A Basis Approach to Loop Parallelization and Synchronization*

Li Liu and Ferng-Ching Lin

Department of Computer Science and Information Engineering
National Taiwan University, Taipei, Taiwan, ROC
E-mail: f8506013@csman.csie.ntu.edu.tw

Abstract

Loop transformation is a crucial step in parallelizing compilers. We introduce the concept of positive coordinate basis for deriving loop transformations. The basis serves to find proper loop transformations to change the dependence vectors into the desired forms. We demonstrate how this approach can systematically extract maximal outer loop parallelism. Based on the concept, we can also construct a minimal set of synchronization vectors, which are deadlock free, to transform the inner serial loops into doacross loops.

1 Introduction

Because loops usually take a large portion of computation time of a program, many researches focus on loop parallelization[12]. Most parallelization methods use vectors to represent the dependencies and do parallelism extraction[8]. The pattern of dependence vectors decides which loop levels can be parallelized. When the number of dependence vectors is large, it may be difficult and inefficient to extract parallelism by considering all the dependence vectors.

Basic transformations, such as loop skewing and loop interchange[11], can be applied to reconstruct the loop to make a better pattern of the dependence vectors such that more levels of the transformed loops can be parallelized. Most of the existing strategies lack a systematic way to combine the basic transformations to produce proper loop transformations. For example, the Parafrase-2 system [6] only gives advices on which loops can be interchanged, and so on, so the users must make their own decisions on what final transformations they want. One way to combine transformations is to take statistical measurements[9]. But, parallelism may still be extracted improperly.

If the extracted parallelism is large enough for the number of processors, we can allocate the computations without any synchronization in each processor. Otherwise, we have to separate the iterations into more parts and different parts of iterations must be synchronized to ensure that the dependencies are preserved. We can use the static synchronization, such as wavefront method or dynamic synchronization to achieve this[11]. Since dynamic synchronization may bring more benefit, we shall try to transform the unparallelized loops into DOACROSS loops. The iterations of a DOACROSS loop are executed in parallel and synchronized by special statements. Because the performance of a DOACROSS loop is significantly affected by the the synchronization overhead, it is important to find a good set of synchronization vectors.

There are three desirable properties for synchronization vectors. First, the synchronization vectors are deadlock free to prohibit deadlock. Second, the number of synchronization vectors is small to reduce message passing. Third, the components of synchronization vectors are not too small. If a processor reaches a wait statement, it is blocked and waits for signals from other processors. If all signals have been received by a processor which reaches a wait statement, it can execute next statement without any delay. A synchronization vector with larger components will decrease the possibility of blocking.

In this paper, we will use the tool *positive coordinate basis* to extract maximal outer loop parallelism and then to minimize the number of synchronization vectors for the inner loops. This pc-basis concept has been used to do loop partitioning by the present authors[4]. If we have a pc-basis for the dependence vectors, the legality of a transformation can be verified on the basis. The time complexity of finding a legal transformation can thus be greatly reduced. Furthermore, dependence vectors may be linearly depen-

*This research was supported by National Science Council of ROC under contract NSC83-0408-E002-064.

dent causing difficulties when we directly transform them to be in the desired pattern. Since the vectors in a pc-basis are linearly independent, to find a proper transformation becomes easier. When the vectors in a pc-basis are transformed to be in a desired form, so are the dependence vectors.

It is well known that a loop with outer loop parallelism is suitable for parallel computation[12,6]. The pc-basis can be used to automatically construct unimodular transformations for such requirements by systematically combining loop reversal, loop skewing and loop interchange. Then for the inner serial loops, we transform them into DOACROSS loops by adding some synchronization statements. The number of vectors in the pc-basis is equal to the rank of the dependence vectors and we can use them to construct a minimal set of synchronization vectors which are deadlock free.

The rest of the paper is organized as follows. In Section 2, we use a typical example to illustrate our approach. In Section 3, we concentrate on how to construct the pc-basis. Fundamental methods of using basic unimodular transformations to modify vectors are shown in Section 4. We then apply a series of these basic transformations on the pc-basis to extract maximal outer loop parallelism in Section 5. In Section 6, we use the pc-basis to construct a minimal set of synchronization vectors and transform the inner serial loops into DOACROSS loops. These transformation constructions have been coded and added in the loop restructuring part of Parafrase-2.

2 Preliminaries and an Illustration

We assume the depth of a loop nest to be n . The set of iterations $I^n = \{(i_1, i_2, \dots, i_n)^t \mid l_k \leq i_k \leq u_k, \text{ for } 1 \leq k \leq n\}$, where l_k and u_k are the boundaries of the i -th level loop, is called the *iteration space*. The execution of a statement S in iteration I_i is called the *instance* of S associated with I_i . The set of instances of all the statements in the loop associated with I_i form a unit of computation which is denoted by C_i . Let $U = (u_1, u_2, \dots, u_n)^t$ and $V = (v_1, v_2, \dots, v_n)^t$. If there is k such that $u_h = v_h$ for $1 \leq h < k$ and $u_k < v_k$, we say $U \prec V$. The relation so defined is the *lexicographical order*. A vector V with $V \succ (0, 0, \dots, 0)$ is said to be *lexicographically positive*. In the semantics of a sequential language, computations are performed by the lexicographical order of the iterations. Suppose $I_i \prec I_j$ and computations C_i and C_j access to the same variables or array elements, we say that C_j depends on C_i and $I_j - I_i$ is called a *dependence vector*. It is clear that every de-

pendence vector is lexicographically positive. Let the number of dependence vectors be m . The $n \times m$ matrix $D_{n \times m}$, whose i -th column is the i -th dependence vector, is called the *dependence matrix*.

Example 2.1:

```
DO i = 1 TO 5
  DO j = 1 TO 5
    DO k = 1 TO 5
      DO l = 1 TO 5
        A(i, j, k+2, l-1) = C(i, j, k, l) + B(i, j, k, l);
        A(i, j, k, l) = A(i-2, j, k, l-1) * B(i, j, k, l)
      ENDDO
    ENDDO
  ENDDO
ENDDO
```

The iteration space of the above loop is $I^4 = \{(i_1, i_2, i_3, i_4)^t \mid 1 \leq i_k \leq 5 \text{ for } 1 \leq k \leq 4\}$. The dependence vectors of the loop are $(0, 0, 2, -1)^t$, $(2, 0, 2, 0)^t$ and $(2, 0, 0, 1)^t$. They form the following dependence matrix with three columns:

$$D_{4 \times 3} = \begin{pmatrix} 0 & 2 & 2 \\ 0 & 0 & 0 \\ 2 & 2 & 0 \\ -1 & 0 & 1 \end{pmatrix}$$

Let M be the unimodular matrix

$$\begin{pmatrix} 1 & 0 & -1 & -2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

which will be constructed in Section 6. When we apply M on the dependence matrix, we get

$$MD_{n \times m} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 2 & 2 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

The transformed dependence vectors are all lexicographically positive and their first two components are zero. So M is legal and parallelizes two outer levels of the loop. Since the outer two levels of the transformed loop are *DOALL* loops, they can be separated into 85 independent parts without any synchronization among them. If the number of processors is not more than 85, we can allocate the iterations without any synchronization to each processor. Otherwise, we must separate the iterations into more parts and the parts must be synchronized to ensure the dependencies. Usually, an iteration is a unit of work assigned to a processor. Thus, the iterations of a loop can be parallelly executed in different processors if their dependence constraints are preserved. Let D be the set of dependence vectors. The iteration I can be executed after the iterations $I - d_i$, d_i in D , have been executed. So, the iteration I must wait for signals from the iterations $I - d_i$, d_i in D . The vectors d_i in D are called the *synchronization vectors*.

For doubly nested loops, there is a dependence decomposition method which produces two synchronization vectors to preserve the dependencies[8]. Each dependence vector can be represented as a linear combination of some basic vectors with positive integer coefficients. The iteration I waits for the iterations $I - w_i$, w_i is a basic dependence vector. However, the basic dependence vectors do not guarantee the deadlock free property. When a positive linear combination of some basic dependence vectors is the zero vector, a circular wait is caused and deadlock will happen. Furthermore, the number of synchronization vectors is not minimized.

Consider the inner two loops in the previous example. If we take $(0, 0, 2, 0)^t$ and $(0, 0, 0, 1)^t$ as the synchronization vectors, which will be constructed in Section 6, then

$$\begin{aligned}(0, 0, 2, 1)^t &= (0, 0, 2, 0)^t + (0, 0, 0, 1)^t \\ (0, 0, 2, 2)^t &= (0, 0, 2, 0)^t + 2 \times (0, 0, 0, 1)^t \\ (0, 0, 0, 1)^t &= (0, 0, 0, 1)^t\end{aligned}$$

Because these two synchronizaiton vectors are linearly independent, any positive linear combination of them can not be the zero vector and deadlock will not happen. Besides, the number of synchronization vectors is equal to the subspace spanned by the dependence vectors, so it is minimized.

We use two statements *WaitSignal* and *SendSignal* to do synchronization. The statement *WaitSignal*(I_1, I_2) means to wait a signal corresponding to the synchronization vector $I_2 - I_1$ from the processor executing iteation I_1 to the processor executing iteration I_2 . The statement *SendSignal*(I_1, I_2) is defined similarly. The final loop obtained is shown below:

```
DOALL i = -14 TO 2
DOALL j = 1 TO 5
DOACR k = max(1, -9-i) TO min(5, 3-i)
DOACR l = max((1-i+k)/2, 1+k) TO min((5-i+k)/2, 5+k)
  WaitSignal((i, j, k-2, l)^t, (i, j, k, l)^t);
  WaitSignal((i, j, k, l-1)^t, (i, j, k, l)^t);
  A(i-k+2×l, j, k+2, -k+l-1) = C(i-k+2×l, j, k, -k+l)
    + B(i-k+2×l, j, k, -k+l);
  A(i-k+2×l, j, k, -k+l) = A(i-k+2×l-2, j, k, -k+l-1)
    × B(i-k+2×l, j, k, -k+l)
  SendSignal((i, j, k, l)^t, (i, j, k+2, l)^t);
  SendSignal((i, j, k, l)^t, (i, j, k, l+1)^t)
ENDDO
ENDDO
ENDDO
ENDDO
```

We now outline our approach. First, we find a unimodular transformation to make as many as possible upper rows of the transformed dependence matrix be

zero. The transformed loop will have maximal outer loop parallelism. Second, we find another unimodular transformation to make the nonzero entries of the transformed dependence matrix positive and the resulting dependence vectors are used to construct a set of synchronization vectors. Last, we combine these two transformations to form a transformation to be applied on the original loop.

3 Positive Coordinate Basis

Let M be a linear transformation on the iteration space. It should not violate the dependence constraints of the original loop. So $MI_2 \succ MI_1$ and hence $M(I_2 - I_1) \succ \mathbf{0}$ if C_2 depends on C_1 . Therefore, a transformation is *legal* if all the dependence vectors are transformed into lexicographically positive vectors.

Definition 3.1: Let $D = \{d_i \mid 1 \leq i \leq m\}$ be the set of dependence vectors with rank r . The set of linearly independent vectors $B = \{b_i \mid 1 \leq i \leq r\}$ is a *positive coordinate basis (pc-basis)* for D if for all $d_i \in D$, $d_i = \sum_{j=1}^r c_{i,j} b_j$, where $c_{i,j} \geq 0$.

Lemma 3.2: Let $B = \{b_j \mid 1 \leq j \leq r\}$ be a positive coordinate basis for the set of dependence vectors $D = \{d_i \mid 1 \leq i \leq m\}$. A transformation M is legal if $Mb_j \succ \mathbf{0}$ for all $b_j \in B$.

Proof: For each d_i in D , $Md_i = M \sum_{j=1}^r c_{i,j} b_j = \sum_{j=1}^r c_{i,j} Mb_j$, $c_{i,j} \geq 0$. It is clear that $Md_i \succ \mathbf{0}$. $\square$

From Lemma 3.2, the legality of M can be justified if all Mb_j , $1 \leq j \leq r$, are lexicographically positive. In the following, we will show that for a set of lexicographically positive vectors, pc-basis always exists and its vectors are also lexicographically positive. The construction of pc-basis is an iterative process. It finds for the first $k-1$ vectors a pc-basis which is then either "extended" or "modified" to be a pc-basis for the first k vectors.

Algorithm 3.3: (Construct a pc-basis for lexicographically positive vector set D)

```
B = {d1}.
FOR k = 2 TO |D|
  IF ( dk is not in Span (B) )
    THEN B = B ∪ {dk}
    ELSE B = Change_Basis ( B, dk )
  ENDF
ENDFOR
```

When d_k is not in the space spanned by the current basis, we simply add d_k into the basis. If d_k is in the space spanned by the current basis, we should use *Change_Basis* to modify the current basis.

Lemma 3.4: Let $B = \{b_1, b_2, \dots, b_t\}$ be a pc-basis for $\{d_1, d_2, \dots, d_{k-1}\}$. If d_k is not in the space spanned by B , then $B \cup \{d_k\}$ is a pc-basis for $\{d_1, d_2, \dots, d_k\}$.

Proof: For $1 \leq i \leq k-1$, $d_i = \sum_{j=1}^t c_j b_j + 0d_k$.
 $d_k = \sum_{j=1}^t 0b_j + 1d_k$, so $B \cup \{d_k\}$ is a pc-basis for $\{d_1, d_2, \dots, d_k\}$. $\square$

Change_Basis first tests whether the coordinates of d_k with respect to the current basis are all non-negative. If there is any negative coordinate, it finds a lexicographically positive vector to replace a vector in the basis until all coordinates become non-negative.

Algorithm 3.5: (*Change_Basis* (B, d_k))

```
Solve the linear system  $B_{n \times r} c = d_k$ .
WHILE( there exists  $c_i < 0$  )
  Find  $c_p > 0$ .
  IF( $c_p b_p + c_i b_i$  is lexicographically positive)
    THEN
       $b_p = c_p b_p + c_i b_i$ ;
       $c_p = 1, c_i = 0$ 
    ELSE
       $b_i = -(c_p b_p + c_i b_i)$ ;
       $c_p = 0, c_i = -1$ 
  ENDIF
ENDWHILE
```

The while loop body in Algorithm 3.5 is to change the current basis such that the modified one is still a pc-basis for $\{d_1, d_2, \dots, d_{k-1}\}$ and the number of non-zero coordinates of d_k is decreased. The next lemma gives the condition for the modified basis to be valid.

Lemma 3.6: Let B be a pc-basis for D . If another basis B' spans the same space and, for all $b_i \in B$, $b_i = \sum_{j=1}^r c'_{i,j} b'_j$, where $c'_{i,j} \geq 0$, then B' is also a pc-basis for D .

Proof: For all $d_k \in D$, $d_k = \sum_{i=1}^r c_{k,i} b_i$, where $c_{k,i} \geq 0$. $d_k = \sum_{i=1}^r c_{k,i} b_i = \sum_{i=1}^r c_{k,i} (\sum_{j=1}^r c'_{i,j} b'_j) = \sum_{i=1}^r \sum_{j=1}^r c_{k,i} c'_{i,j} b'_j = \sum_{j=1}^r (\sum_{i=1}^r c_{k,i} c'_{i,j}) b'_j$ where $\sum_{i=1}^r c_{k,i} c'_{i,j} \geq 0$. Therefore B' is also a pc-basis for D . $\square$

The positive coefficient c_p in Algorithm 3.5 always exists because vectors in the pc-basis and d_k are all lexicographically positive. If $c_p b_p + c_i b_i$ is lexicographically positive, we use it to replace b_p ; otherwise, we use $-(c_p b_p + c_i b_i)$ to replace b_i . Such modification satisfies the precondition of Lemma 3.6 and decrease the number of non-zero coefficients.

Lemma 3.7: Let the lexicographically positive vector set $B = \{b_1, b_2, \dots, b_r\}$ be a pc-basis for the lexicographically positive vector set $D = \{d_1, d_2, \dots, d_{k-1}\}$ and the lexicographically positive vector d_k be in $\text{Span}(B)$. $B' = \text{Change_Basis}(B, d_k) = \{b'_1, b'_2, \dots, b'_r\}$ is a pc-basis for $D \cup \{d_k\}$, where b'_i , $1 \leq i \leq r$, are all lexicographically positive.

Proof: Because $d_k \in \text{Span}(B)$, $d_k = \sum_{j=1}^r c_j b_j$. We make induction on the number of non-zero c_j 's. When the number of non-zero coordinates is 1, $d_k = c_j b_j$ for some j . By the positivity of d_k and b_j , c_j is positive. So B is a pc-basis for $D \cup \{d_k\}$, we can simply take B' as B .

Assume the lemma holds when the number of non-zero c_j 's is t . Suppose now the number of non-zero elements is $t+1$. If $c_j \geq 0$ for all $1 \leq j \leq r$, we can just take B' as B . So we may assume that there exists some $c_i < 0$. Since vectors in the current pc-basis and d_k are all lexicographically positive, there must be some $c_p > 0$, $p \neq i$.

Case 1: $c_p b_p + c_i b_i$ is lexicographically positive.

Let $b''_p = c_p b_p + c_i b_i$ and $b''_m = b_m$ for $1 \leq m \leq r$, $m \neq p$. So $b_p = (b''_p - c_i b''_i)/c_p$ and $b_m = b''_m$ for $1 \leq m \leq r$, $m \neq p$. We have $d_k = \sum_{j \neq p,i} c_j b''_j + b''_p$. The number of non-zero coordinates of d_k in B'' is t .

Case 2: $c_p b_p + c_i b_i$ is not lexicographically positive.

Let $b''_i = (-c_p) b_p + (-c_i) b_i$ and $b''_m = b_m$ for $1 \leq m \leq r$, $m \neq i$. So $b_i = (b''_i + c_p b''_p)/(-c_i)$ and $b_m = b''_m$ for $1 \leq m \leq r$ and $m \neq i$. We have $d_k = \sum_{j \neq p,i} c_j b''_j - b''_i$. The number of non-zero coordinates of d_k in B'' is t .

According to Lemma 3.6, the new basis $B'' = \{b''_m \mid 1 \leq m \leq r\}$ is also a pc-basis for D and the number of non-zero coordinates of d_k in B'' is t . By the induction hypothesis, there exists B' which is a pc-basis for $D \cup \{d_k\}$. $\square$

Example 3.8: (Find a pc-basis for Example 2.1)

Dependence vectors:

$d_1 = (0, 0, 2, -1)^t$, $d_2 = (2, 0, 2, 0)^t$ and $d_3 = (2, 0, 0, 1)^t$.

Iterations:

0. A pc-basis for $(0, 0, 2, -1)^t$:
 $\{(0, 0, 2, -1)^t\}$
1. A pc-basis for $(0, 0, 2, -1)^t, (2, 0, 2, 0)^t$:
 $\{(0, 0, 2, -1)^t, (2, 0, 2, 0)^t\}$
2. A pc-basis for $(0, 0, 2, -1)^t, (2, 0, 2, 0)^t, (2, 0, 0, 1)^t$:
 $\{(0, 0, 2, -1)^t, (2, 0, 0, 1)^t\}$

In the above example, we initialize the pc-basis to be $\{d_1\}$, $b_1 = d_1$. In iteration 1, we just add d_2 ($=b_2$) into the basis because d_2 is not in the space spanned by $\{d_1\}$. In iteration 2, because $d_3 = (2, 0, 0, 1)^t = d_2 - d_1$ is in the space spanned by $\{b_1, b_2\}$, we use *Change_Basis* to modify the basis. In here, the coordinates are 1 and -1. Since $b_2 - b_1$ is lexicographically positive, we select $b'_2 = (2, 0, 0, 1)^t = b_2 - b_1$ and $b'_1 = b_1$. So, $b_1 = b'_1$ and $b_2 = b'_1 + b'_2$. By Lemma 3.6, we know that $\{b'_1, b'_2\}$ is also a pc-basis for $\{d_1, d_2\}$. Since $d_3 = b'_2$, $\{b'_1, b'_2\}$ is a pc-basis for $\{d_1, d_2, d_3\}$.

4 Basic Unimodular Transformations

A linear transformation is called *unimodular* if its corresponding matrix is integral and has determinant 1 or -1 [1]. Let M be a linear transformation. If I' is an iteration within the boundaries of the transformed loop indices, then $M^{-1}I'$ should be in the original iteration space. (Otherwise, I' will do redundant work causing side effects.) Unimodular transforma-

tions match such requirement because their inverses are integral. Our main idea is to apply a series of elementary row operations on the dependence matrix to make the transformed dependence vectors in a suitable form.

Definition 4.1: There are three row operations to be used:

Reversal: $R_{p,c}(A)_{p,j} = cA_{p,j}$, where $c = -1$, and

$$R_{p,c}(A)_{i,j} = A_{i,j} \text{ for } i \neq p.$$

Skewing: $S_{p,q,c}(A)_{p,j} = A_{p,j} + cA_{q,j}$, and

$$S_{p,q,c}(A)_{i,j} = A_{i,j} \text{ for } i \neq p.$$

Interchange: $I_{p,q}(A)_{p,j} = A_{q,j}$, $I_{p,q}(A)_{q,j} = A_{p,j}$, and

$$I_{p,q}(A)_{i,j} = A_{i,j} \text{ for } i \neq p, q.$$

These row operations provide three basic unimodular transformations which actually produce the three well-known loop transformations: loop reversal, loop skewing and loop interchange. We know that if the i -th components of the dependence vectors are all zero, the i -th level of the nested loop can be executed simultaneously. We will focus on how to use the basic unimodular transformations to annihilate as many non-zero components of vectors as possible. The annihilating process of a two-dimensional vector is equivalent to the Euclidean algorithm for finding the greatest common divisor [3].

Algorithm 4.2:

```

M = I;
Apply  $R_{1, \text{sign}(v_1)}$  and  $R_{2, \text{sign}(v_2)}$  on  $\mathbf{v}$  and  $M$ ;
WHILE ( $v_1 \neq 0$ )
   $c = v_2 \text{ DIV } v_1$ ;
  Apply  $S_{2,1,-c}$  on  $\mathbf{v}$  and  $M$ ;
  Apply  $I_{1,2}$  on  $\mathbf{v}$  and  $M$ 
ENDWHILE

```

For more general case, we must find a unimodular matrix of higher dimension rather than two. Those methods proposed in [3] use many vector operations. Our strategy is to work on scalars rather than vectors.

Algorithm 4.3:

```

Use Algorithm 4.2 to find a unimodular matrix
M to annihilate the  $n$ -th component of  $\mathbf{v}$ ;
Find  $c_i, 1 \leq i \leq n-1$ , such that  $\sum_{i=1}^{n-1} c_i v_i$ 
=  $\gcd\{v_i \mid 1 \leq i \leq n-1\}$ ;
Apply  $S_{n,i,c_i}, 1 \leq i \leq n-1$ , on  $\mathbf{v}$  and  $M$ ;
Apply  $S_{i,n,v_i/v_n}, 1 \leq i \leq n-1$ , on  $\mathbf{v}$  and  $M$ .

```

Example 4.4: (Transform an n -dimensional vector)

$$\begin{array}{ll}
\text{Make the } n\text{-th} & \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -3 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 1 \\ 3 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 1 \\ 0 \end{pmatrix} \\
\text{component zero.} & \\
\text{Make the } n\text{-th} & \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & -3 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 1 \\ 3 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 1 \\ 1 \end{pmatrix} \\
\text{component the} & \\
\text{gcd of the first} & \\
\text{n-1 components.} & \\
\text{Make the first} & \begin{pmatrix} 0 & 0 & 3 & -1 \\ -2 & 1 & 6 & -2 \\ -1 & 0 & 4 & -1 \\ 1 & 0 & -3 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 1 \\ 3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \\
\text{n-1 components} & \\
\text{zero.} &
\end{array}$$

So far, we have shown how to use basic unimodular transformations to modify one n -dimensional vector to

be zero in $n-1$ components. In the following section, we will make the dependence matrix have as many zero rows as possible.

5 Outer Loop Parallelism Extraction

For outer loop parallelism, we hope to move the zero components upward and preserve the positivity in the mean time [12]. That is, each transformed dependence vector will be in the form $(0, 0, \dots, 0, d_{n-r+1}, d_{n-r+2}, \dots, d_n)^t$ and is lexicographically positive. This process *internalizes* the dependencies inward.

Lemma 5.1: Let B be a pc-basis for D . If the transformation M internalizes B into the innermost r levels, then M also internalizes the dependence vectors into the innermost r levels.

Proof: Because M internalizes the B into the innermost r levels, for any $\mathbf{b}_j$ in B , $M\mathbf{b}_j$ is lexicographically positive and its first $n-r$ components are 0. For any $\mathbf{d}_i$ in D , $M\mathbf{d}_i = M \sum_{j=1}^r c_{i,j} \mathbf{b}_j = \sum_{j=1}^r c_{i,j} M\mathbf{b}_j$. This is a positive sum of lexicographically positive vectors, hence is also lexicographically positive. For $1 \leq k \leq n-r$, $(M\mathbf{d}_i)_k = (M \sum_{j=1}^r c_{i,j} \mathbf{b}_j)_k = \sum_{j=1}^r (c_{i,j} M\mathbf{b}_j)_k = 0$. So the first $n-r$ components of $M\mathbf{d}_i$ are 0. $\square$

Consider Example 2.1 and Example 3.8 again. We first find a pc-basis for the column vectors of $D_{n \times m}$, $D_{n \times m} = B_{n \times r} C_{r \times m}$. Then, we find a unimodular transformation M to internalize $B_{n \times r}$.

$$MB_{n \times r} = \begin{pmatrix} 1 & 0 & -1 & -2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 2 \\ 0 & 0 \\ 2 & 0 \\ -1 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 2 & 0 \\ -1 & 1 \end{pmatrix}$$

M also internalizes the dependence matrix.

$$MD_{n \times m} = \begin{pmatrix} 1 & 0 & -1 & -2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 2 & 2 \\ 0 & 0 & 0 \\ 2 & 2 & 0 \\ -1 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 2 & 2 & 0 \\ -1 & 0 & 1 \end{pmatrix}$$

We now show how to find the optimal unimodular transformation by way of pc-basis. The idea is to apply Algorithm 4.3 on the vectors in the pc-basis one by one. The dimension of the vectors to be processed must be decreased to ensure that the transformed vectors are not altered again.

Algorithm 5.2: (Construct a unimodular transformation M to annihilate the first $n-r$ components of r linearly independent vectors $\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_r$)

```

M = I;
FOR i = r TO 1
  last = n-r+i,  $\mathbf{v} = M\mathbf{b}_i$ ;
  Annihilate the last-th component of  $\mathbf{v}$ ;
  Find  $c_k$  such that  $\sum_{k=1}^{\text{last}-1} c_k v_k = \gcd\{v_k \mid 1 \leq k \leq \text{last}-1\}$ ;
  Apply  $S_{\text{last},k,c_k}, 1 \leq k \leq \text{last}-1$ , on  $\mathbf{v}$  and  $M$ ;
  Apply  $S_{k,\text{last},v_k/v_{\text{last}}}, 1 \leq k \leq \text{last}-1$ , on  $\mathbf{v}$  and  $M$ 
ENDFOR

```

Theorem 5.3: Let $B_{n \times r}$ be a matrix whose col-

umn vectors are linearly independent. There exists a unimodular matrix M such that the first $n - r$ row vectors of $MB_{n \times r}$ are zero and the last r row vectors of $MB_{n \times r}$ form an $r \times r$ lower triangular matrix with positive diagonal elements.

Example 5.4:(Transform n -dimensional vectors)

1. Make the first 3 components of the second column vector zero by applying row operations on the first 4 rows.

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

2. Make the first 2 components of the first column vector zero by applying row operations on the first 3 rows.

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

With Algorithm 5.2, we can find a unimodular transformation to internalize the pc-basis into the innermost r levels. By Theorem 5.1, the transformation also internalizes the dependence vectors to produce r (which is maximal) outer parallel loops. The time complexity is $O(rn^2)$.

Kumar and others[3] considered the construction of a legal unimodular transformation for a set of vectors which may not be linearly independent. The transformations found there may not be optimal. An algorithm for finding optimal unimodular transformations has been proposed recently in [10]. They use all the dependence vectors one by one to construct a valid transformation. Their algorithm has high time complexity and is for internalization only. We use pc-basis to reduce the time complexity of internalization and intend to systematize more processes of producing valid transformations for more purposes, like the one presented in the following section.

6 PC-Basis for Synchronization Vectors

For constructing a set of synchronization vectors with the good properties mentioned in Section 1, we introduce a special pc-basis as follows.

Definition 6.1: Let $D = \{d_i \mid 1 \leq i \leq m\}$ be the set of dependence vectors with rank r . The set of linearly independent vectors $B = \{b_j \mid 1 \leq j \leq r\}$ is a *positive integer coordinate basis (pic-basis)* for D if for all $d_i \in D$, $d_i = \sum_{j=1}^r c_{i,j} b_j$, where $c_{i,j}$, $1 \leq j \leq r$, are non-negative integers.

Since the vectors in pic-basis are linearly independent, any positive linear combination of them can not be the zero vector. Hence, we can take pic-basis as the synchronization vectors and deadlock will never happen. Besides, the number of synchronization vec-

tors is minimized because it is equal to the dimension of the space spanned by the dependence vectors. It is in general hard to find the pic-basis, but if the dependence vectors span the iteration space and all the components of them are non-negative, pic-basis is easy to construct.

Lemma 6.2: Let $D = \{d_i \mid 1 \leq i \leq m\}$ be a set of r -dimensional vectors with rank r and components of d_i are all non-negative. If $g_j = GCD \{ (d_i)_j \mid 1 \leq i \leq m \}$ and e_j is the r -dimensional standard basis, where $1 \leq j \leq r$. Then the set $B = \{b_j = g_j e_j \mid 1 \leq j \leq r\}$ is a pic-basis for D .

Proof: Since the rank of D is r and all components of vectors in D are non-negative, $g_j > 0$ for all $1 \leq j \leq r$. We have $d_i = \sum_{j=1}^r ((d_i)_j / g_j) g_j e_j = \sum_{j=1}^r ((d_i)_j / g_j) b_j$, where $(d_i)_j / g_j$, $1 \leq i \leq m$ and $1 \leq j \leq r$, are all positive integers. Since all vectors in D can be represented as a linear combination of some vectors in B with positive integer coefficients and the number of vectors in B is equal to the rank of D , B is a pic-basis for D . $\square$

After we apply Algorithm 5.2 to internalize the dependence vectors, the first $n-r$ components of the transformed dependence vectors and the transformed pc-basis all become zero. We may focus on their last r components only. If we can find another unimodular transformation to make the components of the transformed dependence vectors all non-negative, the pic-basis can be constructed by Lemma 6.2 consequently.

Lemma 6.3: Let B be a pc-basis for D . If the elements of $M_{n \times n} B_{n \times r}$ are non-negative, then the elements of $M_{n \times n} D_{n \times m}$ are all non-negative.

Proof: Since B is a pc-basis for D , $D_{n \times m} = B_{n \times r} C_{r \times m}$, where the entries of $C_{r \times m}$ are non-negative. $(M_{n \times n} D_{n \times m})_{i,j} = (M_{n \times n} B_{n \times r} C_{r \times m})_{i,j} = \sum_{k=1}^r (M_{n \times n} B_{n \times r})_{i,k} (C_{r \times m})_{k,j} \geq 0$. $\square$

By Lemma 6.3, we can just use the transformed pc-basis to construct a unimodular transformation to make all the components of the transformed dependence vectors non-negative. The following Algorithm is to find such a unimodular transformation. It scans the components of pc-basis dimension by dimension to decrease the number of negative entries until all become non-negative. Recall that Algorithm 5.2 transforms the pc-basis to be a lower triangular matrix with positive diagonal with $n-r$ zero rows on the top. So, we just do basic unimodular transformations on the last $n-r$ rows.

Algorithm 6.4: (Construct a unimodular transformation $M_{n \times n}$ such that the elements of $M_{n \times n} B_{n \times r}$

are non-negative)

```

 $M_{n \times n} = I;$ 
FOR  $i = n-r$  TO  $n$ 
  FOR  $j = 1$  TO  $i-1$ 
    IF ( $b_{i,j} < 0$ ) THEN
      multiplier =  $\lfloor -b_{i,j}/b_{j,j} \rfloor + 1;$ 
      Apply  $S_{i,j, multiplier}$  on  $M_{n \times n}$  and  $B_{n \times r}$ 
    ENDIF
  ENDFOR
ENDFOR

```

Example 6.5:

The third row:

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

The forth row: Skewing $S_{4,3,1}$ (multiplier = 1).

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

Product of the unimodular matrices obtained from Algorithm 5.2 and Algorithm 6.4 is the final transformation we desire. For example, the transformation matrix for Example 2.1 is

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

When we apply it on the dependence vectors,

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

Components of the resulting dependence vectors are all non-negative. Consider the last two components of the resulting dependence vectors. They are two-dimensional vectors $(2,1)^t$, $(2,2)^t$ and $(0,1)^t$. The dimension of the subspace spanned by them is two and their components are all non-negative. So, they meet the need of the Lemma 6.2 and the pic-basis of the resulting dependence vectors is $\{ (0,0,2,0)^t, (0,0,0,1)^t \}$.

We can take the pic-basis as the set of synchronization vectors which is deadlock free and optimal. While [8] constructs synchronization vectors for doubly nested loops only, our method is general. The synchronization vectors constructed by [8] are in the form $(1, c)$ and $(0, 1)$, where c may be negative. The first non-zero components of our pic-basis is related to the GCD of the dependence vectors. They are positive and not smaller than 1. (The difference between these two methods is similar to that between a serial loop and a blocking loop by the GCD method [5].) Our method obtains more implicit parallelism.

7 Concluding Remarks

Positive coordinate basis is a new concept proposed by the authors[4] to attack the problems in loop restructuring. Basis is the most appropriate tool for

describing the behaviors of vectors. Using pc-basis to derive proper loop transformations is very efficient. In this paper, we have presented the constructions of two unimodular transformations, one for extracting outer loop parallelism, the other for making the none-zero components of dependence vectors positive. We modify the pc-basis to be a pic-basis, which is suitable for optimizing the number of synchronization vectors. More applications of pc-basis are expected to appear.

References

- [1] U. Banerjee, "Unimodular Transformations of Double Loops," *Proc. Third Workshop on Programming Languages and Compilers for Parallel Computing*, 1990.
- [2] D. K. Chen and P. C. Yew, "An Empirical Study on DOACROSS Loops," *Proc. ACM International Conference on Supercomputing*, pp. 620-632, 1991.
- [3] K. G. Kumar, D. Kulkarni and A. Basu, "Generalized Unimodular Loop Transformations for Distributed Memory Multiprocessors," *Proc. 1991 International Conf. on Parallel Processing*, Vol. 2, pp. 146-149, 1991.
- [4] L. Liu and F. C. Lin, "Two-Level Loop Partitioning Based on the Concept of Positive Coordinate Basis," to be presented at *International Workshop on Massively Parallelism: Hardware, Software and Applications*, Capri, Italy, Oct. 1994.
- [5] C. D. Polychronopoulos, *Parallel Programming and Compilers*, Kluwer Academic Publishers, Boston, 1988.
- [6] C. D. Polychronopoulos, M. B. Girkar, M. R. Haghighat, C. L. Lee, B. Leung and D. A. Schouten, "Parafuse-2: An Environment for Paralleling, Partitioning, Scheduling Programs on Multiprocessors," *Proc. 1989 International Conf. on Parallel Processing*, Vol. 2, pp. 39-48, 1989.
- [7] A. Schreiber, *Theory of Linear and Integer Programming*, John Wiley and Sons, 1986.
- [8] T. H. Tzen and L. M. Ni, "Data Dependence Analysis and Uniformization for Doubly Nested Loops," *Proc. 1992 International Conf. on Parallel Processing*, Vol. 2, pp. 91-99, 1992.
- [9] D. Whitfield and M. L. Soffa, "Investigating Properties of Code Transformations," *Proc. 1993 International Conf. on Parallel Processing*, Vol. 2, pp. 156-160, 1993.
- [10] M. E. Wolf and M. S. Lam, "A Loop Transformation Theory and an Algorithm to Maximize Parallelism," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 2, No. 4, pp. 452-471, Oct. 1991.
- [11] M. J. Wolfe, *Optimizing Supercompilers for Supercomputers*, The MIT Press, 1989.
- [12] H. Zima, *Supercompilers for Parallel and Vector Computers*, ACM Press, 1990.