectively, which are all zero's.

\[
\begin{pmatrix}
2 & 0 & 3 & -3 & 6 \\
1 & 0 & 1 & -1 & 2 \\
0 & 0 & 0 & 0 & 0
\end{pmatrix}
\]

Finally, \( k = 5 \). In Step 2.1 \( (t_{5,1}, t_{5,2}, t_{5,3}) \leftarrow 
(0, 0, 0) \); in Step 2.2 \( z_1 \leftarrow \gcd (-3, 6) = -3 \); in Step 2.3 
\( z_2 \leftarrow \gcd (-3, 6) = 6 \); in Step 2.4 \( (y_1, y_2) \leftarrow \text{int2}(1, -1) = 
(0, 1) \); then in Step 2.5 \( t_{5,4} \) is assigned with 0, and 
\( t_{5,5} \) is assigned with 1. Hence, we have obtained the 
unimodular transformation matrix:

\[
\begin{pmatrix}
2 & 0 & 3 & -3 & 6 \\
1 & 0 & 1 & -1 & 2 \\
0 & 0 & 0 & 0 & 0
\end{pmatrix}
\]

Readers may want to check that this matrix indeed is
unimodular.

We should mention that since the return value of 
\( \text{int2}(x_1, x_2) \) might not be unique - different implementa-
tion can produce different values - the unimodular 
transformation matrix might not be unique as well. 
For example, in the last stage of the above instance, 
\( \text{int2}(1, -1) \) could return \((-1, -1)\) which apparently 
satisfies the definition. This results a different legal 
transformation matrix.

It is possible that \( c_r = 0 \). Consider the follow-
in situation: Suppose \( c_1 d_1 + c_2 d_2 + \cdots + c_q d_q = 0 \) 
is an \( \alpha \)-equation of an \( r \)-deep nested loop, where
\( c_r \neq 0 \) and \( g < r \). Let \( T' \) be the matrix produced 
by \( \text{getT}(c_1, c_2, \ldots, c_q) \). It can be proved that ma-
trix \( \begin{pmatrix} T' & 0 \\ 0 & U_{r-q} \end{pmatrix} \) is a legal unimodular matrix and 
it transforms \( L \) such that the outermost loop of the 
transformed code is parallelizable. Here \( U_{r-q} \) is a
\((r-q) \times (r-q)\) unit matrix. Suppose \( d_1 - d_2 = 0 \) is an 
\( \alpha \)-equation of a 4-deep nested loop, for instance. In 
the two matrices below, the left matrix is generated by 
\( \text{getT}(1, -1) \) and the right matrix is the transformation 
matrix.

\[
\begin{pmatrix}
1 & -1 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\]

5 Generating \( \alpha \)-inequalities from loop 
limits and index functions

A geometrical approach is adopted in this section. 
For a given nested loop, we view the iteration space 
as a polytope, and view the index functions as hyper-
planes. The \( \alpha \)-inequalities of the nested loop is 
obtained by analyzing the geometry relation between 
the polytope and hyperplanes. For the sake of simplic-
ity we discuss one dimensional array reference only; 
the approach is easily extended to multi-dimensional 
array. Let \( f(I) = \Sigma_{k=1}^{r} a_k I_k + a_0 \) and \( g(I) = 
\Sigma_{k=1}^{r} b_k I_k + b_0 \) be two index functions of an \( r \)-depth 
nested loop with iteration space \( R \). Since the loop 
limits are formed with linear functions of induction 
variables, the iteration space \( R \), which is defined 
by these functions, is a polytope from a geometry point 
of view. A hyperplane is defined from index functions:

\( f(I) - g(I) = 0 \), i.e., \( \Sigma_{k=1}^{r} (a_k - b_k) I_k + a_0 - b_0 = 0 \).

Hyperplane \( f(I) - g(I) = 0 \) has no intersection with it-
eration space \( R \), if any instance of iterations in \( R \), say
\( (i_1, \ldots, i_r) \), satisfies either \( \Sigma_{k=1}^{r} (a_k - b_k) I_k + a_0 - b_0 > 0 \),
or \( \Sigma_{k=1}^{r} (a_k - b_k) I_k + a_0 - b_0 < 0 \). The hyperplane is 
above iteration space \( R \), if the former is true, otherwise 
the hyperplane is below iteration space \( R \). The follow-
ing theorem tells the condition by which we can check 
if an \( \alpha \)-inequalities can be found from index functions 
and loop limits.

**THEOREM 5.1:** Let \( L \) be an \( r \)-deep nested loop. 
Suppose that the hyperplane \( f(I) - g(I) = 0 \) of it has no 
intersection with its iteration space \( R \). If the hyper-
plane is above \( R \), 
\[ \Sigma_{k=1}^{r} b_k d_k > 0 \quad \text{and} \quad \Sigma_{k=1}^{r} a_k d_k > 0, \]

are two \( \alpha \)-inequalities for the flow-dependence of the 
nested loop and

\[ \Sigma_{k=1}^{r} b_k d_k < 0 \quad \text{and} \quad \Sigma_{k=1}^{r} a_k d_k < 0, \]

are two \( \alpha \)-inequalities for the anti-dependence of the 
nested loop. If the hyperplane is below \( R \),

\[ \Sigma_{k=1}^{r} b_k d_k < 0 \quad \text{and} \quad \Sigma_{k=1}^{r} a_k d_k < 0, \]

are two \( \alpha \)-inequalities for the flow-dependence of the 
nested loop and

\[ \Sigma_{k=1}^{r} b_k d_k > 0 \quad \text{and} \quad \Sigma_{k=1}^{r} a_k d_k > 0, \]

are two \( \alpha \)-inequalities for the anti-dependence of the 
nested loop.

We now use Example 3 to show how \( \alpha \)-inequalities 
can be used to exploiting parallelism.
Example 3:

```fortran
  do l_1 = 1, n
    do l_2 = 1, n
      S[l_1 + 2l_2 + 1] = f(S[l_1 - l_2])
    enddo
  enddo
```

It is quite obvious that the innermost loop is parallelizable. However, it is not clear whether higher level parallelism is available. We apply Theorem 5.1. The hyperplane is \(I_1 + 2I_2 + 1 - I_1 + I_2 = 0\), that is, \(3I_2 + 1 = 0\). Since it is above the iteration space, according to Theorem 5.1, \(d_1 + 2d_2 > 0\) and \(d_1 - d_2 < 0\) are two \(\alpha\)-inequalities for flow dependence of the loop if there exists any \(\alpha\)-inequality for flow dependence. Notice that any distance vector of flow dependence \(d\) with \(d \cdot 2 > 0\) can not satisfy the second \(\alpha\) inequality, therefore there is no distance vector \(d\) of flow dependence satisfies \(d \cdot 2 > 0\) for this nested loop. Similarly, according to Theorem 5.1, \(d_1 + 2d_2 < 0\) and \(d_1 - d_2 < 0\) are two \(\alpha\)-inequalities for anti-dependence of the loop if there exists any \(\alpha\)-inequality for flow dependence. Notice that any distance vector of anti-dependence \(d\) with \(d \cdot 2 > 0\) can not satisfy \(d_1 + 2d_2 < 0\), therefore there is no distance vector \(d\) of anti-dependence satisfies \(d \cdot 2 > 0\) for this nested loop. We conclude there is no distance vector \(d\) with \(d \cdot 2 > 0\). Hence, \(l_2\) is also parallelizable.

6 Previous work

The foundation of unimodular loop transformation theory has been established by Banerjee[2], Wolf and Lam [12]. In their algorithms, however, transformation matrices are generated based on computing distance vectors or direction vectors. A different approach of designing loop transformation has been presented by Tzen and Ni [11], in which loop transformation are made based on the calculation of the slopes of the distance vectors without computing distance vectors themselves. Their approach can deal with double nested loop.

7 Summary

We presented in this paper a new technique for generating loop transformation matrices. It does not need to compute distance vectors or direction vectors, can handle nested loop with any depth, and is capable to exploit parallelism of nested loop in different level. Though only a simple model of nested loops is used in this paper, it should be no difficulties to apply our approach to more complicated nested loops which may contain multiple assignment statements. Our further research includes: 1) Finding an efficient algorithm to determine whether hyperplane \(f(I) - g(I) = 0\) is above or below or intersect with iteration space \(R\); 2) Finding a general way to utilize the information provided by \(\alpha\)-inequalities for the purpose of loop transformation and exploiting parallelism.

References