Loop Transformation
Using Nonunimodular Matrices

Agustín Fernández, José M. Llabería, Member, IEEE, and Miguel Valero-García, Member, IEEE

Abstract—Linear transformations are widely used to vectorize and parallelize loops. A subset of these transformations are unimodular transformations. When a unimodular transformation is used, the exact bounds of the transformed loop nest are easily computed and the steps of the loops are equal to 1. Unimodular loop transformations have been widely used since they permit the implementation of many useful loop transformations. Recently, nonunimodular transformations have been proposed to reduce communication requirements or to use the memory hierarchy efficiently. The methods used for unimodular transformations do not work in the case of nonunimodular transformations, since they do not produce the exact bounds of the transformed loop nest. In this paper, we present a method for nested loop transformation which gives the exact bounds for both unimodular and nonunimodular transformations. The basic idea is to use the Hermite Normal Form (HNF) of the transformation matrix.

Index Terms—Iteration space, Hermite Normal Form, linear transformations, loop transformations, nonunimodular transformations, unimodular transformations.

I. INTRODUCTION

The transformation of nested loops is a widely used technique for program vectorization and parallelization. Linear transformations permit the specification of a wide range of proposed nested loop transformations, such as loop interchange, reversal, skewing, and their composition [23].

A linear transformation specifies a time and space distribution of the operations to be performed, preserving the data dependences of the algorithm. Operations assigned to a given time instant can be performed in parallel if they are assigned to different space points. After the transformation, some of the nested loops determine the space mapping and the rest of them (which can be one or more [19]) specify the temporal ordering for the operations assigned to the same space point [11], [12].

The nested loop transformations more frequently used for vectorization and/or parallelization [23] can be specified by unimodular transformations (matrices whose determinant is equal to ±1). In [2], [23] methods are proposed to rewrite in a systematic way the loop nest according to the required unimodular transformation. These methods give the exact bounds for all the loops in the nest.

Recently, some loop transformations whose objective is to reduce communication or to use the memory hierarchy efficiently are nonunimodular transformations [8], [14]. In the case of a nonunimodular transformation, the techniques proposed in [2], [23] do not compute the exact bounds of the loops and some conditional statements must be included to determine if a given iteration of the loop nest must be executed [10]. These conditional statements introduce a significant run time overhead.

In this paper, we present a method for code transformation using nonunimodular transformations. The method gives the exact bounds and step of every loop, eliminating the need of conditional statements.

First, we present an overview of linear transformations (Section II). Next we present some examples in which nonunimodular transformations are required (Section III). In Section IV, we discuss nonunimodular transformations, pointing out their main features and showing how they can be characterized by the Hermite Normal Form of the transformation matrix. Finally, in Section V, we present our method to modify the original code when the transformation matrix is nonunimodular.

II. FRAMEWORK

We consider algorithms specified by a number of perfectly nested loops, all of them with step equal to 1. The bounds of the loops have the form:

\[ \text{for } i_1 = L_{i_1}, \ldots, U_{i_1}, \text{ do} \]

where \( L_i = \max(l_{i,1}, l_{i,2}, \ldots, l_{i,p}) \), \( U_i = \min(u_{i,1}, u_{i,2}, \ldots, u_{i,q}) \) and \( l_{i,j}, u_{i,j} \) are linear functions which have the form:

\[ l_{i,j} = \sum_{k=1}^{i-1} \alpha_{i,j}^k \cdot i_k + \alpha_{i,j} \]

\[ u_{i,j} = \sum_{k=1}^{i-1} d_{i,j}^k \cdot i_k + \delta_{i,j} \]

where \( \alpha_{i,j}^k \) and \( d_{i,j}^k \) (1 ≤ k ≤ i) are constants, and \( \alpha_{i,j} \) and \( \delta_{i,j} \) can be constants or parameters (problem size). If the steps of the bounds are different from 1, it is necessary to normalize the loops. In [17] a method can be found to perform this normalization in a systematic way.

Any single iteration of the n loops can be represented by an n-dimensional vector,

\[ I = (i_1, \ldots, i_n)^t \]

where each component of this vector is associated with one of the loops. The component on the left of the vector is associated with the outermost loop and the component on the right with the innermost loop.

The set of iterations determined by the bounds of the n nested loops is a convex subset of \( \mathbb{Z}^n \). We call this set the Bounded Iteration Space (BIS):
BIS = \{ I = (i_1, \ldots, i_n)' \mid L_1 \leq i_1 \leq U_1, \ldots, L_n \leq i_n \leq U_n \}

We will use the term Iteration Space (IS) to refer to the unbounded iteration space (lattice of integer points of \( \mathbb{Z}^n \)):

\[ IS = \{ I = (i_1, \ldots, i_n)' \mid i_j \in \mathbb{Z} \} \]

Fig. 1 shows an example of loop nest (1a) and the corresponding BIS (1b).

(a) for \( i_1 = 0, 4 \) do
   for \( i_2 = 0, 6 \) do
     \( x[i_1, i_2] = f(x[i_1-1, i_2]) \)
     \( b[i_1, i_2] = g(b[i_1, i_2-1], x[i_1, i_2]) \)
   endfor
endfor

(b) \hspace{1cm}

Fig. 1. (a) Example of loop nest. (b) Bounded iteration space (BIS) for this loop nest.

The BIS can be specified in a matrix form [5], [6]. Taking into account the semantics of a for loop, each one of the values \( l_{i,j} \) and \( u_{i,j} \) defines an inequality of the form:

\[
\sum_{k=1}^{l_{i,j}} a_{i,j}^k \cdot i_k + \alpha_{i,j} \leq i_k \leq \sum_{k=1}^{u_{i,j}} d_{i,j}^k \cdot i_k + \delta_{i,j}
\]

Putting all these inequalities together, the following matrix inequality can be built:

\[
A \cdot I \leq \beta
\]

Every row of matrix \( A \) defines a lower bound \( l_{i,j} \) (or an upper bound \( u_{i,j} \)); it is built from coefficients \( a_{i,j}^k \) and \( -1 \) (or \( d_{i,j}^k \) and \( 1 \)). The \( n \) elements of vector \( I \) are the iteration control variables \( i_k \), and \( \beta \) is a vector whose components are the coefficients \( \alpha_{i,j} \) and \( \delta_{i,j} \).

As an example, the bounds of the BIS shown in Fig. 1 are represented by the following matrix inequality:

\[
\begin{align*}
0 \leq i_1 & \quad i_1 \leq 4 \\
0 \leq i_2 & \quad i_2 \leq 6
\end{align*}
\]

\[
\begin{bmatrix}
1 & 0 \\
0 & 1 \\
-1 & 0 \\
0 & -1
\end{bmatrix}
\begin{bmatrix}
i_1 \\
i_2
\end{bmatrix}
\leq
\begin{bmatrix}
4 \\
6 \\
0 \\
0
\end{bmatrix}
\]

When a data item (i.e., a memory word) is accessed several times the relative order of reads and writes must be preserved.

In particular, read-write, write-read, and write-write sequences must be preserved. This fact is characterized through the concept of dependence. When the code is transformed for vectorization and/or parallelization, the three types of dependences mentioned above must be preserved. Moreover, when it is important to use efficiently the memory hierarchy the locality of the program must be exploited. In these cases it is important to identify dependences read-read; these types of dependences are called input dependences.

The dependences are represented by \( n \) dimensional vectors called dependence vectors

\[ d = (\delta_1, \ldots, \delta_n)' \]

Each component of a dependence vector is associated with one of the nested loops, from the outermost loop to the innermost loop. The dependence vectors can provide different types of information (distances, \( \delta_i \in \{>, <, =\} \)), etc.) depending on the objectives of the dependence analysis [22]. If the dependence vector provides distances, then each component of the vector is the number of iterations of the corresponding loop, separating two consecutive accesses to the data item associated to the dependence.

Given a loop nest to perform certain computation, the dependence vectors are always lexicographically positives \( (d_i > 0) \). This means that if the dependence vectors provide distances then, the first nonnull component of the vector is positive, and if the dependence vector provides directions (following the notation proposed by Wolfe [22]), the first component different from '=' must be equal to '<'.

The dependences in the example shown in Fig. 1a are \( d_1 = (1, 0)' \) and \( d_2 = (0, 1)' \). The dependence graph shows the iterations of the BIS (dots in Fig. 1b), which represent computations, and edges between them, which represent the dependence vectors.

A transformation, represented by matrix \( T \), maps each iteration \( I = (i_1, \ldots, i_n)' \) of the \( n \)-dimensional iteration space (IS) to one iteration \( J = (j_1, \ldots, j_n)' \) of the transformed \( n \)-dimensional iteration space (TIS):

\[
TIS = \{ J = T \cdot I \mid I \in \mathbb{Z}^n \}
\]

\( T \) is a valid transformation only if:

1) \( T \) is nonsingular
2) Dependences of type read-write, write-read, and write-write are preserved. The dependences are preserved if the transformed dependence vectors are lexicographically positives.

\[ T \cdot d_i > 0 \text{ (forall \( d_i \) )} \]

where \( d_i \) are the dependence vectors.

If \( T \) is unimodular (its determinant is \( \pm 1 \)), then both \( T \) and \( T^{-1} \) map points with integer coordinates (integer points for short) into integer points. In this case the TIS is equal to \( \mathbb{Z}^n \).

Nevertheless, if \( T \) is nonunimodular (its determinant is different from \( \pm 1 \)) the coefficients of \( T^{-1} \) belong to \( \mathbb{Q} \). Then \( T^{-1} \) maps some integer points of the TIS into points which do not have integer coordinates and therefore do not belong to the
IS. Those points of the TIS without integer antimage in the IS are called holes. The loop nest required to implement transformation $T$ must skip these holes of the TIS.

As an example, Figs. 2a and 2b show a piece of the TIS obtained when applying the transformation matrices:

$$T_1 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \quad |T_1| = 1 \quad T_2 = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \quad |T_2| = 3$$

to the IS shown in Fig. 1.

In these figures, only points represented by white or black dots have integer antimage. The rest are holes. Since $T_1$ is unimodular, the TIS shown in Fig. 2a does not have holes.

![Fig. 2. (a) TIS obtained through a unimodular transformation. (b) TIS obtained through a nonunimodular transformation. White and black dots represent points of the TIS with integer antimage. The black dots represent the points of the BTIS with integer antimage.](image)

The image of the bounds of the BIS when applying a transformation $T$ can be put in a matrix form, using the transformation matrix $T$ and the matrix inequation which represents the bounds of the BIS:

$$A \cdot J \leq \beta \quad \Rightarrow \quad A \cdot T^{-1} \cdot J \leq \beta \quad \Rightarrow \quad A' \cdot J \leq \beta$$

Since the BIS is convex, the above matrix inequation also delimits a convex space. This space is the minimum convex space which contains all the points of the TIS whose antimage belongs to the BIS. We call this minimum convex space as the Bounded Transformed Iteration Space (BTIS).

The bounds of the BTIS can be extracted from the matrix inequation $A' \cdot J \leq \beta$ by using the Fourier-Motzkin algorithm. This algorithm, or slightly different versions of it, has already been proposed to the same purpose by other authors [1], [5], [6].

The Fourier-Motzkin algorithm (shown in Appendix A) consists in a loop which iterates $n - 1$ times, where $n$ is the number of elements of vector $J$. In each iteration of this loop the bounds of the convex closure of BTIS for one of the dimensions is obtained. The order in which these bounds are computed is important. They must be computed from the innermost loop to the outermost loop.

If the rows of $A'$ can be grouped into two groups in such way that all the rows in one of the groups have the form $(\ast, \ldots, \ast, 0, \ldots, 0)$ and all the rows in the other group have the form $(\ast, \ldots, \ast, -1, 0, \ldots, 0)$ then the Fourier-Motzkin algorithm is not required since the bounds of the BTIS can be obtained in a trivial way (the matrix $A$ which defines the BIS has this form). This particular form of matrix $A'$ has been called triangular (or pseudotriangular) domain [5], [6].

When $T$ is unimodular all the integer points of the BTIS have integer antimage in the BIS. Therefore, the transformed loop nest must scan all the integer points of the BTIS. In this case, the bounds given by the Fourier-Motzkin algorithm can be directly used to build the loop nest required to scan the BTIS (we say that these bounds are exact). In Fig. 2a, the black dots represent the points of the BTIS which integer antimage, in a case in which $T$ is unimodular.

When $T$ is nonunimodular there are holes in the BIS and in consequence, in the BTIS and, in particular, in the boundaries of the BTIS. To scan correctly the BTIS, all the holes must be skipped. In particular the bounds of the BTIS obtained through the Fourier-Motzkin algorithm, must be corrected to obtain the exact bounds of the BTIS, required to build the transformed loop nest. Fig. 2b shows an example in which the dashed lines represent the bounds of the BTIS obtained through the Fourier-Motzkin algorithm, and the continuous lines, which bounds shadowed area, are the exact bounds of the BTIS. Finally, the black dots are the points of the BTIS with integer antimage.

In this paper we address the problem of correcting in a systematic way the bounds of the BTIS given by the Fourier-Motzkin algorithm in order to produce the exact bounds of the BTIS.

### III. Previous Work on Nonunimodular Transformations

There are several cases in which a nonunimodular transformation can be required:

- There are some type of systolic algorithms which are derived through nonunimodular transformations. As an example, in [20] the authors describe the required code transformation for programming a systolic algorithm of this type (for matrix-by-vector multiplication) on a systolic processor.
- The technique called loop tiling requires a previous transformation of the original code. One of the criterion to select the transformation matrix is to minimize communication among tiles. Ramanujan shows in [14] that a nonunimodular matrix may be required for this objective.
The access ordering to the elements of a vector or matrix can be modified through a linear transformation to optimize the use of the memory hierarchy. This type of code transformation has been called access normalization by Li and Pingali [8]. In some cases, the required transformation matrix has to be nonunimodular.

Some temporal transformations such as slow-down and retiming are useful for systolic algorithm partitioning. These transformations can be represented through nonunimodular matrices [21].

Some authors have studied the problem of rewriting the original code to implement a nonunimodular transformation. As mentioned in the introduction of the paper, in [10] the authors propose to include conditional statements in the transformed code to determine if a given point of the transformed space has an integer antiimage in the original space. These statements can increase significantly the execution time.

In [3] the authors propose to correct the bounds obtained through the Fourier-Motzkin algorithm to produce the exact bounds. An example is given but no general solution is proposed.

Two general solutions have been proposed to solve the problem addressed in this paper. Both solutions were developed independently to our work.

Li and Pingali propose in [9] a general solution to the problem. They propose to decompose matrix $T$

$$T = H \cdot C$$

where $H$ is a Hermite Normal Form and $C$ is unimodular. The method to rewrite the code consists of two steps. In the first step the unimodular matrix $C$ is applied to produce an auxiliary iteration space. The exact bounds of this space are obtained through the Fourier-Motzkin algorithm, since $C$ is unimodular. In a second step, the auxiliary iteration space is transformed using $H$. Since $H$ is lower triangular, it is easy to convert the bounds of the auxiliary space into the exact bounds of the BTIS.

The solution given by Ramanujan in [15] only works for two dimensional loop nests. However, in [16] a general solution is given which is based on the use of the Hermite Normal Form of $T$ and some Hesseberg matrices.

IV. NONUNIMODULAR TRANSFORMATIONS

In this section we describe the method proposed for code rewriting when using nonunimodular transformation matrices.

When $T$ is nonunimodular the bounds obtained through the Fourier-Motzkin algorithm are not exact. The method proposed here gives a general expression for correcting these bounds and produces the exact bounds of the BTIS. This expression is built by composition of the expression obtained through the Fourier-Motzkin algorithm and nonlinear functions [4], [20]. Once the exact bounds are obtained, the holes of the BTIS are skipped by using steps greater than 1 in the different loops.

In the following, we present Theorems 1 and 2. These theorems were introduced in [13], [18] and permit the use of the Hermite Normal Form (HNF) of the transformation matrix $T$ to characterize the TIS. This characterization is given in Lemmas 1 and 2 and in Corollary 1. Then, Lemma 3 and Corollary 2 give a characterization of the BTIS.

**Definition 1.** The set:

$$L(T) = \{ y \in \mathbb{Z}^n, y = T \cdot x \mid x \in \mathbb{Z}^n \}$$

where $T$ is an $n$-by-$n$ integer matrix, is called the lattice generated by the columns of $T$. This set represents the set of points transformed by matrix $T$.

**Theorem 1.** Let $T$ be an $n$-by-$n$ integer matrix and $C$ be an $n$-by-$n$ unimodular matrix, (which represents elementary column operations), then [13]:

$$L(T \cdot C) = L(T)$$

**Definition 2.** Let $H$ be an $n$-by-$n$ matrix with the following features: 1) it is lower triangular, 2) it is nonnegative, 3) the maximum value of each row is unique and belongs to the diagonal of $H$. Then $H$ is called a Hermite Normal Form (HNF) [13], [18]. This matrix represents a lattice minimally.

**Theorem 2.** Any matrix $T$, rational and nonsingular, can be transformed in its HNF in the following way:

$$H = T \cdot C_1 \cdot C_2 \cdots \cdot C_p = T \cdot C$$

where $C_i$ are unimodular matrices which represent elementary column operations. As a result, matrices $T$ and $H$ generate the same lattice, $L(T) = L(H)$ [18].

The lower triangular structure of matrix $H$ makes easy the characterization of TIS. We first introduce this characterization through an example.

Fig. 3 shows a part of the TIS shown in Fig. 2b. The HNF $H$ of $T_2$ is:

$$H = \begin{bmatrix} h_{11} & 0 \\ h_{21} & h_{22} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 2 & 3 \end{bmatrix}$$

In the original definition given in [13], [18] the elements below the diagonal are negative. In our case for convenience, we require them to be positive.
The white and black dots of the lattice in Fig. 3 have integer antiimage. If \( J = (j_1, j_2, \ldots, j_k) \) is a white or black dot \((H^{-1} \cdot J = I) \in \mathbb{Z}^2 \) then
\[
\begin{align*}
  j_1 &= h_{11} \cdot c_1 \\
  j_2 &= h_{22} \cdot c_2 + \text{gap}_2
\end{align*}
\]
where \( c_1 \) and \( c_2 \) are integer numbers and
\[
\text{gap}_2 = \left( \frac{h_{11} \cdot j_1}{h_{22}} \right) \mod h_{22}
\]
As an example, one of these white and black points is \((5,7)\).

We can write:

\[
(5,7) = (5 \cdot h_{11}, 2 \cdot h_{22} + 2 \cdot 5 \mod 3) = (5 \cdot 1, 2 \cdot 3 + 1)
\]

In general, for an \( n \)-dimensional TIS, given a point with integer antiimage, its coordinate \( j_k \) is the sum of two terms. The first term is a multiple of \( h_{kk} \), this is shown in Lemma 1. The second term, which we call \( \text{gap}_k \), is an expression involving coordinates from \( j_1 \) to \( j_{k-1} \). This expression is introduced in Lemma 2 and Corollary 1.

**Lemma 1.** Let \( H \) be an HNF. Let \( J = (j_1, \ldots, j_k, \ldots, j_k) \) be a point of the space transformed by \( H \) such that the point \( H^{-1} \cdot J \) belongs to \( Z^2 \).

The point, \( H^{-1} \cdot J' \) where \( J' \) is any point of the set
\[
\left\{ (j_1, \ldots, j_k + m, \ldots, j_k) \mid m = 1, \ldots, h_{kk} - 1 \right\}
\]
does not belong to \( Z^2 \). Moreover, a point \( H^{-1} \cdot J' \), where \( J' \) is a point of the set \( \left\{ (j_1, \ldots, j_k + h_{kk}, \ldots) \right\} \), may belong to \( Z^2 \).

**Proof.** Let \( I = (i_1, \ldots, i_k, \ldots, i_k) \) be the point belonging to \( Z^2 \) such that \( I = H^{-1} \cdot J \). Since \( H \) is lower triangular, the first \( k-1 \) components of \( H^{-1} \cdot J' \) are equal to the first \( k-1 \) components of \( I \) and the \( k \)th component is:
\[
\frac{j_k + m - \sum_{r=1}^{k-1} h_{rr} \cdot i_r}{h_{kk}} = i_k + \left( \frac{m}{h_{kk}} \right)
\]
Since, \( 1 \leq m \leq h_{kk} - 1 \), \( i'_k \) does not belong to \( Z \). Therefore, the points \( H^{-1} \cdot J' \) do not belong to \( Z^2 \).

If \( m = h_{kk} \), then \( i'_k \) belongs to \( Z \). Therefore, a point \( H^{-1} \cdot J' \) may belong to \( Z^2 \), since the first \( k \) components are integer values.

**Lemma 2.** Let \( H \) be an HNF. Let \( J = (j_1, \ldots, j_k, 0, \ldots, 0) \) be a point of the space transformed by \( H \) such that the first \( k-1 \) components of point \( I = H^{-1} \cdot J \) belong to \( Z \).

Let \( J' = (j_1, \ldots, j_k, j'_k, \ldots, j'_k) \) be the closest point in the lexicographic order, to \( J \), such that the first \( k \) components of \( I' = H^{-1} \cdot J' \) belong to \( Z \). The \( k \)th component of point \( J' \) is:
\[
\left( \sum_{r=1}^{k-1} h_{rr} \cdot i_r \right) \mod h_{kk}
\]
We call this value \( \text{gap}_k \).

**Proof.** Since \( H \) is lower triangular, the first \( k-1 \) components of \( H^{-1} \cdot J \) are equal to the first \( k-1 \) components of \( H^{-1} \cdot J' \).

Moreover, the \( k \)th component of \( I' = H^{-1} \cdot J' \) is:
\[
\frac{j'_k - \sum_{r=1}^{k-1} h_{rr} \cdot i_r}{h_{kk}} \mod h_{kk} = \frac{\left( \sum_{r=1}^{k-1} h_{rr} \cdot i_r \right)}{h_{kk}} = \frac{h_{kk} \cdot \left( \sum_{r=1}^{k-1} h_{rr} \cdot i_r \right) \mod h_{kk}}{h_{kk}} = \frac{h_{kk}}{h_{kk}} \left( \sum_{r=1}^{k-1} h_{rr} \cdot i_r \right) \mod h_{kk}
\]
Therefore, \( i'_k \) is integer value only if:
\[
\left( \sum_{r=1}^{k-1} h_{rr} \cdot i_r \right) \mod h_{kk} = 0
\]

**Corollary 1.** Let \( H \) be an HNF. Let
\[
J = (j_1, \ldots, j_{k-1}, 0, \ldots, 0)'
\]
be a point of the space transformed by \( H \) such that the first \( k-1 \) components of point \( I = H^{-1} \cdot J \) belong to \( Z \).

Let \( J' = (j_1, \ldots, j_{k-1}, j'_k, j'_k+1, \ldots, j'_k) \) be the closest point in the lexicographic order to \( J \), such as the point \( I' = H^{-1} \cdot J' \) belongs to \( Z^n \). The components from \( k \) to \( n \) of point \( J' \) are:
\[
\left( \sum_{r=1}^{k-1} h_{kr} \cdot i_r \right) \mod h_{kk} \quad k \leq g \leq n
\]

**Proof.** Apply Lemma 2 to components from \( k \) to \( n \).

The following lemma and corollary give a characterization of BTIS. First, let us go back to the example shown in Fig. 3. In this figure, the dashed lines are the boundary of the convex closure of the BTIS.

For a given value of coordinate \( j_1 \), the first point included in the BTIS is \((j_1, I^2_2)\). The value \( I^2_2 \) is the lower bound obtained through the Fourier-Motzkin algorithm; it is a function of \( j_1 \). To obtain the exact bound of the BTIS we have to find the value such that:
\[
j_2 = h_{22} \cdot c_2 + \text{gap}_2
\]
and \( j_2 \) is the minimum value such that
\[
I^2_2 \leq j_2
\]
In Fig. 3, for \( j_1 = 3 \), \((3, I^2_2 = (3,3/2))\), and the exact bound of the BTIS is \((3,3)\). Lemma 3 and Corollary 2 shows how to obtain the exact lower bound of the BTIS.

**Lemma 3.** Let \( T \) be a transformation matrix and \( A' \cdot J \leq B \) be the matrix inequation which describe the bounds of the BTIS. Finally, let \( H \) be the HNF of \( T \).

Let \( J = (j_1, \ldots, j_{k-1}, \ldots) \) be a point such that the first \( k-1 \) components of \( I = H^{-1} \cdot J \) belong to \( Z \), and assume that
\( j_1, \ldots, j_{k-1} \) are within the bounds determined by the convex closure of the BTIS for each component. Now, let \( I_k^* \) be the lower bound of the convex closure of the BTIS, along dimension \( k \). The closest point \( J' \), in the lexicographic order to \((j_1, \ldots, j_{k-1}, L_k^*, \ldots)^t\), such that the first \( k \) components of \( H^{-1} \cdot J' \) belong to \( \mathbb{Z} \) is:

\[
  J' = (j_1, \ldots, j_{k-1}, j'_k, \ldots)^t
\]

where

\[
  j'_k = \left[ \frac{L_k^* - \text{gap}_k}{h_{kk}} \right] h_{kk} + \text{gap}_k
\]

\[
  \text{gap}_k = \left( \sum_{r=1}^{k-1} h_{kr} \cdot i_r \right) \mod h_{kk}
\]

\[
  = \left( \sum_{r=1}^{k-1} h_{kr} \cdot \sum_{d=1}^r h_{rd} \cdot j_d \right) \mod h_{kk}
\]

and \( h_{ij}' \) are the elements of matrix \( H^{-1} \).

PROOF. From Theorem 2 we know that \( H = T \cdot C \), where \( C \) is a unimodular matrix. As a result we can rewrite \( T \cdot I = J \) as a function of matrix \( H \):

\[
  H \cdot C^{-1} \cdot I = J = C^{-1} \cdot H^{-1} \cdot I
\]

Therefore, since \( C^{-1} \) is unimodular, the components of \( I \) are integer only if the components of vector \( H^{-1} \cdot J \) are integer.

By Lemmas 1 and 2, the \( k \)th component of vector \( H^{-1} \cdot J \), with \( J = (j_1, \ldots, j_{k-1}, j'_k, \ldots)^t \), belongs to \( \mathbb{Z} \) only if \( j'_k \) has the form:

\[
  c \cdot h_{kk} + \text{gap}_k
\]

where \( c \) is any integer value and

\[
  \text{gap}_k = \left( \sum_{r=1}^{k-1} h_{kr} \cdot i_r \right) \mod h_{kk}.
\]

Then we can write:

\[
  I_k^* = I_k^* - \text{gap}_k + \text{gap}_k
\]

\[
  = \left[ \frac{L_k^* - \text{gap}_k}{h_{kk}} \right] h_{kk} + (I_k^* - \text{gap}_k) \mod h_{kk}
\]

\[
  + \text{gap}_k \leq \left[ \frac{L_k^* - \text{gap}_k}{h_{kk}} \right] h_{kk} + \text{gap}_k
\]

As a result:

\[
  J' = (j_1, \ldots, j_{k-1}, \left[ \frac{L_k^* - \text{gap}_k}{h_{kk}} \right] h_{kk} + \text{gap}_k, \ldots)^t
\]

is the closest point to \((j_1, \ldots, j_{k-1}, L_k^*, \ldots)^t\), in the lexicographic order, such that the first \( k \) components of \( H^{-1} \cdot J' \) belong to \( \mathbb{Z} \). \( \square \)

COROLLARY 2. Let \( T \) be a transformation matrix, \( T \cdot J \leq \beta \) the matrix inequation which describe the boundaries of the BTIS and \( H \) the NNF of \( T \).

Assume we know the first \( k-1 \) components of a given point \( J = (j_1, \ldots, j_{k-1}, \ldots)^t \) such that the first \( k-1 \) components of \( I = H^{-1} \cdot J \) belong to \( \mathbb{Z} \), and these components are within the bounds of the BTIS for each component.

Let \( J' = (j_1, \ldots, j_{k-1}, j'_k, j'_{k+1}, \ldots)^t \) be the closest point to \((j_1, \ldots, j_{k-1}, L_k^*, \ldots, L_{x_0})^t\), in the lexicographic order such that components from \( j'_k \) to \( j'_n \) are within the corresponding bounds of the BTIS and \( H^{-1} \cdot J' \) belong to \( \mathbb{Z} \). Then the values of components from \( j'_k \) to \( j'_n \) of point \( J' \) and \( \text{gap}_k \) to \( \text{gap}_n \) are:

\[
  j'_k = \left[ \frac{L_k^* - \text{gap}_k}{h_{kk}} \right] h_{kk} + \text{gap}_k
\]

\[
  \text{gap}_k = \left( \sum_{r=1}^{k-1} h_{kr} \cdot i_r \right) \mod h_{kk}
\]

PROOF. Apply Lemma 3 to components from \( k \) to \( n \). \( \square \)

V. CODE TRANSFORMATION

Now, we show the code transformation corresponding to a nonunimodular transformation \( T \).

THEOREM 3. Given the following nested loop structure:

\[
\text{for } i_1=1:t_1, u_1 \text{ do}
\]

\[
\text{for } i_2=1:t_2, u_2 \text{ do}
\]

\[
\ldots
\]

\[
\text{for } i_n=1:t_n, u_n \text{ do}
\]

\[
A(\{i_1, i_2, \ldots, i_n\})
\]

\[
\text{endfor}
\]

\[
\ldots
\]

\[
\text{endfor}
\]

we want to transform this code with the transformation matrix \( T \).

1) The step of loop \( k \) is required to cover the BTIS is element \( \text{gap}_k \) of \( H \) (the HNF of \( T \)).

2) The initial value for the control variable \( j_k \) (loop \( k \)) is:

\[
  \left[ \frac{L_k^* - \text{gap}_k}{h_{kk}} \right] h_{kk} + \text{gap}_k
\]

where \( L_k^* \) is the lower bound of the convex closure along dimension \( k \) and

\[
  \text{gap}_k = \left( \sum_{r=1}^{k-1} h_{kr} \cdot \sum_{d=1}^r h_{rd} \cdot j_d \right) \mod h_{kk}
\]

where \( h_{ij}' \) are the elements of \( H^{-1} \) and \( j_d (1 \leq d \leq n) \) are the control variables for the loops which cover the BTIS.

3) The indexing functions for the variables of the algorithm are transformed into:

\[
  f(T^{-1} \cdot (j_1, j_2, \ldots, j_n))
\]

where \( j_k (1 \leq k \leq n) \) are the control variables for the loops which cover the BTIS.
Therefore, if $L_k^T$ and $U_k^T$ are the bounds of the BTIS, obtained by the Fourier-Motzkin algorithm, along dimension $k$ then the transformed code is:

```
  for $j_2 = L_k^T, U_k^T, \text{step}=h_{l_k}$ do
    gap = $(1 \times h_{l_k}, j_1)$ mod $h_{l_k}$
    for $j_1 = (L_1^* - \text{gap}) / h_{l_k} \times h_{l_k} + \text{gap}_1, U_1^*, \text{step}=h_{l_k}$ do
      gap = $(\sum_{(c)} h_c, \sum_{(d)} h_d, \sum_{(f)} h_f) \mod h_{l_k}$
      for $j_3 = (L_3^* - \text{gap}_3) / h_{l_k} \times h_{l_k} + \text{gap}_3, U_3^*, \text{step}=h_{l_k}$ do
        $\ldots$
        $A = \{j_1, j_2, \ldots, j_k, \ldots, j_a\}$
      endfor
    endfor
  endfor
```

PROOF. The theorem is a direct result of Lemma 1 and Corollary 2.

Because of the semantics of the for loop, it is not necessary to compute the exact upper bound of the BTIS. The value $U_k^T$ can be used as upper bound.

In the following we discuss some aspects related to the code generation based on the method proposed in this paper. Some possible optimizations are also mentioned.

The code is much simpler if step $h_{l_k}$ is 1. The statement:

```
  for $j_1 = L_1^*, U_1^*$ do
    if the transformation matrix is unimodular then its HNF is the identity matrix. Since the elements on the diagonal of the HNF are equal to 1, all the iteration control statements have the form:
      for $j_2 = L_2^*, U_2^*$ do
        Therefore, when transformation matrix is unimodular our method gives the same result as other methods previously proposed for unimodular transformations.
        When the step $h_{l_k}$ is greater than 1, the generated code can be optimized. Assume the following piece of code:
          for $j_2 = L_2^*, U_2^*, \text{step}=h_{l_k}$ do
            gap = $(h_{l_k}, j_1 + h_{l_k}, j_2)$ mod $h_{l_k}$
            for $j_3 = (L_3^* - \text{gap}) / h_{l_k} \times h_{l_k} + \text{gap}_3, U_3^*, \text{step}=h_{l_k}$ do
              $\ldots$
              $\ldots$
            endfor
          endfor
```

In every iteration of the $j_2$ loop, a new value of $\text{gap}_3$ is computed. However, only a small part of the expression to compute $\text{gap}_3$ depends on $j_2$. The rest of the expression is a constant. Therefore, the code can be optimized in the following way:

```
  for $j_2 = L_2^*, U_2^*, \text{step}=h_{l_k}$ do
    gap = $(h_{l_k}, h_{l_k}, j_1 + h_{l_k}, j_2)$ mod $h_{l_k}$
    for $j_3 = L_3^*, U_3^*, \text{step}=h_{l_k}$ do
      $\ldots$
      $\ldots$
    endfor
  endfor
```

The first value of $\text{gap}_3$ is computed before the first iteration of loop $j_2$ and updated at the end of every iteration of loop $j_2$. In general, the expression for update $\text{gap}_3$ is:

```
  $\text{gap}_3 = (\text{gap}_3 + h_{l_k}) \mod h_{l_k}$
```

Notice that the computation of the modulo operation is simple in this case since the values of $\text{gap}_3$ and $h_{l_k}$ are always less than $h_{l_k}$ (Definition 2 and Lemma 2). Therefore, $\text{gap}_3$ can be updated through an addition, a subtraction, a comparison and a conditional jump.

In an RISC architecture, conditional jumps can degrade the pipeline performance. However, these conditional jumps can be reordered in such way that performance penalties are eliminated (assuming a delay branch equal to 1). On the other hand in a processor such as the HP PA-RISC 7100 [7] there are instructions which are able to conditionally inhibit the writing of the result of the next instruction. This type of instructions can be used to implement this modulo operation, without a conditional jump, reducing in that way the number of instructions and the pipeline penalties, as shown in Fig. 4.

As a final example, we give now the transformed code for the example of Fig. 1, corresponding to the following transformation matrix $T$ and its HNF $H$ (the BTIS is shown in Fig. 2b):

```
  \begin{bmatrix}
    1 & 0 & 0 \\
    0 & 1 & 2
  \end{bmatrix}
```

```
  gap2 = 0
  for $j_1 = 0, 1$ do
    Lt = max(2*\lfloor j1/12 \rfloor, 2*\lfloor j1/1 \rfloor)
    Ut = min(18*\lfloor j1/12 \rfloor, 2*\lfloor j1 \rfloor)
    for $j_2 = \lfloor \text{max}(Lt - gap2)/3 \rfloor$ 3 + gap2, Ut, step=3 do
      $\ldots$
      $\ldots$
    endfor
  endfor
```

```
  R0 <- (R0 + h_{kk-1}) \mod h_{kk}
```

```
  (a) CMP R0, (h_{kk} - h_{kk-1})
  BLSS $1$
  ADD h_{kk-1}, R0
  SUB h_{kk}, R0

  $1$:
  ADD h_{kk-1}, R0
  SUB $0$, h_{kk}, R0
  $R0' \leftarrow R0 - h_{kk}$
  ADD h_{kk}, R0
  $\text{if } R0' < 0 \text{ then } R0 <- R0 + h_{kk}$
```

Fig. 4. Updating the value of gap3 (a) Code for a processor with delayed branch. (b) Code for a processor which instructions which are able to inhibit the writing of the result of the next instructions.
VI. CONCLUSIONS

In this paper we present a method to transform in a systematic way, a program expressed by a number of perfectly nested loops. The matrix which represents the transformation can be either unimodular or nonunimodular. If the transformation matrix is unimodular we obtain the same code as other authors. If the transformation matrix is nonunimodular, the exact bounds and steps for the loops are computed. Conditional statements are not required in the body of the loop.

When the transformation matrix is nonunimodular the BTIS is full of holes. To cover the BTIS, it is necessary to skip these holes and identify the functions which delimit the BTIS.

To characterize the BTIS we use the Hermite Normal Form $H$ of matrix. Both $H$ and $T$ generate the same lattice in $Z^n$. Since $H$ is lower triangular, it permits an easy characterization of the BTIS.

The holes of the BTIS are skipped using a step greater than 1 in the loops. These steps are just the elements on the diagonal of the HNF of $T$. The exact bounds of the BTIS are obtained using the functions which delimit the BTIS and a function $gap_k$ involving the elements of the HNF out of the diagonal. It has been shown how the computation of $gap_k$ can be optimized.

APPENDIX A

FOURIER-MOTZKIN ALGORITHM TO OBTAIN THE BOUNDS OF THE TIS

Let $A \cdot x \leq B$ be the matrix inequality which defines the TIS. The following iterative procedure bounds each of the control variables, from that corresponding to the innermost loop to that corresponding to the outermost loop. The steps of the procedure are:

1) Transform the matrix inequality in such a way that all the coefficients in the column of $A$ associated with the first control variable to be bounded have value 1, -1, or 0. To that purpose, each row of the matrix inequality is divided by the required value. At the end of this step, the matrix inequality can be decomposed into three matrix inequalities:

$$A_1 \cdot x \leq \beta_1 \quad A_2 \cdot x \leq \beta_2 \quad A_0 \cdot x \leq \beta_0$$

where matrices $A_1$, $A_2$, and $A_0$ have 1, -1, and 0, respectively in the columns associated with the control variable to be bounded.

2) The first two matrix inequalities are rewritten as follows:

$$A_1 \cdot x \leq \beta_1 \Rightarrow x_j \leq \beta_1 - (A'_1 \cdot x')$$

$$A_2 \cdot x \leq \beta_2 \Rightarrow (A'_2 \cdot x') - \beta_1 \leq x_j$$

where $x_j$ is the control variable to be bounded, $A'_1$ is matrix $A_1$ without the column associated with $x_j$, and $x'$ is vector $x$ without component $x_j$. Note that this transformation is possible since the coefficients of $A_1$ which multiply $x_j$ are 1. Analogously, $A'_2$ is defined.

3) The bounds for $x_j$ are:

$$\max (A'_1 \cdot x' - \beta_1) \leq x_j \leq \min (\beta_1 - (A'_1 \cdot x'))$$

(a) The bounds expressed in that way may take noninteger values. Therefore, we take the ceiling for the lower bound and the floor for the upper bound.

4) From (a) we build the following matrix inequation:

$$A'_2 \cdot x' - \beta_1 \leq \beta_1 - (A'_1 \cdot x')$$

(b) Steps 1) to 5) are repeated to obtain the bounds for the next control variable, until vector has a single component.

ACKNOWLEDGMENTS

This work has been supported by the Spanish Ministry of Education and Science (TIC-299/89 and TIC-880/92) and by the Centro Europeo de Parallelismo en Barcelona (CEPBA).

REFERENCES


Agustín Fernández is an assistant professor in the Computer Architecture Department of the Polytechnic University of Catalonia (UPC), in Barcelona, Spain. He received a computer science degree in 1988 and the PhD in computer science in 1992, both from the UPC. His primary research interest is in parallel architectures and algorithms for them and memory hierarchies. He is a researcher of the European Center for Parallelism of Barcelona (CEPBA).

José M. Llabería is a full professor in the Computer Architecture Department of the Polytechnic University of Catalonia (UPC), in Barcelona, Spain. He received the MS degree in telecommunication, the MS and the PhD degrees in computer science from the UPC in 1980, 1983, and 1983, respectively. His research interests are parallel processing, vector processors, superscalar processors, memory hierarchies, and compiler technology for these processors. He is a member of the IEEE and a researcher of the European Center for Parallelism of Barcelona (CEPBA).

Miguel Valero-García is an assistant professor in the Computer Architecture Department of the Polytechnic University of Catalonia (UPC), in Barcelona, Spain. He received a computer science degree in 1985 and the PhD in computer science in 1989, both from the UPC. Dr. Valero-García’s primary research interest is in parallel architectures and algorithms. He is a member of the IEEE and ACM and a researcher of the European Center for Parallelism of Barcelona (CEPBA).