Partitioning Loops with Variable Dependence Distances

Yijun Yu and Erik H. D'Hollander
Department of Electronics and Information Systems
University of Ghent, Belgium
{yijun, dhollander}@elis.rug.ac.be

Abstract

A new technique to parallelize loops with variable distance vectors is presented. The method extends previous methods in two ways. First, the present method makes it possible for array subscripts to be any linear combination of all loop indices. The solutions to the linear dependence equations established from such array subscripts are characterized by a pseudo distance matrix(PDM). Second, it allows us to exploit loop parallelism from the PDM by applying unimodular and partitioning transformations that preserve the lexicographical order of the dependent iterations. The algorithms to derive the PDM, to find a suitable loop transformation and to generate parallel code are described, showing that it is possible to parallelize a wider range of loops automatically.

Keywords *loop parallelization, dependence equation, distance vector, pseudo distance matrix, unimodular transformation, iteration space partitioning*[1]

1 Introduction

The parallelism in loops is still the key objective of many program analyzers. The general way to reveal the loop parallelism is first solve the dependence order between the loop iterations that must run in sequence, then a particular loop can run in parallel when no dependencies exist between the iterations for the corresponding loop index.

In the past, ad hoc algorithms, pattern recognition, dependence test and loop transformation were some of the directions followed [4, 11, 16]. But there was an extra difficulty with the different techniques because a compiler had a hard time finding the right algorithm, or creating the right sequence of dependence tests.

In the last decade, a more systematic approach to address the uniform dependence distance problem has been developed by using unimodular matrix [1, 6], which presents an one-to-one mapping between two integer lattices. With this approach, any unimodular loop transformation which exposes or enhances the parallelism is applicable, under the condition that the lexicographical order between dependent iterations is preserved after transformation.

Our method here is a further development of the unimodular approaches by generalizing the uniform distance vectors to the pseudo distance matrix(PDM), which is constructed from the linear dependence equations. The purpose of using the PDM is to find suitable transformations that parallelize a loop with variable dependence distances.

The remainder of the paper is organized as follows. Section 2 introduces basic concepts for solving linear dependence equations and describes a method to obtain the PDM from the linear dependence equations. Section 3 discusses the algorithms to derive the loop parallelizing transformations from a non-full or a full-rank PDM. Section 4 presents examples illustrating the application of the method. Section 5 overviews the related work and compares this technique with other approaches. Finally, Section 6 concludes the paper.

2 Loop dependencies

The loop iteration space and the dependence equations are the framework to analyze and extract the parallelism between iterations. The variable distance between dependent iterations with linear subscript expressions is described uniformly by the concept of a *pseudo distance matrix*. The pseudo distance matrix allows us to extract the parallelism and also provides a way to enhance the parallelism by a suitable transformation of the iteration space.

2.1 Iteration space and dependence order

Consider an m-fold perfectly nested loop

$$\begin{array}{ll} L_1: & \text{do } I_1 = p_1, q_1 \\ & \quad \cdots \\ L_m: & \quad \text{do } I_m = p_m, q_m \\ & \quad\quad \mathcal{H}(I_1, \ldots, I_m) \\ & \quad \text{enddo} \\ & \cdots \\ & \text{enddo} \end{array} \tag{2.1}$$

[1]This work was supported by the Belgian government under contract GOA-12.0508.95.

where the loop limits p_1, q_1 are integer constants; p_r, q_r are integer functions of $I_1, \ldots, I_{r-1}$ for $1 < r \leq m$; the loop body $\mathcal{H}(I_1, \ldots, I_m)$ is a sequence of assign statements and there are no statements between loops. The loop nest is denoted by $\vec{L} = (L_1, \ldots, L_m)$. Its **iteration space** $\Phi \subset \mathcal{Z}^m$ contains all the integer vectors $\vec{i} = (i_1, \ldots, i_m)$ within the loop limits, i.e.:

$$\Phi = \{\vec{i} \mid p_r \leq i_r \leq q_r, r = 1, \ldots, m\}. \tag{2.2}$$

As a loop nest is sequentially executed, its iterations are traversed one by one in a total order. Index vectors $\vec{i} = (i_1, \ldots, i_m)$ and $\vec{j} = (j_1, \ldots, j_m)$ are **lexicographically ordered**, denoted by $\vec{i} \prec \vec{j}$, iff $i_1 < j_1$ or there exists an index $l : 1 \leq l \leq m$ such that $i_1 = j_1 \ldots i_{l-1} = j_{l-1}$, and $i_l < j_l$. When reordering the execution of the loop iterations, the lexicographical order among the dependent iterations must be preserved. If two iterations $\vec{i}, \vec{j}$ access the same memory location while at least one write, and $\vec{i} \prec \vec{j}$, then iteration $\vec{j}$ **directly depends on** iteration $\vec{i}$, denoted by $\vec{i}\ \delta\ \vec{j}$ and the **distance vector** $\vec{d} = \vec{j} - \vec{i}$. Since $\vec{i} \prec \vec{j}$, one always has $\vec{d} \succ \vec{0}$.

The dependence order of a loop is obtained by solving the dependence equations.

2.2 Dependence equations

Assume a perfectly nested loop $\vec{L}$ with loop indices, $\vec{I} = (I_1, \ldots, I_m)$ and let $X(\vec{f}(\vec{I}))$ and $X(\vec{g}(\vec{I}))$ denote two variables in the loop body, where X is an n dimensional array, and $\vec{f}, \vec{g} : \mathcal{Z}^m \to \mathcal{Z}^n$ denote index expressions. Both index expressions refer to the same array element if the following **dependence equations** hold:

$$\vec{f}(\vec{i}) = \vec{g}(\vec{j}). \tag{2.3}$$

In this paper, the array subscripts $\vec{f}(\vec{I})$ are linear functions of the loop indices $\vec{I}$, and can be written as $\vec{f}(\vec{I}) = \vec{I}\mathbf{A} + \vec{a}$ where $\mathbf{A}$ and $\vec{a}$ are a constant matrix and constant vector respectively. Therefore the **linear dependence equations** (2.3) become:

$$\vec{i}\mathbf{A} + \vec{a} = \vec{j}\mathbf{B} + \vec{b} \tag{2.4}$$

where $\mathbf{A}, \mathbf{B} \in \mathcal{Z}^{m \times n}$ are constant matrices and $\vec{a}, \vec{b} \in \mathcal{Z}^n$ are constant vectors. This is a system of *diophantine equations* because the solutions must be integral.

To solve the equations (2.4), they are rewritten as:

$$(\vec{i}; \vec{j}) \begin{pmatrix} \mathbf{A} \\ -\mathbf{B} \end{pmatrix} = \vec{b} - \vec{a}\,. \tag{2.5}$$

With $\mathbf{C} = \begin{pmatrix} \mathbf{A} \\ -\mathbf{B} \end{pmatrix}$ and $\vec{c} = \vec{b} - \vec{a}$, equations (2.5) reduce to:

$$(\vec{i}; \vec{j})\mathbf{C} = \vec{c} \tag{2.6}$$

The diophantine equations are solved using a reduction of the matrix $\mathbf{C}$ into the echelon form by a unimodular transformation.

A **unimodular** matrix $\mathbf{U}$ is an integer matrix with $\det(\mathbf{U}) = \pm 1$. Consequently, $\mathbf{U}^{-1}$ is also a unimodular matrix. A unimodular transformation of row index vector $\vec{I}$ is $\vec{I'} = \vec{I}\mathbf{U}$. Since $\vec{I} = \vec{I'}\mathbf{U}^{-1}$, there is a 1-1 mapping between $\vec{I}$ and $\vec{I'}$.

An echelon matrix is defined using the leading element and the level of a vector. The **leading element** d_l of vector $\vec{d}$ is the first nonzero element, while the **level** l of vector $\vec{d}$ is the index of the leading element.

Now an **echelon** matrix $\mathbf{S}$ with rank $r = rank(\mathbf{S})$ has the following properties:

1. only the first r rows are nonzero;
2. the successive row vectors have increasing levels $l_1 < l_2 < \cdots < l_r$ where the level l_i is the index of the first nonzero element in the i-th row.

To solve equations (2.6), the system of diophantine equations is replaced by an equivalent system:

$$(\vec{i}\,;\ \vec{j}) = \vec{t}\,\mathbf{U} \tag{2.7}$$

$$\vec{t}\,\mathbf{U}\mathbf{C} = \vec{c} \tag{2.8}$$

where $\mathbf{U}$ is a unimodular matrix, such that $\vec{t} = (\vec{i}; \vec{j})\mathbf{U}^{-1}$ exists, is unique and is integral. In other words, there is a *bijection* between $\vec{t}$ and $(\vec{i}; \vec{j})$.

Choosing $\mathbf{U}$ such that $\mathbf{UC} = \mathbf{S}$ is echelon by a common algorithm [2], the equations (2.8) become

$$\vec{t}\,\mathbf{S} = \vec{c} \tag{2.9}$$

which can be easily solved for $\vec{t}$ by backward substitution. Next substitute $\vec{t}$ in equation (2.7) to obtain $(\vec{i}; \vec{j})$, or

$$\begin{aligned} \vec{i} &= \vec{t}\,\mathbf{U}_l \\ \vec{j} &= \vec{t}\,\mathbf{U}_r \end{aligned} \tag{2.10}$$

where $\mathbf{U}_l, \mathbf{U}_r \in Z^{2m \times m}$ are the left and right submatrices of $\mathbf{U}$ respectively such that $\mathbf{U} = \begin{pmatrix} \mathbf{U}_l & \mathbf{U}_r \end{pmatrix}$.

Solution (2.10) has an associated distance vector $\vec{d}$, where $\vec{d} = \vec{j} - \vec{i}$ if $\vec{i} \preccurlyeq \vec{j}$, or $\vec{d} = \vec{i} - \vec{j}$ if $\vec{j} \prec \vec{i}$. For arbitrary dependent iterations $\vec{i}$ and $\vec{j}$, this is conveniently noted as

$$\vec{d} = |\vec{j} - \vec{i}|\,. \tag{2.11}$$

The **distance set** includes all the direct distance vectors obtained by solving the dependence equations (2.4):

$$\Delta = \{\vec{d} \mid \vec{d} = |\vec{j} - \vec{i}| = |\vec{t}\,(\mathbf{U}_r - \mathbf{U}_l)| = |\vec{t}\,\mathbf{F}|\ \} \tag{2.12}$$

where $\mathbf{F} = \mathbf{U}_r - \mathbf{U}_l$.

2.3 The pseudo distance matrix, PDM

The linear echelon reduced system (2.9) contains n equations in $2m$ unknowns, where n is the array dimension and m is the loop depth. If the rank of the echelon matrix $r = \text{rank}(\mathbf{S}) < 2m$, then the solution vector $\vec{t}$ contains r constant elements, and $2m - r$ elements are undetermined or so-called *free* variables. Without loss of generality, it can be assumed that the $\vec{t} = (\vec{t}_1; \vec{t}_2)$, where $\vec{t}_1 \in \mathcal{Z}^r$ is constant and $\vec{t}_2 \in \mathcal{Z}^{2m-r}$ is arbitrary.

Taking into account that the distance between two lexicographically executed dependent iterations is positive, the distance set is given by

$$\Delta = \{\vec{d} \mid \vec{d} = \vec{t}_1\ \mathbf{F}' + \vec{t}_2\ \mathbf{F}'', \vec{t}_2 \in \mathcal{Z}^{2m-r}, \vec{d} \succcurlyeq \vec{0}\} \quad (2.13)$$

where $\mathbf{F}', \mathbf{F}''$ are the upper and lower submatrices of $\mathbf{F}$ respectively such that $\mathbf{F} = \begin{pmatrix} \mathbf{F}' \\ \mathbf{F}'' \end{pmatrix}$.

A **lattice** is a group of vectors that contains all the linear combinations of the independent row vectors of a matrix $\mathbf{A} \in \mathcal{Z}^{n \times m}$:

$$\mathcal{L}(\mathbf{A}) \quad = \quad \{\vec{x}\mathbf{A} \mid \vec{x} \in \mathcal{Z}^n\}. \quad (2.14)$$

It can be used to characterize the distance set. In (2.13), Δ contains all **direct** distance vectors between two dependent iterations when an unbounded loop nest is considered. The **indirect** dependence between iterations has a distance vector which is a linear combination of the direct distance vectors in Δ. Therefore both the direct and indirect distance vectors are contained in the lattice $\mathcal{L}(\mathbf{R})$ where

$$\mathbf{R} = \begin{cases} \mathbf{F}'' & \vec{t}_1\mathbf{F}' \in \mathcal{L}(\mathbf{F}'') \text{ i.e.} \exists\ \vec{t}_0 \mid \vec{t}_1\mathbf{F}' = \vec{t}_0\mathbf{F}'' \\ \begin{pmatrix} \vec{t}_1\mathbf{F}' \\ \mathbf{F}'' \end{pmatrix} & \text{otherwise} \end{cases} \quad (2.15)$$

Note that the distance set in (2.12) becomes

$$\Delta \quad = \quad \{\vec{d} \mid \vec{d} = \vec{x}\mathbf{R}\ ,\ \vec{d} \succcurlyeq \vec{0}\} \quad (2.16)$$

with $\vec{x} \in \mathcal{Z}^q$:

$$\begin{cases} \vec{x} = \vec{t}_0 + \vec{t}_2, q{=}2m{-}r & \text{if } \vec{t}_1\mathbf{F}' = \vec{t}_0\mathbf{F}'' \\ \vec{x} = (1; \vec{t}_2), q{=}2m{-}r{+}1 & \text{otherwise} \end{cases} \quad (2.17)$$

In other words, the distance vectors are linear combinations of the row vectors in $\mathbf{R} \in \mathcal{Z}^{q \times m}$:

$$\Delta \subset \mathcal{L}(\mathbf{R}).$$

Then a Hermite normal form matrix can be used to reduce the lattice generating matrix $\mathbf{R}$.

A **Hermite normal form** matrix $\mathbf{H} \in \mathcal{Z}^{r \times m}$ is the full row rank matrix reduced from the echelon form: $h_{i,l_i} > h_{k,l_i} \geq 0$ for $1 \leq k < i$ where l_i is the level of the i-th row vector.

The unique Hermite normal form of arbitrary matrix $\mathbf{R}$ is obtained by a unimodular row transformation matrix $\mathbf{U}$: $\mathbf{H}$ is the full row rank submatrix of $\mathbf{UR}$, denoted as HNF($\mathbf{R}$).

$$\mathbf{H} \quad = \quad \text{HNF}(\mathbf{R}) \quad (2.18)$$

We will use $\mathbf{H}$ as the **pseudo distance matrix**, for the following reasons:

1. Since the unimodular transformation $\mathbf{U}$ is a 1-1 mapping between index vectors, the lattice $\mathcal{L}(\mathbf{H})$ is the same as the lattice $\mathcal{L}(\mathbf{R})$, i.e.,

$$\{\vec{x}\mathbf{R} \mid \vec{x} \in \mathcal{Z}^q\} = \{\vec{X}\mathbf{H} \mid \vec{X} \in \mathcal{Z}^p\} \quad (2.19)$$

   where $p = \text{rank}(\mathbf{R})$, $(\vec{X}; \vec{0}) = \vec{x}\mathbf{U}^{-1}$;
2. If there are zero columns in $\mathbf{H}$, according to Lemma 1, the corresponding loops are parallel;
3. If there are no zero columns in $\mathbf{H}$, two possibilities arise:
   (a) $\mathbf{H}$ is non-full rank: $\text{rank}(\mathbf{H}) < m$, the number of nested loops. In this case, $m - \text{rank}(\mathbf{H})$ loops can be parallelized after a suitable unimodular transformation;
   (b) $\mathbf{H}$ is full rank: $\text{rank}(\mathbf{H}) = m$. In that case, the row vectors in $\mathbf{H}$ can be considered as "constant" distance vectors and the parallelism equal to $\det(\mathbf{H})$ can be extracted using a loop transformation similar to the partitioning method described in [6];
4. Since $\mathbf{H}$ is an echelon matrix with lexicographically positive row vectors, according to Theorem 1 in the Section 3.1, it is easy to find *legal* unimodular loop transformations.

Lemma 1. *If the pseudo distance matrix* $\mathbf{H}$ *contains one or more zero columns, the corresponding loops can run in parallel.*

Proof. Suppose the i-th column contains only zeroes. Each distance $\vec{d}$ has the form $\vec{d} = \vec{X}\mathbf{H}$ for some suitable $\vec{X}$, consequently, $d_i = 0$. Loop L_i can run in parallel since every two dependent iterations have the same loop index. □

Now the distance set in (2.16) can be restated by $\mathbf{H} \in \mathcal{Z}^{p \times m}$:

$$\Delta \quad = \quad \{\vec{d} \mid \vec{d} = \vec{X}\mathbf{H}, \vec{X} \in \mathcal{Z}^p, \vec{d} \succcurlyeq \vec{0}\} \quad (2.20)$$

Multiple pairs of references The PDM for a single pair of array references is extended to represent all distance vectors in the loop. Each pair of dependent array references has a distance set Δ_l, the distance set of the loop is the union of all pair-wise distance sets: $\Delta_L = \bigcup_l \Delta_l$. For each pair of references l, a lattice generating matrix $\mathbf{R}_l$ is found by equation (2.15). The lattice generating matrix of the loop, $\mathbf{R}_L$, contains distinct rows of the matrices $\mathbf{R}_l$. Equally, the Hermite normal form matrix is then

$$\mathbf{H}_L \quad = \quad \text{HNF}(\mathbf{R}_L) \; . \tag{2.21}$$

In the remainder of the paper, if not explicitly explained, $\mathbf{H}$ denotes $\mathbf{H}_L$, the PDM of the loop.

3 Loop transformations using PDM

Parallelism in nested loops can be enhanced by loop transformations. In this section the legality of loop transformations for variable distance vectors is first analyzed. Then two methods are given to extract parallel iterations while preserving the lexicographical dependencies by using legal transformations only.

3.1 Legal operations

Transforming the iteration space in a loop with variable distances, requires a framework to delineate which operations are permitted.

A **legal** loop transformation has the following properties:

1. it reorders the iteration space as 1-to-1 mapping, i.e. for any iteration $\vec{i}$ in iteration space Φ there is one and only one iteration $\vec{i}'$ in the new iteration space Φ';
2. it preserves the lexicographical order of the dependent iterations, i.e. for any two dependent iterations $\vec{i}$ and $\vec{j}$ in Φ, the new iterations $\vec{i}'$ and $\vec{j}'$ are also dependent in the same order: $\vec{i} \; \delta \; \vec{j} \Rightarrow \vec{i}' \; \delta \; \vec{j}'$.

Unimodular transformations satisfy the first property. The second property must be ensured by the proper transformations. A unimodular transformation matrix $\mathbf{T}$ transforms the iteration space Φ into iteration space Φ', such that the index vector $\vec{I} \in \Phi$ corresponds to the index vector $\vec{I}' = \vec{I}\mathbf{T} \in \Phi'$, as shown in Figure 1.

$\vec{L}$:
do $\vec{I} \in \Phi$
$\mathcal{H}(\vec{I})$ $\Rightarrow$
enddo

$\vec{L}'$:
do $\vec{I}' \in \Phi'$
$\vec{I} = \vec{I}'\mathbf{T}^{-1}$
$\mathcal{H}(\vec{I})$
enddo

Figure 1. Unimodular loop transformation

By unimodular transformation, the requirement of legal loop transformation is stated as follows.

Definition 1. Legal unimodular transformation*: A* **legal** *unimodular transformation matrix* $\mathbf{T}$ *preserves the lexicographical order of the dependent iterations:* $\vec{i} \; \delta \; \vec{j} \Rightarrow \vec{i}\mathbf{T} \; \delta \; \vec{j}\mathbf{T}$ *i.e.* $\vec{d} \succ \vec{0} \Rightarrow \vec{d}\mathbf{T} = (\vec{j} - \vec{i})\mathbf{T} \succ \vec{0}$. □

Corollary 1. *If a unimodular matrix* $\mathbf{T}_1$ *legally transforms loop* $\vec{L}$ *to* $\vec{L}'$*, and another unimodular matrix* $\mathbf{T}_2$ *legally transforms* $\vec{L}'$ *to* $\vec{L}''$*, then* $\mathbf{T} = \mathbf{T}_1\mathbf{T}_2$ *is a unimodular matrix legally transforms* $\vec{L}$ *to* $\vec{L}''$.

Proof. As the multiple of two unimodular matrices, $\mathbf{T}$ is still unimodular. Since $\mathbf{T}_1$ is legal for $\vec{L}$, by Definition 1, for any distance vector $\vec{d} \succ \vec{0}$ in $\vec{L}$, the corresponding distance vector in $\vec{L}'$ is $\vec{d}\mathbf{T}_1 \succ \vec{0}$. Because $\mathbf{T}_2$ is legal for $\vec{L}'$, $\vec{d}\mathbf{T}_1\mathbf{T}_2 \succ \vec{0}$. Therefore $\vec{d}\mathbf{T} \succ \vec{0}$ for $\mathbf{T} = \mathbf{T}_1\mathbf{T}_2$. By Definition 1, T is also legal for $\vec{L}$. □

From Lemma 1, the loops corresponding to the zero columns in the pseudo distance matrix can run in parallel. Therefore we are going to use an algorithm to find legal unimodular transformations that eliminate the columns to zero.

From Corollary 1, we can use legal unimodular transformations successively to obtain the legal unimodular transformation. The unimodular transformations to be used in the algorithm are:

1. **right skewing**, denoted by skewing(i, j, f): add fI_i to I_j where $f \in Z$ and $j > i$;
2. **interchange**, denoted by interchange(i, j): exchange I_i and I_j;
3. **shift**, denoted by shift(i, j): shift I_i to I_j.

Now we are going to explain the conditions for these transformations to be legal according to the pseudo distance matrix.

First, the relationship between the lexicographical positive echelon matrix and the distance vectors is stated by the following lemma [3].

Lemma 2. *Given an echelon matrix* $\mathbf{S}$ *with lexicographically positive row vectors. Then* $\vec{X}\mathbf{S} \succ \vec{0}$ *iff* $\vec{X} \succ \vec{0}$*, and* $\vec{X}\mathbf{S} = \vec{0}$ *iff* $\vec{X} = \vec{0}$.

The next theorem shows the possibility to check the legality of the unimodular transformation according to the pseudo distance matrix.

Theorem 1. *Given a pseudo distance matrix* $\mathbf{H}$*, if a unimodular transformation matrix* $\mathbf{T}$ *is such that* $\mathbf{H}' = \mathbf{H}\mathbf{T}$ *is an echelon matrix with lexicographically positive row vectors, then* $\mathbf{T}$ *is legal.*

Proof. For any distance vector $\vec{d} \succ \vec{0}$ in the loop there exists an integer vector $\vec{X}$ such that $\vec{d} = \vec{X}\mathbf{H}$. Since $\mathbf{H}$ is a HNF, it is an echelon matrix with lexicographically positive row vectors. Therefore, according to Lemma 2, $\vec{X} \succ \vec{0}$. Since $\vec{X} \succ \vec{0}$ and by assumption $\mathbf{H}' = \mathbf{H}\mathbf{T}$ is an echelon

matrix with lexicographically positive row vectors, then the transformed distance vector $\vec{d}' = \vec{X}\mathbf{H}' \succ \vec{0}$, again using Lemma 2. □

Corollary 2. *The right-skewing transformation skewing(i, j, f) is always legal.*

Proof. Denote l_k as the level of the k-th row in the PDM $\mathbf{H}$. If $l_k < j$ then right skewing will not change the leading element; if $l_k = j$ then $h_{k,i} = 0$ for $i < j$ and the leading element will not change either. Since none of the leading elements is changed, $\mathbf{HT}$ is still an echelon matrix with lexicographically positive vectors. Thus right skewing is legal by Theorem 1. □

Corollary 3. *Given a PDM that the i-th column is zero, transformation shift(i, j) is legal.*

Proof. The shift of a zero column may change the index of the leading element, but not its sign. Therefore the resulting matrix still has lexicographically positive row vectors. It is still echelon because the levels of the row vectors still keep the order $l_1 < \cdots < l_r$. Therefore the shifting is legal by Theorem 1. □

Together with Lemma 1, the parallel loops found by zero columns can be shifted to the outermost of the loop nest to obtain coarse-grain parallelism, or to the innermost to obtain fine-grain parallelism.

Corollary 4. *If in the PDM $\mathbf{H}$, the j-th column is linearly dependent on the first i columns and $h_{i,j} > 0$, $interchange(i, j)$ is legal.*

Proof. If $\mathbf{H}' = \mathbf{HT}$ is echelon and has lexicographically positive row vectors, then the interchange is legal according to Theorem 1. By assumption, the j-th column of $\mathbf{H}$ is linearly dependent on the first i columns, i.e., there exists a vector $\vec{y}$ such that $h_{i,j} = y_1 h_{i,1} + \cdots + y_i h_{i,i}$. According to the property of HNF, the components of $\mathbf{H}$ below the diagonal are zero, and by assumption $h_{i,j} > 0$, therefore $h_{i,j} = y_i h_{i,i} > 0$. Consequently $y_i > 0$. For each k-th row vector in the HNF $\mathbf{H}$, after the interchange of i-th and j-th columns:

- If $l_k < i$, then the leading element is not changed;
- If $l_k = i$, then $h_{k,j} = y_i h_{k,i} > 0$, the new leading element is still positive;
- If $l_k > i$, then $h_{k,j} = y_i h_{k,i} = 0$, the leading element is not changed.

Therefore the row vectors in the echelon matrix $\mathbf{HT}$ are still lexicographically positive. Since $h_{i,j} > 0$, and only the k-th row vector that $l_k = i$ has the leading element changed, but not its sign and level, therefore, the resulting matrix is still echelon. □

3.2 Unimodular transformation

The next algorithm starts with a Hermite normal form matrix, $\mathbf{H} \in \mathcal{Z}^{r \times m}$. Given $r = \text{rank}(H)$, the algorithm will find a legal unimodular transformation matrix $\mathbf{T}$, which reduces $\mathbf{H}$ into $\mathbf{HT}$, where the first $m - r$ columns are zero.

The algorithm loops over each column j. At the beginning of each j-iteration, there are z zero columns and p nonzero columns before the j-th column, such that $j = z + p + 1$. Therefore the rank of the first $j - 1$ columns is p. In average, the algorithm takes $O(p \cdot m \cdot \ln(M))$ column operations where each column has m elements, M is the largest element in the PDM.

Algorithm 1. Transforming non-full rank PDM

/ Initialization: */* $\mathbf{T} = \mathbf{I}, p = z = 0$
do $j = 1, m$
/ Is column j independent of the previous columns? */*
  if $p < r$ *and* $h_{p+1,j} > 0$ *then*
    / Yes, increase the number of nonzero columns p */*
    $p \leftarrow p + 1$
  else / No, step-wise eliminate element $h_{i,j}$ with*
      ** the nonzero element $h_{i,z+i}$ */*
    do $i = p, 1, -1$
      do while $h_{i,j} \neq 0$
        $f = (h_{i,j} - h_{i,j} \mod h_{i,z+i})/h_{i,z+i}$
        / record the unimodular transformation */*
        col(j) = *col*(j) $-$ f $*$ *col*(i) *in* $\mathbf{H}, \mathbf{T}$
        if $h_{i,j} > 0$ *then*
          / put the smaller element in j-th column */*
          exchange col$(z + i)$ *and col*(j) *in* $\mathbf{H}, \mathbf{T}$
        endif
      enddowhile
    enddo
    $z \leftarrow z + 1$ */* count the extra zero column */*
    shift j-th (zero) column to the first in $\mathbf{H}, \mathbf{T}$
  endif
enddo

3.3 Iteration space partitioning

This subsection presents a method to exploit $\det(\mathbf{H})$ parallelism for the full rank PDM $\mathbf{H}$ by the partitioning loop transformation [6].

Each distance vector $\vec{d}$ is within the lattice $\mathcal{L}(\mathbf{H})$:

$$\vec{d} \quad = \quad \vec{X}\mathbf{H} \tag{3.1}$$

where $\mathbf{H}$ is an upper triangular matrix. According to Lemma 2, $\vec{X}$ must be lexicographically positive such that $\vec{X}\mathbf{H} \succ \vec{0}$. Therefore, one must be prudent that the following partitioning is still a legal transformation, i.e., the dependent iterations must be executed in the same order as in the original loop.

Theorem 2. *Given Loop (2.1) with full rank PDM* $\mathbf{H} \in \mathcal{Z}^{m\times m}$*, the following transformation is legal.*

$$
\begin{array}{l}
doall\ Io_1 = 0, h_{11}-1 \\
\cdots \\
\ doall\ Io_m = 0, h_{mm}-1 \\
\ \ io'_1 = Io_1 \\
\ \ do\ I_1 = p_1 + mod(io'_1 - p_1, h_{11}), q_1 - mod(q_1 - io'_1, h_{11}), h_{11} \\
\ \ \cdots \\
\ \ \ Io'_j = Io_j + \sum_{k=1}^{j-1}(I_k - Io'_k)h_{kj}/h_{kk} \\
\ \ \cdots \\
\ \ \ do\ I_m = p_m + mod(io'_m - p_m, h_{mm}), \\
* \qquad q_m - mod(q_m - io'_m, h_{mm}), h_{mm} \\
\qquad \mathcal{H}(I_1, \ldots, I_m) \\
\ \ enddo \\
\ \ \cdots \\
\ enddo \\
enddo \\
\cdots \\
enddo
\end{array}
\tag{3.2}
$$

Proof. The following mapping between the original index $\vec{I}$ and $\vec{J} = (\vec{I_o}; \vec{X})$ is one-to-one according the lemma 4.7 in [3]:

$$
\begin{cases}
\vec{I} = \vec{I_o} + \vec{X}\mathbf{H} \\
0 \le Io_j < h_{j,j} \text{ for } 1 \le i \le m
\end{cases}
\tag{3.3}
$$

Since $\mathbf{H}$ as full rank HNF is triangular, thus given $\vec{I}$ and $\vec{I_o}$, $\vec{X}$ can be solved by forward substitution on $j = 1, \ldots, m$ in solution (3.4):

$$
\begin{array}{rcl}
X_j & = & (I_j - Io'_j)/h_{j,j} \\
Io'_j & = & Io_j + \sum_{k=1}^{j-1} X_k h_{k,j} \\
 & = & Io_j + \sum_{k=1}^{j-1}(I_k - Io'_k)h_{k,j}/h_{k,k}
\end{array}
\tag{3.4}
$$

The new loop is constructed as Loop (3.2) by considering the loop limits in Loop (2.1), as in [6]:
Any two dependent iterations $\vec{i}_1$ and $\vec{i}_2$ in the original iteration space are mapped one-to-one to $\vec{i}'_1$ and $\vec{i}'_2$ respectively:

$$
\begin{array}{rcl}
\vec{i}_1 = i\vec{o}_1 + \vec{x}_1\mathbf{H} & \leftrightarrow & \vec{i}'_1 = (i\vec{o}_1, \vec{x}_1) \\
\vec{i}_2 = i\vec{o}_2 + \vec{x}_2\mathbf{H} & \leftrightarrow & \vec{i}'_2 = (i\vec{o}_2, \vec{x}_2)
\end{array}
\tag{3.5}
$$

Their distance vector is $\vec{d} = \vec{i}_2 - \vec{i}_1 = \vec{X}\mathbf{H}$. Since $\vec{i}_2 = \vec{i}_1 + \vec{d} = i\vec{o}_1 + \vec{x}_1\mathbf{H} + \vec{X}\mathbf{H}$, therefore $\vec{i}_2 = i\vec{o}_1 + (\vec{x}_1 + \vec{X})\mathbf{H}$. It can also be mapped to

$$
\vec{i}'_2 \quad = \quad (i\vec{o}_1, \vec{x}_1 + \vec{X}) \tag{3.6}
$$

By the uniqueness of the one-to-one mapping,

$$
\begin{array}{rcl}
i\vec{o}_2 & = & i\vec{o}_1 \\
\vec{x}_2 & = & \vec{x}_1 + \vec{X}.
\end{array}
\tag{3.7}
$$

The new distance vector is $\vec{d'} = \vec{i}'_2 - \vec{i}'_1 = (i\vec{o}_2 - i\vec{o}_1, \vec{x}_2 - \vec{x}_1) = (\vec{0}, \vec{X})$. Because $\vec{d} = \vec{X}\mathbf{H} \succ \vec{0}$, by Lemma 2, $\vec{X} \succ \vec{0}$, hence $\vec{d'} \succ \vec{0}$. □

4 Examples

This section presents two complete examples to show the application of the method to the loops with variable distances.

4.1 Non-full rank pseudo distance matrix

Consider the following loop nest:

```
do I1 = -N, N
   do I2 = -N, N
      a(3I1 + 1, 2I1 + I2 - 1) = ···
      ··· a(I1 + 3, I2 + 1) ···
   enddo
enddo
```

Figure 2 shows its iteration space dependence graph(ISDG) when $N = 10$. The loop has non-uniform distance analyzed as follows. From references $X(3I_1 + 1, 2I_1 + I_2 - 1)$ and $X(I_1 + 3, I_2 + 1)$, $\vec{i}$ and $\vec{j}$ are dependent if the following equations (2.6) have integer solutions:

$$
\begin{pmatrix} i_1 & i_2 & j_1 & j_2 \end{pmatrix}
\begin{pmatrix} 3 & 2 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \end{pmatrix}
= \begin{pmatrix} 2 & 2 \end{pmatrix}. \tag{4.1}
$$

Deriving an echelon matrix from the coefficient matrix by a unimodular matrix $\mathbf{U}$:

$$
\begin{pmatrix} 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & -2 & 3 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}
\begin{pmatrix} 3 & 2 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \end{pmatrix}
= \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \tag{4.2}
$$

the equations (4.1) can be simplified to:

$$
\begin{pmatrix} t_1 & t_2 & t_3 & t_4 \end{pmatrix}
\begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}
= \begin{pmatrix} 2 & 2 \end{pmatrix}. \tag{4.3}
$$

whose solutions are $\vec{t} = (\vec{t_1}; \vec{t_2}) = (2, 2; t_3, t_4)$, or, $\vec{i} = \vec{t}\,\mathbf{U}_l = (t_3, 2 - 2t_3 + t_4), \vec{j} = \vec{t}\,\mathbf{U}_r = (-2 + 3t_3, t_4)$. Consequently,

$$
\mathbf{F} = \begin{pmatrix} -1 & 0 \\ 0 & -1 \\ 2 & 2 \\ 0 & 0 \end{pmatrix}, \mathbf{R} = \begin{pmatrix} \vec{t_1}\ \mathbf{F}' \\ \mathbf{F}'' \end{pmatrix} = \begin{pmatrix} -2 & -2 \\ 2 & 2 \\ 0 & 0 \end{pmatrix}. \tag{4.4}
$$

From the reference $X(3I_1 + 1, 2I_1 + I_2 - 1)$ to itself, the following output dependence equations are established:

$$
(\vec{i}; \vec{j}) \begin{pmatrix} 3 & 2 \\ 0 & 1 \\ -3 & -2 \\ 0 & -1 \end{pmatrix}
= (1, -1) - (1, -1) \tag{4.5}
$$

From equation (4.5), similarly, the $\mathbf{R}$ is derived as:

$$\mathbf{R} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}. \quad (4.6)$$

Merging the two $\mathbf{R}$ in (4.6) and (4.4) yields

$$\mathbf{H} = \text{HNF}(\mathbf{R}_L) = \begin{pmatrix} 2 & 2 \end{pmatrix}. \quad (4.7)$$

Since $\mathbf{H}$ is not full rank, algorithm 1 is then applied to eliminate its leftmost column to zero by the legal unimodular transformation:

$$\mathbf{T} = \begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} -1 & 1 \\ 1 & 0 \end{pmatrix} \quad (4.8)$$

The loop limits of the transformed loop are found by using Fourier-Motzkin elimination [1, 13]:

doall $J_1 = -2N, 2N$
 do $J_2 = \max(-N, -N - J_1), \min(N, N - J_1)$
 $I_1 = J_2$
 $I_2 = J_1 + J_2$
 $a(3I_1 + 1, 2I_1 + I_2 - 1) = \cdots$
 $\cdots a(I_1 + 3, I_2 + 1) \cdots$
 enddo
enddo

Now the new PDM $\mathbf{H}' = \mathbf{HT} = \begin{pmatrix} 0 & 2 \end{pmatrix}$ has a full rank sub-matrix with determinant greater than 1. Further applying the method described in Section 3.3, more parallelism is exploited by a partitioning transformation:

doall $Jo_2 = 0, 1$
 doall $J_1 = -2N, 2N$
 $p_2 = \max(-N, -N - J_1)$
 $q_2 = \min(N, N - J_1)$
 do $J_2 = p_2 + \text{mod}(Jo_2\text{–}p_2, 2), q_2\text{–mod}(q_2\text{–}Jo_2, 2), 2$
 $I_1 = J_2$
 $I_2 = J_1 + J_2$
 $a(3I_1 + 1, 2I_1 + I_2 - 1) = \cdots$
 $\cdots a(I_1 + 3, I_2 + 1) \cdots$
 enddo
 enddo
enddo

Figure 3 shows the ISDG of the transformed loop.

4.2 Full rank pseudo distance matrix

Consider the following loop.

do $I_1 = -N, N$
 do $I_2 = -N, N$
 $a(4I_1 - I_2 + 3, 2I_1 + I_2 - 2) = \cdots$
 $\cdots a(I_1 + I_2 - 1, I_1 - I_2 + 2) \cdots$
 enddo
enddo

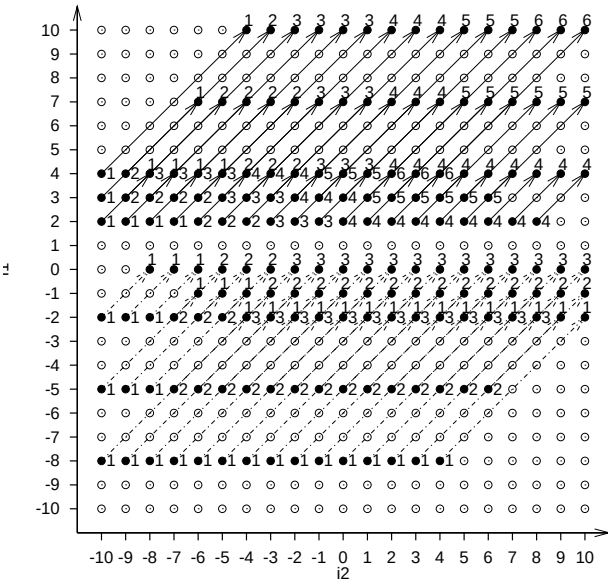


Figure 2. The ISDG of the original loop in Section 4.1 (N=10). Solid nodes are dependent while empty nodes are independent iterations. Each arrow shows the dependence order of two dependent iterations: solid one means a dependence from reference (1) to (2) while dashes one means a dependence from reference (2) to (1). To avoid the ambiguity, $(\vec{i}_1, \vec{j}_1)$ and $(\vec{i}_2, \vec{j}_2)$ on the same line are numbered in the order of $\vec{i}_1 \prec \vec{i}_2$.

From the ISDG in Figure 4 of this loop, one can see that it has non-uniform distances either.
Let us solve the pseudo distance matrix first.
For the references $a(4I_1 - I_2 + 3, 2I_1 + I_2 - 2)$ and $a(I_1 + I_2 - 1, I_1 - I_2 + 2)$, equations (2.5) are established:

$$(i_1, i_2, j_1, j_2) \begin{pmatrix} 4 & 2 \\ -1 & 1 \\ -1 & -1 \\ -1 & 1 \end{pmatrix} = \begin{pmatrix} -4 & 4 \end{pmatrix} \quad (4.9)$$

To simplify the diophantine equations (4.9), reduce the matrix to echelon form by a unimodular matrix:

$$\begin{pmatrix} 0 & 0 & -1 & 0 \\ 0 & 1 & -1 & 0 \\ 1 & 1 & 3 & 0 \\ 0 & -1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 4 & 2 \\ -1 & 1 \\ -1 & -1 \\ -1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 0 & 2 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \quad (4.10)$$

The solution in (2.15) is derived as $\vec{t} = (-4, 4; t_3, t_4)$, and $\vec{d} = |\vec{t}\,\mathbf{F}| = |(1; t_3, t_4)\mathbf{R}|$ where

$$\mathbf{F} = \begin{pmatrix} -1 & 0 \\ -1 & -1 \\ 2 & -1 \\ 0 & 2 \end{pmatrix}, \mathbf{R} = \begin{pmatrix} 0 & -4 \\ 2 & -1 \\ 0 & 2 \end{pmatrix}. \quad (4.11)$$

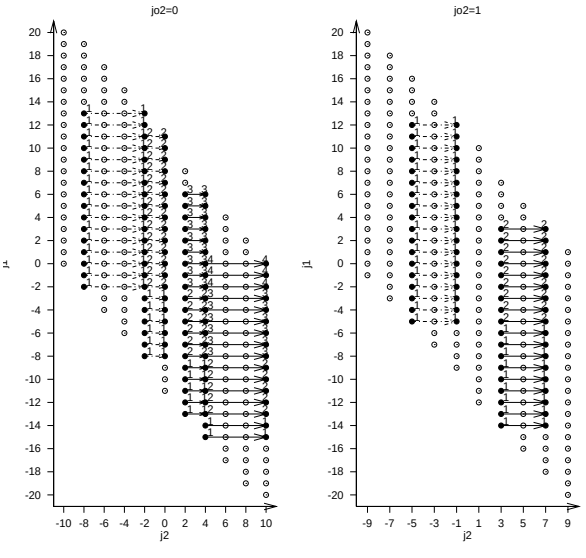


Figure 3. The ISDG of the transformed loop in Section 4.1 after unimodular and partitioning transformations. The original iteration space in Figure 2 has become two separate partitions. The shortened(in proportion to the increased step size of J_2) dependence arrows are vertical to the loop index J_1 axis, which means that the iterations along the J_1 direction are independent.

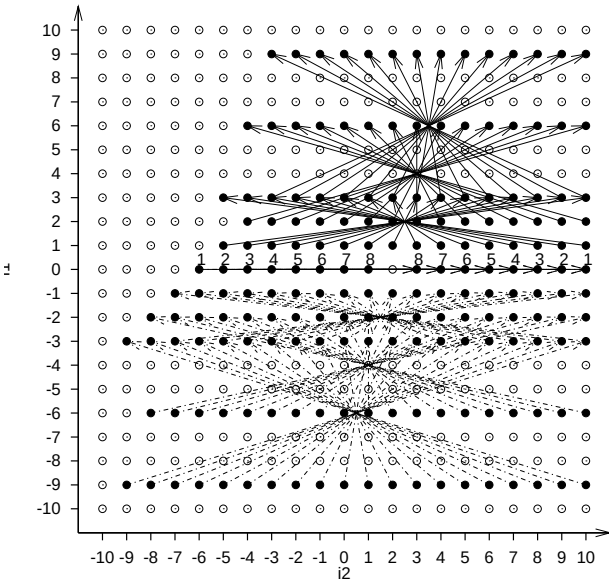


Figure 4. The iteration space of the original loop when N=10. An arrow between two dependent iterations always jumps a stride greater than 1 along direction of I_1 and/or I_2, which implies the existence of independent partitions.

Similarly, the reference $a(4I_1 - I_2 + 3, 2I_1 + I_2 - 2)$ to itself yields a zero generating matrix $\mathbf{R}$. Append it to the bottom of the $\mathbf{R}$ in (4.11), the $\mathbf{R_L}$ is obtained, whose HNF is the pseudo distance matrix:

$$\mathbf{H} = \begin{pmatrix} 2 & 1 \\ 0 & 2 \end{pmatrix} . \tag{4.12}$$

Applying partitioning method on the full rank PDM, the loop nest is transformed to:

doall $Io_1 = 0, 1$
  doall $Io_2 = 0, 1$
    do $I_1 = -N + \text{mod}(N + Io_1, 2), N - \text{mod}(N - Io_1, 2), 2$
      $io_2' = Io_2 + (I_1 - Io_1)/2$
      do $I_2 = -N + \text{mod}(N + io_2', 2), N - \text{mod}(N - io_2', 2), 2$
        $a(4I_1 - I_2 + 3, 2I_1 + I_2 - 2) = \cdots$
        $\cdots a(I_1 + I_2 - 1, I_1 - I_2 + 2) \cdots$
      enddo
    enddo
  enddo
enddo

It has det($\mathbf{H}$) parallel iterations in the Io_1 and Io_2 loops. The corresponding ISDG is shown in Figure 5.

5 Related work

The related work are compared in these directions: the accuracy of dependence information, the applicability to the type of programs, the parallelism extracted from the loops and the code generation difficulty. See Table 1.

5.1 Constant distance and linear loop transformations

To enhance loop parallelism, Banerjee [1, 2, 3] introduced the unimodular transformation framework, which has integrated the three elementary loop transformations: loop reversal, interchanging, skewing. D'Hollander [6] used partitioning transformation which can extract more parallelism for the full-rank distance matrix. Both treatments of unimodular loop transformation consider a *uniform* distance matrix.

A vector $\vec{d}$ is called *uniform* or *constant* distance, if it is a constant vector such that $\forall \vec{i} \in \Phi : (\vec{i}+\vec{d}) \in \Phi \Rightarrow \vec{i}\ \delta\ (\vec{i}+\vec{d})$; otherwise is called *non-uniform* or *variable* distance. The following corollary shows that only special linear dependence equations will yield a uniform distance vector.

Corollary 5. *The distance vector between two dependent iterations $\vec{i}$ and $\vec{j}$: $\vec{i}\mathbf{A} + \vec{a} = \vec{j}\mathbf{B} + \vec{b}$ is a constant distance vector iff $\mathbf{A} = \mathbf{B}$ are nonsingular and $\vec{d} = (\vec{b} - \vec{a})\mathbf{A}^{-1}$ is integral.*

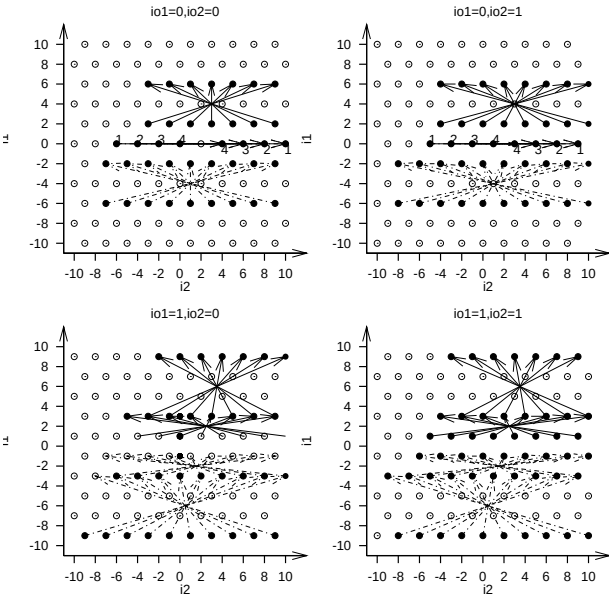
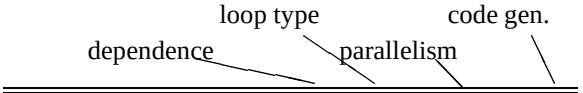


Figure 5. The new ISDG after being partitioned into four 2-D iteration spaces. The dependence arrows have shorter length in proportion to the increased step size than that in the original ISDG Figure 4. The skewing affects the offsets of the iteration indices, while the iteration space has the same square shape as the original.



Banerjee [1]	U	PL	O/NA	U
D'Hollander [6]	U	PL	O/NA	P
Ramanujam [12], Xue [18]	U	PL	O/NA	L
Xue [19]	U	NL	NA/NA	U
Wolf et al [14]	D	PL	O/O^-	U
Shang et al [17]	B	PL	O/O^-	S
Lim et al [9], Kelly et al [8]	U	SP	O^-/O^-	M
This work	P	PL	O/O^-	U

Table 1. Related work: **Column 2** compares the accuracy of dependence information: U - uniform distance vectors, D - dependence vectors, B - Base dependence vectors, P - pseudo distance matrix. **Column 3** shows whether optimal parallelism is exploited for uniform distance(u) or variable distance problem(v) by the methods: NA - no explicit parallelism exploiting mechanism proposed. O^- - suboptimal: optimal in degree of parallelism(proportion to loop size), O - optimal. **Column 4** shows the loop types that are applicable: PL - Perfectly nested loops, NL - Non-perfectly nested loops, SP - Statement-level parallelism. **Column 5** shows the methods in generating code: P - loop partitioning, U - unimodular loop transformation, L - non-singular linear loop transformation, M - statement-level mapping, S - linear scheduling.

Therefore a uniform distance vector is a special case of the pseudo distance matrix. Consequently, the statements regarding the PDM apply to the uniform distance case as well.

The advantage of using pseudo distance matrix instead of the distance set is mainly due to the fact that affine vectors are covered by the lattice generated by the PDM and therefore the legal transformation can be constructed without calculating each distance vectors and loop limits.

Xue [19] presented the idea of applying unimodular transformation on *non-perfectly* nested loops to generate loop without using guarding IF statement to the loop boundary. Essentially, an additional dimension can be added to the iteration space for the body statements, allowing us to reorder the statements in non-perfectly nested loops too.

Ramanujam [12] as well as Xue [18] proposed *non-singular linear* transformation where the integer transforming matrix $\mathbf{T}$ has non-zero determinant, to enhance parallelism. Since the distance set of the transformed loop can be expressed as a lattice generated by the HNF$(\mathbf{R}_L\mathbf{T})$, their methods apply to the PDM. In our work, except for the elementary unimodular transformations, only the simple partitioning transformations are non-unimodular. This eases the code generation, while the HNF guarantees as much parallelism as in the non-singular linear transformations.

5.2 Other variable distances methods

To test variable distance dependence, the *range test* [4] is based on the value range of non-linear expressions and the *omega* test [10] is based on exact integer programming. When dependence exists, loops are transformed to enhance the parallelism [8].

Wolf et al [14, 15] extended the uniform distance vectors to *dependence vectors*, which include *direction vectors* as a special case. Both distance and direction vectors are treated in the same framework of dependence vectors. However, the dependence vectors are less accurate than the pseudo distance vectors for the linear dependence distance.

Shang et al [17] represented the variable distance vector as an affine(*nonnegative* linear) combination of the basic dependence vectors(BDV). The Basic Ideas I and III always generate a set of full-rank BDV which inhibit parallelizing the outermost loops by a unimodular transformation, while the Basic Idea II of searching for a set of *cone-optimal* BDV, i.e., the BDV are minimal in rank, is closer to us. But the lexicographical positiveness is not carried by the BDV, so an additional *linear scheduling* [7] is needed to maintain the lexicographical order.

5.3 Processor mapping

Kelly and Pugh [8] unified the statement-level loop transformations using affine *processor mapping*. The idea can be briefly described as : (1) Each statement $s_k(\vec{i})$ is executed by processor $\phi_k(\vec{i})$; (2) For each dependence equation, it is required that $\vec{f}(\vec{i}) = \vec{g}(\vec{j}) \Rightarrow \phi_1(\vec{i}) = \phi_2(\vec{j})$. Given a linear mapping function, Omega calculator can be used to generate code. Their statement-level mapping can be specialized to an iteration-level mapping requiring all statements in the loop body bound to the same processor.

Lim and Lam [9] solved the affine mapping functions such that the parallelism is maximized. To obtain the mapping function in Lim's approach, however, it is required to solve dependence equations together with the loop boundary constraint. In our method, calculating the PDM and the transformations does not require loop limits calculation. The boundary information is only used for the code generation.

6 Conclusion

As long as the array subscripts are linear functions of the loop indices, a pseudo distance matrix, PDM, can be obtained such that any dependence distance vector in the loop is a linear combination of the rows of the pseudo distance matrix. A method has been presented to find a parallelizing loop transformation using the properties of the PDM. This work extends previous results for constant distance vectors, so that a more general class of loops can be addressed. The transformation requires no loop bounds calculations and is therefore quite efficient. The method has been implemented in the FPT [5, 20] compiler.

References

[1] U. Banerjee. Unimodular transformations of double loops. In *Advances in Languages and Compilers for Parallel Computing, 1990 Workshop*, Research Monographs in Parallel and Distributed Computing, pages 192–219, Irvine, Calif., Aug. 1990. Cambridge, Mass.: MIT Press.

[2] U. Banerjee. *Loop Transformations for Restructuring Compilers: The Foundations*. Norwell, Mass.: Kluwer Academic Publishers, 1993.

[3] U. Banerjee. *Loop Parallelization*. Norwell, Mass.: Kluwer Academic Publishers, 1994.

[4] W. Blume and R. Eigenmann. The range test: a dependence test for symbolic, non-linear expressions. In IEEE, editor, *Proceedings, Supercomputing '94: Washington, DC, November 14–18, 1994*, Supercomputing, pages 528–537. IEEE Computer Society Press, 1994.

[5] E. D'Hollander, F. Zhang, and Q. Wang. The fortran parallel transformer and its programming environment. *Journal of Information Sciences*, 106:293–317, 1998.

[6] E. H. D'Hollander. Partitioning and labeling of loops by unimodular transformations. *IEEE Transactions on Parallel and Distributed Systems*, 3(4):465–476, July 1992.

[7] P. Feautrier. Some efficient solutions to the affine scheduling problem. I. one-dimensional time. *International Journal of Parallel Programming*, 21(5):313–347, Oct. 1992.

[8] W. Kelly and W. Pugh. Minimizing communication while preserving parallelism. In ACM, editor, *FCRC '96: Conference proceedings of the 1996 International Conference on Supercomputing: Philadelphia, Pennsylvania, USA, May 25–28, 1996*, pages 52–60. ACM Press, 1996.

[9] A. W. Lim and M. S. Lam. Maximizing parallelism and minimizing synchronization with affine partitions. *Parallel Computing*, 24(3-4):445–475, May 1998.

[10] P. M. Petersen and D. A. Padua. Static and dynamic evaluation of data dependence analysis techniques:. *IEEE Transactions on Parallel and Distributed Systems*, 7(11):1121–1132, Nov. 1996.

[11] K. Psarris. The Banerjee-Wolfe and GCD tests on exact data dependence information. *Journal of Parallel and Distributed Computing*, 32(2):119–138, Feb. 1996.

[12] J. Ramanujam. Beyond unimodular transformations. *Journal of Supercomputing*, 9(4):365–389, Oct 1995.

[13] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley and Sons, New York, 1987.

[14] M. E. Wolf. Improving locality and parallelism in nested loops. Ph.D. Dissertation CSL-TR-92-538, Stanford University, Dept. Computer Science, Aug. 1992.

[15] M. E. Wolf and M. S. Lam. A loop transformation theory and an algorithm to maximize parallelism. In *IEEE Transactions on Parallel and Distributed Systems*, Oct 1991.

[16] M. Wolfe. Engineering a data dependence test. *Concurrency: Practice and Experience*, 5(7):603–622, Oct. 1993.

[17] W.Shang, E.Hodzic, and Z.Chen. On uniformization of affine dependence algorithms. *IEEETC: IEEE Transactions on Computers*, 45(7):827–840, July 1996.

[18] J. Xue. Automating non-unimodular loop transformations for massive parallelism. *Parallel Comput.*, 20(5):711–728, 1994.

[19] J. Xue. Unimodular transformations of non-perfectly nested loops. *Parallel Computing*, 22(12):1621–1645, Feb. 1997.

[20] F.-B. Zhang. The FPT programming environment. Ph.D. dissertion ELIS-D096.073, University of Ghent, Dept. of Electronics and Information Systems, Ghent, Belgium, Sept. 1996.