Automating Non-Unimodular Loop Transformations for Massive Parallelism

Jingling Xue

School of Electrical and Electronic Engineering, Nanyang Technological University, Nanyang Avenue, Singapore 2263. Email: ejxue@ntu.edu.sg

Abstract. Loop transformations have been shown to be very useful in parallelising compilation and regular array design. This paper provides a solution to the open problem of automatic rewriting loop nests for non-unimodular loop transformations. We present an algorithm that rewrites a loop nest under any non-singular (unimodular or non-unimodular) transformation in a mechanical manner. The algorithm works nicely with unimodular transformations being treated as a special case. The extra time complexity incurred due to non-unimodularity is polynomially bounded by the depth of the loop nest.

Keywords. Loop transformation, parallelising compilation, regular array design, non-unimodular, Hermite normal form, lattice, Fourier-Motzkin method.

1 Introduction

Loop transformations, such as loop interchange, reversal, skewing and tiling, have been shown to be very useful in the presently two closely related areas: parallelising compilation [1, 3, 21, 22] and regular array design [6, 7, 15, 16, 19, 24]. The process of loop transformations can be divided into three steps. The first step is to gather useful knowledge about the underlying dependences of a program [2, 13, 22]. The second step is to choose the optimal loop transformations in such a way that the dependences of the program are respected and certain predefined goals are fulfilled. The third step is to rewrite a loop nest transformed by a loop transformation. It consists of rewriting the loop body, strides and bounds. Algorithms for the rewriting of loop nests exist. See [12, 20] for algorithms in regular array design and [1, 21] in parallelising compilation. However, these algorithms work only for unimodular loop transformations.

Why are we concerned with non-unimodular transformations? For a loop nest of the depth $n$, a loop transformation, $T$, is an $n \times n$ non-singular integer matrix:

$$T = \begin{bmatrix} S \\ A \end{bmatrix}$$

where $S \in \mathbb{Z}^{r \times n}$ ($0 < r < n$) is called the scheduling matrix, which maps the iterations of a loop nest to the temporal domain, and $A$ is called the allocation matrix, which maps the iterations of a loop nest to the spatial domain. A loop transformation is legal if it respects the dependences of the loop nest. The legality of a loop transformation is completely determined by the scheduling matrix. A loop transformation is legal if $SI > SJ$ whenever $x(I)$ depends on $x(J)$. Broadly speaking, there are two relatively independent goals in the choice of loop transformations: parallelism and hardware utilisation (e.g., minimising the processor count in regular array design and efficient use
of the memory hierarchy in parallelising compilation). We refer to [19, 23] for methods on the
minimisation of the latency and processor count of a regular array and to [10] for the maximisation
of parallelism in a loop nest. The first goal restricts the choice of the scheduling matrix. The second
goal restricts the choice of the allocation matrix. Therefore, the space of legal transformations may
sometimes contain non-unimodular transformations only.

The relevance of non-unimodular transformations to regular array design is well known. The
Kung-Leiserson’s array for band-matrix product is described by a non-unimodular transformation
(Ex. 3). The same transformation applied to LU-decomposition yields a two-dimensional array
in which the functionality of every processor is time-invariant [14]. The advantage is that control
signals [24], which would be required if differing transformations are used, are dispensed with. If the
iteration space is infinite, the allocation matrix must be chosen in such a way that the processor
count is finite [15]. If, in addition, the goal in the choice of scheduling vector is to minimise
the latency of the array, then the space of legal transformations may contain non-unimodular
transformations only. Finally, non-unimodular transformations can be used to advantage in the
synthesis of fixed-size arrays [7].

There have been some work on generating systolic code from loop nests for programmable
systolic arrays [4, 6, 17]. However, the loop transformations are restricted to be unimodular.
Recently, an attempt on relaxing this restriction was described in [5]. It was shown how to rewrite
one example double loop for one non-unimodular transformation. It is unclear whether and how
general loop nests can be dealt with.

In the context of parallelising compilation, loop transformations have been confined to be uni-
modular [1, 3, 22, 21]. Several primitive transformations such as loop interchange, reversal and
skewing are unimodular transformations. Recently, there have been some work on automatically
generating loop transformations to maximise parallelism inherent in a loop nest [10, 21]. The loop
transformations considered are unimodular. Essentially, one searches for an optimal scheduling
vector $S$ and then chooses the loop transformation matrix $T$ as any unimodular matrix whose first
row is $S$. Let $H$ be the Hermite normal form of $S$. Then there exists a unimodular matrix $C$ such
that $SC = H$. $T$ can be simply set to the inverse of $C$: $T = C^{-1}$. It makes no difference which
allocation matrix to choose if there exists unbounded number of processors. In this case, each
processor is assigned only one iteration from the processor space at any time step. All independent
iterations in the processor space are executed simultaneously at different processors. But it does
make a difference if only a finite number of processors are available. In this case, different loop
transformations lead to syntactically different loop nests, which can have different impacts on the
amount of data transfers required among the memory hierarchy and on the data locality in the
access of cache or memory [8, 9]. Finding the best loop transformation is a difficult problem. We
expect that inclusion of non-unimodular transformations into the space of legal transformations
can contribute to solving this problem.

In this paper, we present an algorithm that rewrites a loop nest under any non-singular (uni-
modal or non-unimodular) loop transformation in a mechanical manner. The first step of the
algorithm calculates the loop bounds using the Fourier-Motzkin elimination method that has been
previously promoted in the literature. The second step of the algorithm relies on a method based
on the theory of Hermite normal form and lattice and is only needed for non-unimodular transfor-
mations. It consists of calculating the loop strides and adjusting the loop bounds derived in the first step so that the “holes” in the image iteration space are skipped. The time complexity of the second step is polynomially bounded by the depth of the loop nest. The adjusted loop bounds is the simplest that can be expected.

The plan of the paper is as follows. Sect. 2 introduces the basic terminology and definitions used in the paper. Sect. 3 discusses the formulation of the problem. Sect. 4 gives a brief sketch of our loop rewriting algorithm. Sect. 5 explains briefly the Fourier-Motzkin elimination method through the rewriting of a loop nest for a unimodular transformation. Sect. 6 describes the details of the algorithm by focusing on the non-unimodular transformations of loop nests. Sect. 7 provides a geometrical interpretation of our loop rewriting algorithm. Sect. 8 demonstrates further the algorithm through the rewriting of two more example loop nests. Sect. 9 contains the concluding remarks of the paper.

2 Unimodularity, Lattice, Hermite Normal Form

We recall several concepts in linear algebra that are central to the paper [18].

**Definition 1** A square matrix is unimodular if it is integral and has determinant ±1.

The inverse of a unimodular matrix is still a unimodular matrix. The inverse of a non-unimodular matrix is not integral. This is why the concept of unimodularity comes in.

**Definition 2** Let $A$ be an $m \times n$ integer matrix. The set $\mathcal{L}(A) = \{ y \mid y = Ax \land x \in \mathbb{Z}^n \}$ is called the lattice generated by the columns of $A$.

**Theorem 1** If $A$ is an $m \times n$ integer matrix and $C$ is an $n \times n$ unimodular matrix, then $\mathcal{L}(AC) = \mathcal{L}(A)$.

If $A$ is a non-singular square matrix, the columns are called a basis of the lattice $\mathcal{L}(A)$. The columns of $A$ generate the lattice $\mathbb{Z}^n$ iff $A$ is a unimodular matrix.

**Definition 3** A non-singular square integer matrix is said to be in Hermite normal form if it is lower triangular, nonnegative matrix, in which each row has a unique maximum entry, which is located on its main diagonal.

**Theorem 2** If $A$ is an $m \times n$ integer matrix of full row rank, then there exists an $n \times n$ unimodular matrix $C$ such that $AC = [H \ 0]$ and $H$ is in Hermite normal form.

By Thm. 1, we conclude that $\mathcal{L}(A) = \mathcal{L}(H)$. If the greatest common divisor of the determinants of all the $m \times m$ submatrices of $A$ is 1, then $H$ is the identity matrix. Every integer matrix of full row rank has a unique Hermite normal form.
3 Problem Statement

As is customary, we write \([x]\) for the ceiling of \(x\), i.e., the smallest integer not smaller than \(x\), and we write \([x]\) for the floor of \(x\), i.e., the greatest integer not greater than \(x\). The operator \(\%\) denotes the modulo operation. For \(x, y \in \mathbb{Z}^n\), \(x \% y = r\) iff \(x = qy + r\), where \(q, r \in \mathbb{Z}\) and \(0 \leq r < |y|\). (Note that \(x \% y\) is always a non-negative integer smaller than \(|y|\).)

We represent a loop nest of depth \(n\) in the following format:

\[
\text{for } l_i \text{ from } \max([L_{i,1}], [L_{i,2}], \cdots) \text{ to } \min([U_{i,1}], [U_{i,2}], \cdots) \text{ by } \text{step}_i \\
\vdots \\
\text{for } l_n \text{ from } \max([L_{n,1}], [L_{n,2}], \cdots) \text{ to } \min([U_{n,1}], [U_{n,2}], \cdots) \text{ by } \text{step}_n
\]

(1)

The lower bounds \(L_{k,i}\) and upper bounds \(U_{k,j}\) are of the form:

\[
L_{k,i} = \left( \frac{\ell^0_{k,i} + \ell^1_{k,i} I_1 + \cdots + \ell^{k-1}_{k,i} I_{k-1}}{\ell^k_{k,i}} \right)
\]

\[
U_{k,j} = \left( \frac{u^0_{k,j} + u^1_{k,j} I_1 + \cdots + u^{k-1}_{k,j} I_{k-1}}{u^k_{k,j}} \right)
\]

(2)

where \(\ell^p_{k,i}\) and \(u^q_{k,j}\) are invariants in the loop nest. \(L_{k,i}\) and \(U_{k,j}\) may evaluate to non-integral values. This explains the necessity of the ceiling and floor functions in the loop bounds. The stride of loop variable \(I_k\) is \(\text{step}_k\). The statements inside the loop nest are omitted. Their rewriting after a transformation is straightforward [21]. A loop nest is said to be normalised if all its loops have stride 1. By convention, we identify each iteration in the loop nest by a point or an index vector \((I_1, \cdots, I_n)\). We write \(\Phi\) for the iteration space, i.e., the set of all iterations of the loop nest. If the loop nest is normalised, the iteration space \(\Phi\) is the (bounded) convex polyhedron in \(\mathbb{Z}^n\) defined by the loop bounds:

\[
\max(L_{1,1}, L_{1,2}, \cdots) \leq I_1 \leq \min(U_{1,1}, U_{1,2}, \cdots) \\
\vdots \\
\max(L_{n,1}, L_{n,2}, \cdots) \leq I_n \leq \min(U_{n,1}, U_{n,2}, \cdots)
\]

(3)

A system of linear constraints that are in the form of (2) and (3) is called a triangular system [1]. Unless triangular, a system of linear constraints does not directly contribute to the calculation of the loop bounds of the respective loop nest.

In a sequential loop nest, the iterations in the iteration space \(\Phi\) are executed in the lexicographical order of their index vectors. Let \(\ll\) be the lexicographical order:

\[
\ll = \{(I, J) \mid I < J \land I, J \in \Phi\}
\]

A loop transformation \(T\) changes the lexicographical order \(\ll\) to:

\[
\ll_T = \{(TI, TJ) \mid TI < TJ \land I, J \in \Phi\}
\]

The problem concerning us in this paper is the following. Given a normalised loop nest of depth \(n\) and a non-singular loop transformation \(T\), find an algorithm that rewrites the loop nest into another loop nest of depth \(n\) that has the lexicographical order \(\ll_T\).

The loop nest returned by the algorithm is called the transformed loop nest. We say that the transformed loop nest is correct if it has the lexicographical order \(\ll_T\).
4 Algorithm Sketch

The loop nest input to our loop rewriting algorithm is normalised; it has the form (1) where \( s = 1 \). A loop transformation \( T \) maps an iteration \((I_1, \cdots, I_n)\) to an iteration \((I'_1, \cdots, I'_n)\):

\[
\left[ \begin{array}{c} I'_1 \\ \vdots \\ I'_n \\ \end{array} \right] = T \left[ \begin{array}{c} I_1 \\ \vdots \\ I_n \\ \end{array} \right], \text{ i.e. } \left[ \begin{array}{c} I'_1 \\ \vdots \\ I'_n \\ \end{array} \right] = T^{-1} \left[ \begin{array}{c} I_1 \\ \vdots \\ I_n \\ \end{array} \right]
\]

(4)

The transformed loop nest returned by our loop rewriting algorithm has the form:

\[
\begin{align*}
\text{for } I'_1 \text{ from } & \max([L'_{1,1} + \delta'_{1,1}, L'_{1,2} + \delta'_{1,2}, \cdots]) \text{ to } \min([U'_{1,1}, U'_{1,2}, \cdots]) \text{ by } s'_{1,1} \\
& \cdots \\
\text{for } I'_n \text{ from } & \max([L'_{n,1} + \delta'_{n,1}, L'_{n,2} + \delta'_{n,2}, \cdots]) \text{ to } \min([U'_{n,1}, U'_{n,2}, \cdots]) \text{ by } s'_{n,1}
\end{align*}
\]

(5)

For convenience, we still refer to \( L'_{k,i} \) and \( U'_{k,j} \) as the lower and upper bounds, respectively. We refer to \( \delta'_{k,i} \) as the lower bound offsets. The algorithm consists of two parts:

1. Calculate the loop bounds \( L'_{k,i} \) and \( U'_{k,j} \) using the Fourier-Motzkin elimination method.
2. Calculate the loop strides \( s'_{k,i} \) and the lower bound offsets \( \delta'_{k,i} \) using a method based on the theory of Hermite normal form and lattice.

In the rational space, a loop transformation maps a bounded convex polyhedron to another bounded convex polyhedron. We refer to the image of the iteration space \( \Phi \) under a loop transformation in the rational space as the image iteration space and denote it by \( \Phi' \). The image iteration space \( \Phi' \) is obtained by substituting the solutions of \( I_1, \cdots, I_n \) in (4) into (3).

To calculate \( L'_{k,i} \) and \( U'_{k,j} \) is to rewrite the image iteration space \( \Phi' \) into a triangular system. To accomplish this, Ancourt and Irigion proposed to use the Fourier-Motzkin method. See [18] for details about the Fourier-Motzkin method and [1] about Ancourt and Irigion’s algorithmic implementation. The basic idea of the Fourier-Motzkin method is given below.

The Fourier-Motzkin method works by a successive projection of the linear system defining \( \Phi' \) along \( L'_1, \cdots, L'_k \) in that order. In the \( k \)-th projection (the starting point is the case \( k = 1 \)), we have as input a system of linear constraints containing \( I'_1, \cdots, I'_{n-k+1} \) only. By projecting the system along \( I'_{n-k+1} \), we obtain as output the lower and upper bounds of \( I'_{n-k+1} \):

\[
\max(L'_{1,1}, L'_{1,2}, \cdots) \leq I'_{n-k+1} \leq \min(U'_{1,1}, U'_{1,2}, \cdots)
\]

where \( L'_{k,i} \) and \( U'_{k,j} \) conform to the format of (2) and a system of linear constraints containing \( I'_1, \cdots, I'_{n-k} \) only, which is the input in the \((k+1)\)-st projection. Thus, \( I'_{n-k+1} \) has been eliminated from the system prior to the \( k \)-th projection. After the \((n-1)\)-st, i.e., the last projection along \( L'_2 \), we end up with a system of linear constraints containing only \( I'_1 \) from which the lower and upper bounds of \( I'_1 \) can be trivially calculated.

The image iteration space \( \Phi' \) is now defined by the following triangular system:

\[
\begin{align*}
\max(L'_{1,1}, L'_{1,2}, \cdots) & \leq I'_1 \leq \min(U'_{1,1}, U'_{1,2}, \cdots) \\
& \cdots \\
\max(L'_{n,1}, L'_{n,2}, \cdots) & \leq I'_n \leq \min(U'_{n,1}, U'_{n,2}, \cdots)
\end{align*}
\]
where \( L'_{k,i} \) and \( U'_{k,j} \) are the solutions so desired in the first step of our loop rewriting algorithm.

The second step of the algorithm is to calculate the loop strides and lower bound offsets. We use the following notations. From now on, whenever we speak of lattice points we mean the lattice points in \( L(T) \). For convenience, we refer to the integer points in \( \mathbb{Z}^n \setminus L(T) \) as the non-lattice points. When we say \((I'_1, \ldots, I'_k, *)\) is a lattice point with its last \( n-k \) components unspecified, we mean that there are \( X'_1, \ldots, X'_n \) in \( \mathbb{Z}^n \) such that \((I'_1, \ldots, I'_k, X'_{k+1}, \ldots, X'_n)\) is a lattice point. In other words, if \((I'_1, \ldots, I'_k, *)\) is a lattice point, then \((I'_1, \ldots, I'_k)\) belongs to the projection of \( L(T) \) onto \( I'_1, \ldots, I'_k \).

If the loop transformation \( T \) is unimodular, then \( L(T) = \mathbb{Z}^n \). All the integer points in the image iteration space are lattice points. The transformed loop nest is correct if we set \( step'_k = 1 \) and \( \delta'_{k,i} = 0 \). If the loop transformation \( T \) is non-unimodular, then \( L(T) \neq \mathbb{Z}^n \). Some integer points in the image iteration space are not lattice points. These non-lattice points are what have been conventionally called holes. Their inverse images under \( T \) are non-integral. If we still set \( step'_k = 1 \) and \( \delta'_{k,i} = 0 \), the transformed loop nest is incorrect. This is because the holes in the image iteration space are treated as iterations in the transformed loop nest, although they should not be.

Let \((I'_1, \ldots, I'_k, *)\) be a lattice point. The loop stride \( step'_k \) of \( I'_k \) is simply the positive integer such that \((I'_1, \ldots, I'_k + step'_k, *)\) is the smallest (in the lexicographical sense) lattice point larger than \((I'_1, \ldots, I'_k, *)\). Since \( L(T) \) is a lattice, the existence of \( step'_k \) is guaranteed. To specify the lower bound offset \( \delta'_{k,i} \) for the lower bound \( L'_{k,i} \), we consider the set of lattice points \((I'_1, \ldots, I'_{k-1}, *)\). If \((I'_1, \ldots, I'_{k-1}, [L'_{k,i}], *)\) is a lattice point whenever \((I'_1, \ldots, I'_{k-1}, *)\) is, we choose \( \delta'_{k,i} = 0 \). In general, we choose \( \delta'_{k,i} \), which is an expression of \( I'_1, \ldots, I'_{k-1} \), such that \((I'_1, \ldots, I'_{k-1}, [L'_{k,i}] + \delta'_{k,i}, *)\) is the smallest lattice point not smaller than \((I'_1, \ldots, I'_{k-1}, [L'_{k,i}], *)\). With the loop strides and the lower bound offsets so calculated, the iteration space of the transformed loop nest is just the set of the lattice points in the image iteration space. The correctness of the transformed loop nest is not difficult to see.

## 5 Unimodular Transformations

This section discusses the rewriting of loop nests for unimodular transformations. If the loop transformation is unimodular, we can set the loop strides to 1 and the lower bound offsets to 0. All it remains to be done is to calculate \( L'_{k,i} \) and \( U'_{k,j} \) using the Fourier-Motzkin method.

**Example 1** Consider the double loop:

\[
\text{for } I_1 \text{ from } 1 \text{ to } n \text{ by } 1 \\
\text{for } I_2 \text{ from } 1 \text{ to } n \text{ by } 1
\]

and the following unimodular transformation:

\[
\begin{bmatrix}
I'_1 \\
I'_2
\end{bmatrix} = \begin{bmatrix}
3 & 2 \\
1 & 1
\end{bmatrix} \begin{bmatrix}
I_1 \\
I_2
\end{bmatrix}, \ \text{i.e.} \ \begin{bmatrix}
I_1 \\
I_2
\end{bmatrix} = \begin{bmatrix}
1 & -2 \\
1 & 3
\end{bmatrix} \begin{bmatrix}
I'_1 \\
I'_2
\end{bmatrix}
\]

(6)

Fig. 1(a) depicts the iteration space \( \Phi \) of the loop nest, which is defined as follows:

\[
1 \leq I_1 \leq n \\
1 \leq I_2 \leq n
\]

(7)
Fig. 1(b) depicts the image iteration space $\Phi'$, which is obtained by substituting the solutions of $I_1$ and $I_2$ in (6) into (7) and is defined as follows:

\[
\begin{align*}
1 & \leq I_1' - 2I_2' \leq n \\
1 & \leq -I_1' + 3I_2' \leq n
\end{align*}
\] (8)

This system of linear constraints is not triangular. This is so because the loop bounds of $I_1'$ are not readily available. We then use the Fourier-Motzkin method to rewrite this system into a triangular system. The system (8) is equivalent to:

\[
\begin{align*}
(I_1' - n)/2 & \leq I_2' \leq (I_1' - 1)/2 \\
(I_1' + 1)/3 & \leq I_2' \leq (I_1' + n)/3
\end{align*}
\]

i.e. to:

\[
\max((I_1' - n)/2, (I_1' + 1)/3) \leq I_2' \leq \min((I_1' - 1)/2, (I_1' + n)/3)
\] (9)

This gives rise to the lower and upper bounds of $I_2'$. Eliminating $I_2'$ from (9), we obtain a system of constraints containing $I_1'$ only:

\[
\begin{align*}
(I_1' - n)/2 & \leq (I_1' - 1)/2 \\
(I_1' - n)/2 & \leq (I_1' + n)/3 \\
(I_1' + 1)/3 & \leq (I_1' - 1)/2 \\
(I_1' + 1)/3 & \leq (I_1' + n)/3
\end{align*}
\]

which simplifies to:

\[5 \leq I_1' \leq 5n\] (10)

This yields the lower and upper bounds of $I_1'$. The image iteration space which was defined before by the system (8) has been rewritten to the triangular system of linear constraints in (9) and (10). Since the loop transformation $T$ is unimodular, we choose

\[step_1 = 1, \ step_2 = 1, \ \delta_{1,1}' = 0, \ \delta_{2,1}' = 0, \ \delta_{2,2}' = 0\]

The transformed loop nest becomes:

\[
\begin{align*}
\text{for } I_1' & \text{ from } 5 \text{ to } 5n \text{ by } 1 \\
\text{for } I_2' & \text{ from } \max([(I_1' - n)/2], [(I_1' + 1)/3]) \text{ to } \min([(I_1' - 1)/2], [(I_1' + n)/3]) \text{ by } 1
\end{align*}
\]
6 Non-Unimodular Transformations

If the loop transformation is non-unimodular, an automatic method for the calculation of the loop strides $\text{step}^l_i$ and lower bound offsets $\delta^l_{k,i}$ is called for. To facilitate the development of our method, we consider a motivating example.

Example 2 Consider again the double loop in Ex. 1. This time, we choose a non-unimodular transformation:

$$
\begin{bmatrix}
I'_1 \\
I'_2
\end{bmatrix} = \begin{bmatrix} 3 & 2 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} I_1 \\
I_2
\end{bmatrix}, \text{ i.e. } \begin{bmatrix} I'_1 \\
I'_2
\end{bmatrix} = \begin{bmatrix} 0 & 2/2 \\ 1/2 & -3/2 \end{bmatrix} \begin{bmatrix} I_1 \\
I_2
\end{bmatrix}
$$

Applying the Fourier-Motzkin method, we obtain the transformed loop nest as follows:

\begin{align}
\text{for } I'_1 \text{ from } 5 + \delta^l_{1,1} & \text{ to } 5n \text{ by } \text{step}^l_1 \\
\text{for } I'_2 \text{ from } \max(1 + \delta^l_{2,1}, [(I'_1 - 2n)/3] + \delta^l_{2,2}) & \text{ to } \min(n, [(I'_1 - 2)/3]) \text{ by } \text{step}^l_2
\end{align}

(11)

Fig. 2 depicts the corresponding image iteration space $\Phi'$. The lattice points inside are depicted by heavy dots and the non-lattice points inside are not highlighted.

The loop strides $\text{step}^l_1$ and $\text{step}^l_2$ and the lower bound offsets $\delta^l_{1,1}$, $\delta^l_{2,1}$ and $\delta^l_{2,2}$ in the transformed loop nest (11) remain to be determined. If we simply set

$$\text{step}^l_1 = 1, \text{ step}^l_2 = 1, \delta^l_{1,1} = 0, \delta^l_{2,1} = 0, \delta^l_{2,2} = 0$$

The transformed loop nest thus obtained is incorrect because of the presence of the non-lattice points in $\Phi'$. In fact, our algorithm returns the following loop strides and lower bound offsets:

$$\text{step}^l_1 = 1, \text{ step}^l_2 = 2, \delta^l_{1,1} = 0, \delta^l_{2,1} = (I'_1 - 1)\%2, \delta^l_{2,2} = (I'_1 - [(I'_1 - 2n)/3])\%2$$

To see why the transformed loop nest thus obtained is correct, let us consider the image iteration space depicted in Fig. 2. The loop stride $\text{step}^l_1$ is set to 1, because if $(I'_1, I'_2) \in \mathcal{L}(T)$, then $(I'_1 + 1, \ast) \in \mathcal{L}(T)$. The loop stride $\text{step}^l_2$ is set to 2, because if $(I'_1, I'_2) \in \mathcal{L}(T)$ then $(I'_1, I'_2 + 1) \not\in \mathcal{L}(T)$ and $(I'_1, I'_2 + 2) \in \mathcal{L}(T)$. It is easy to understand why $\delta^l_{1,1} = 0$. To see why $\delta^l_{2,1} = (I'_1 - 1)\%2$, we consider the bottom boundary of the image iteration space, i.e., $L^l_{2,1} = 1 = I'_2$. $L^l_{2,2} = 1$ is a constant. It represents the second component of the integer points closest to the boundary. (These integer points are on the boundary.) It is easy to see that some of these integer points are non-lattice points. This is where the lower bound offset $\delta^l_{2,1}$ comes into play. The lower bound offset $\delta^l_{2,1} = (I'_1 - 1)\%2$ evaluates to 0 and 1 alternatively starting from the first iteration $I'_1 = 5$. Thus, $[L^l_{2,1}] + \delta^l_{2,1}$ evaluates to 1 and 2 alternatively, giving rise to the second components of the lattice points closest to the boundary, as intended. The analysis for $\delta^l_{2,2}$ is similar and is therefore omitted.
We are now ready to describe our method for the derivation of the loop strides $step'_k$ and the lower bound offsets $\delta'_{k,i}$. In fact, all information necessary for the derivation is contained explicitly in the loop transformation $T$. To make this information explicit, all we need to do is to calculate the Hermite normal form, denoted $\Delta$, of $T$. Thus, there exists an $n \times n$ unimodular matrix $C$ such that

$$\Delta = TC$$

By Thm. 1, $L(T) = L(\Delta)$. That is, the columns of $T$ generate the same lattice as those of $\Delta$. From $\Delta$, the loop strides $step'_k$ and the lower bound offsets $\delta'_{k,i}$ can be readily read off.

We shall use the following notations. $\Delta_{i,j}$ denotes the element of $\Delta$ in its $i$-th row and $j$-column. $\Delta_k$ denotes the $k$-th column of $\Delta$.

### 6.1 The Derivation of Loop Strides

To calculate the loop stride of $I'_k$, we only need to pay attention to a subset of lattice points in $L(T)$ whose first $k-1$ components are identical. In addition, it suffices to consider only the set of $k$-th components of all lattice points in the subset. It is for this reason that we introduce the following notation:

$$L_k(T) = \{ I'_k | (0, \cdots, 0, I'_k, \cdots, I'_n) \in L(T) \}$$

The following lemma provides the basis for the calculation of the loop strides. It asserts that $L_k(T)$ is a one-dimensional lattice, which is generated by the $1 \times 1$ matrix $[\Delta_{k,k}]$.

**Lemma 1** $L_k(T) = L([\Delta_{k,k}])$.

**Proof.** By the definition of $L_k(T)$, $0 \in L_k(T)$. It suffices to show that (1) $\Delta_{k,k} \in L_k(T)$ and (2) $x \notin L_k(T)$, for $0 < x < \Delta_{k,k}$. The columns of $\Delta$ belong to $L(T)$. The first $k-1$ components of the $k$-th column of $\Delta$ is 0 and the $k$-th component is $\Delta_{k,k}$. Hence, $\Delta_{k,k} \in L_k(T)$. This proves (1). Let $(0, \cdots, 0, x, \ast)$ be in $\mathbb{Z}^n$ such that its first $k-1$ components are 0 and the $k$-th component $x$ satisfies $0 < x < \Delta_{k,k}$. To prove (2) is to prove $(0, \cdots, 0, x, \ast) \notin L(T)$. $\Delta$ is a lower triangular matrix. If $(0, \cdots, 0, x, \ast) \in L(T)$, $x$ must be an integral multiple of $\Delta_{k,k}$. A contradiction. \[\square\]

**Theorem 3** The loop strides of loop variables $I'_k$ ($0 < k \leq n$) are: $step'_k = \Delta_{k,k}$.

**Proof.** Following directly from Lemma 1. \[\square\]

Let us use this theorem to derive the loop strides in Ex. 2. The loop transformation $T$ is as in (11). Its Hermite normal form $\Delta$ is:

$$\Delta = \begin{bmatrix} 1 & 0 \\ 1 & 2 \end{bmatrix} = \begin{bmatrix} 3 & 2 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ -1 & -3 \end{bmatrix} = TC$$

(12)

Hence, $step'_1 = 1$ and $step'_2 = 2$ are the loop strides so desired.

The loop strides $step'_k$ are unique, since the Hermite normal form $\Delta$ is.
6.2 The Derivation of Lower Bounds Offsets

Just like the calculation of the loop strides, the calculation of the lower bound offsets can be carried out based on $\Delta$ alone. Let there be a lattice point $(I'_1, \cdots, I'_{k-1}, * ) \in \mathcal{L}(\Delta)$. In the column basis of $\Delta$, $(I'_1, \cdots, I'_{k-1}, [L'_{k,i}], *)$ becomes $(X_{1,i}', \cdots, X_{k,i}', *)$, where $X_{1,i}', \cdots, X_{k,i}'$ are the solutions to the following equation:

$$
\Delta \begin{bmatrix}
X_{1,i}' \\
\vdots \\
X_{k,i}' \\
*
\end{bmatrix} = \begin{bmatrix}
I'_1 \\
\vdots \\
I'_{k-1} \\
[L'_{k,i}] \\
*
\end{bmatrix}
$$

(13)

Since $(I'_1, \cdots, I'_{k-1}, *) \in \mathcal{L}(\Delta)$, $X_{1,i}', \cdots, X_{k-1,i}'$ are integers. Since $\Delta$ is a lower triangular matrix, $X_{i,k}'$ does not depend on the last $(n-k)$ components of $(I'_1, \cdots, I'_{k-1}, [L'_{k,i}], *)$. If $X_{i,k}'$ is an integer whenever $(I'_1, \cdots, I'_{k-1}, *)$ is a lattice point. Then, $(I'_1, \cdots, I'_{k-1}, [L'_{k,i}], *)$ is a lattice point whenever $(I'_1, \cdots, I'_{k-1}, *)$ is. In this case, we choose $\delta_{i,k}' = 0$. In general, $(I'_1, \cdots, I'_{k-1}, [L'_{k,i}], *)$ can be a hole for some lattice point $(I'_1, \cdots, I'_{k-1}, * )$. We choose $\delta_{i,k}'$ as follows. The smallest lattice point not smaller than $(I'_1, \cdots, I'_{k-1}, [L'_{k,i}], *)$ is $(I'_1, \cdots, I'_{k-1}, [L'_{k,i}] + \delta_{i,k}', *)$, where

$$
\delta_{i,k}' = (\lfloor L'_{k,i} \rfloor \% \Delta_{k,k} \Delta_{1,1})
$$

(14)

By this formula, $\delta_{i,k}' = \lfloor L'_{k,i} \rfloor \% \Delta_{1,1}$. $\Delta_{1,1}$ is the greatest common divisor of the elements in the first row of $T$. $L'_{k,i}$ is always an integral multiple of $\Delta_{1,1}$. This implies that $\delta_{i,k}' \equiv 0$.

**Theorem 4** The transformed loop nest with the loop strides $step'_k$ as in Thm. 3 and the lower bound offsets $\delta_{i,k}'$ as in (14) is correct.

**Proof.** The transformed loop nest is correct if it has the lexicographical order $\ll_F$. It suffices to show that the iteration space of the transformed loop nest is the set of all lattice points in the image iteration space. This follows from the following two facts. (1) With the adjusted lower bounds, every loop variable $I'_k$ starts to iterate at a lattice point $(I'_1, \cdots, I'_{k-1}, \max([L'_{k,1}], \delta_{1,k}', \cdots, \delta_{k,1}')]$, which is the smallest lattice point not smaller than $(I'_1, \cdots, I'_{k-1}, \max([L'_{k,1}], [L'_{k,2}], \cdots), *)$, and (2) all lattice points whose first $k-1$ components are $I'_1, \cdots, I'_{k-1}$ have the form $(I'_1, \cdots, I'_{k-1}, \max([L'_{k,1}], \delta_{1,k}', \cdots, \delta_{k,1}')] + step'_{k,1} + \delta_{k,2}' + \cdots + step'_{k,k} + \delta_{k-1}' + \cdots + step'_{1,1} + \delta_0')$ (Thm. 3).

Let us calculate the lower bound offsets in Ex. 2. With the $\Delta$ as in (12), a simple algebraic calculation yields:

$$
\begin{align*}
\delta_{1,1}' &= \lfloor L'_{1,1} \rfloor \% \Delta_{1,1} = 5 \% 1 = 0 \\
\delta_{2,1}' &= (I'_1 - \lfloor L'_{1,1} \rfloor) \% \Delta_{2,2} = (I'_1 - 1) \% 2 \\
\delta_{2,2}' &= (I'_1 - \lfloor L'_{1,2} \rfloor) \% \Delta_{2,2} = (I'_1 - \lfloor (I'_1 - 2n)/3 \rfloor) \% 2
\end{align*}
$$

6.3 The Loop Rewriting Algorithm

Our algorithm for the rewriting of loop nests is given below.
**Algorithm** loop Rewrite

**Input:** a loop transformation $T$, a loop nest in the form of (1) where $\text{step}_k = 1$.

**Output:** a loop nest in the form of (5) whose lexicographical order is $\ll T$.

- **Step 1.** Calculate the loop bounds $L'^i_{k,i}$ and $U'^i_{k,i}$ using the Fourier-Motzkin method.
- **Step 2.** Calculate the Hermite normal form $\Delta$ of $T$.
- **Step 3.** Set $\text{step}'_k = \Delta_{k,k}$.
- **Step 4.** Set $\delta'_{k,i} = (-X'^i_{k,i} - \Delta_{k,k}) \% \Delta_{k,k}$, where $X'^i_{k,i}$ is the solution of (13).

The time complexity is dominated by Step 1, which is not polynomial [18]. This step may also introduce redundant lower and upper bounds $L'^i_{k,i}$ and $U'^i_{k,i}$. Ancourt and Irijion’s algorithm uses the Fourier-Motzkin feasibility test to remove some redundant loop bounds [1]. The time taken by the remaining steps is dominated by Step 2, which is polynomially bounded by $n$, the depth of the loop nest.

The ceiling and floor functions are indispensable in the loop bounds $L'^i_{k,i}$ and $U'^i_{k,i}$ for unimodular or non-unimodular transformations, because some loop bounds may evaluate to non-integral values. Similarly, the modulo operation is unavoidable in the lower bound offsets, due to the need of bypassing the non-lattice points in the image iteration space. It is in this sense that the lower bound offsets derived in this paper are the simplest that can be expected.

This algorithm works nicely with unimodular transformations being treated as a special case. If the loop transformation $T$ is unimodular, then $\Delta$ is the identity matrix. Therefore, the loop strides $\text{step}'_k$ are 1 and lower bound offsets $\delta'_{k,i}$ are 0. This provides the justification of why we set $\text{step}'_k = 1$ and $\delta'_{k,i} = 0$ for unimodular transformations (Sect. 5).

### 7 A Geometrical Interpretation of the Algorithm

This section provides a geometrical insight into the development of our loop rewriting algorithm. The insight also sheds the light on how the transformed loop nest works.

Let $O$ be a lattice point in $\mathcal{L}(T)$. Let $P(O)$ be the unit parallelepiped with $O$ being one of its vertices and the columns of $\Delta$ being the edge vectors:

$$P(O) = \{O + x_1 \Delta_1 + \cdots + x_n \Delta_n \mid (\forall k : 0 < k \leq n : 0 \leq x_k < 1)\}$$

Clearly, the number of integer points contained in $P(O)$ is equal to $\det(T)$. The following lemma provides the basis of our geometrical interpretation of the loop rewriting algorithm.

**Theorem 5** $P(O)$ contains only one lattice point, which is $O$.

**Proof.** Let $I$ be an integer point in $P(O)$. If $I$ is a lattice point, then $I - O$ must be an integral combination of the columns of $\Delta$. That is, $I - O = X_1 \Delta_1 + \cdots + X_n \Delta_n$. Since $\Delta$ is a lower triangular matrix, we have $|I_1 - O_1| < \Delta_{1,1}$. Thus, $X_1 = 0$. This implies that $I_1 = O_1$ and consequently that $|I_2 - O_2| < \Delta_{2,2}$. By repeatedly applying the same line of reasoning, we obtain $X_2 = 0, \cdots, X_n = 0$. Hence, $I = O$. That is, $O$ is the only lattice point contained in $P(O)$.

\[\square\]
The image iteration space can be considered as being tiled by unit parallelepipeds. Fig. 3 depicts the tiling of the image iteration space of Ex. 2. This time, the non-lattice points inside are highlighted by circles. Since \( \det(T) = 2 \), each tile contains two integer points: one is a lattice point located at the bottom-right corner and the other is a hole located at the left edge.

If the loop transformation is unimodular, each tile contains only one integer point, which is a lattice point. In this case, the transformed loop nest scans all the integer points in the image iteration space. If the loop transformation is non-unimodular, each tile contains \( \det(T) \) integer points of which only one is a lattice point. The transformed loop nest scans only the lattice point in a tile. In both cases, the transformed loop nest can be regarded as scanning the tiles in the image iteration space. This way, only the lattice point inside a tile is visited. The remaining holes are successfully skipped.

8 Two Examples

In this section, we rewrite two example loop nests by going through the steps of the algorithm loop_rewrite, and at the end of each example, we examine in detail the transformed loop nest.

Example 3 Consider the triple loop:

\[
\begin{align*}
&\text{for } I_1 \text{ from } 1 \text{ to } n \text{ by } 1 \\
&\text{for } I_2 \text{ from } 1 \text{ to } n \text{ by } 1 \\
&\text{for } I_3 \text{ from } 1 \text{ to } n \text{ by } 1
\end{align*}
\]

The following non-unimodular transformation specifies Kung-Leiserson’s two-dimensional systolic array for matrix product [11]. This array is particularly appealing for band-matrix product, because its size is dependent on the size of the band rather than the size of the matrix.

\[
\begin{bmatrix}
I_1' \\
I_2' \\
I_3'
\end{bmatrix} =
\begin{bmatrix}
1 & 1 & 1 \\
1 & 0 & -1 \\
0 & 1 & -1
\end{bmatrix}
\begin{bmatrix}
I_1 \\
I_2 \\
I_3
\end{bmatrix}, \text{ i.e.}
\begin{bmatrix}
I_1 \\
I_2 \\
I_3
\end{bmatrix} =
\begin{bmatrix}
1/3 & 2/3 & -1/3 \\
1/3 & -1/3 & 2/3 \\
1/3 & -1/3 & -1/3
\end{bmatrix}
\begin{bmatrix}
I_1' \\
I_2' \\
I_3'
\end{bmatrix}
\]

An application of the Fourier-Motzkin method yields:

\[
\begin{align*}
L_{I_{1,1}} &= 3 \\
U_{I_{1,1}} &= 3n \\
\text{max}([L'_{2,1}], [L'_{2,2}], [L'_{2,3}]) &= \text{max}(1-n, -I_{I_{1}}+3, I_{I_{1}}-3n) \\
\text{min}([U'_{2,1}], [U'_{2,2}], [U'_{2,3}]) &= \text{min}(n-1, I_{I_{1}}-3, -I_{I_{1}}+3n) \\
\text{max}([L'_{3,1}], [L'_{3,2}], [L'_{3,3}]) &= \text{max}(I_{I_{1}}+2I_{I_{2}}-3n, [(-I_{I_{1}}+I_{I_{2}}+3)/2], I_{I_{1}}-I_{I_{2}}-3n) \\
\text{min}([U'_{3,1}], [U'_{3,2}], [U'_{3,3}]) &= \text{min}(I_{I_{1}}+2I_{I_{2}}-3, [(-I_{I_{1}}+I_{I_{2}}+3n)/2], I_{I_{1}}-I_{I_{2}}-3)
\end{align*}
\]
Following Step 2, we obtain

\[
\Delta = \begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
1 & 2 & 3
\end{bmatrix} \tag{16}
\]

By Step 3, the loop strides are the diagonal elements of \( \Delta \):

\[
\text{step}_1 = 1, \quad \text{step}_2 = 1, \quad \text{step}_3 = 3
\]

The calculation of the lower bound offsets follows Step 4. Trivially, \( \delta_{1,1} = \delta_{2,2} = \delta_{3,3} = 0 \), because \( \Delta_{1,1} = \Delta_{2,2} = 1 \). The lower bound offsets for loop variable \( I_3 \) are:

\[
\begin{align*}
\delta_{3,1} &= (I'_1 + 2I'_2 - (I'_1 + 2I'_2 - 3n)) \% 3 = 0 \\
\delta_{3,2} &= (I'_1 + 2I'_2 - [(-I'_1 + I'_2 + 3) / 2]) \% 3 \\
\delta_{3,3} &= (I'_1 + 2I'_2 - (I'_1 - I'_2 - 3n)) \% 3 = 0
\end{align*}
\]

The transformed loop nest is:

\[
\begin{align*}
\text{for } I'_1 \text{ from } 3 & \text{ to } 3n \text{ by } 1 \\
\text{for } I'_2 \text{ from } \max(1-n, -I'_1 + 3, I'_1 - 3n) & \text{ to } \min(n-1, -I'_1 - 3, -I'_1 + 3n) \text{ by } 1 \\
\text{for } I'_3 \text{ from } \max(I'_1 + 2I'_2 - 3n, \lfloor (-I'_1 + I'_2 + 3) / 2 \rfloor + (I'_1 + 2I'_2 - \lfloor (-I'_1 + I'_2 + 3) / 2 \rfloor) \% 3, I'_1 - I'_2 - 3n) & \text{ to } \min(I'_1 + 2I'_2 - 3, \lfloor (-I'_1 + I'_2 + 3n) / 2 \rfloor, I'_1 - I'_2 - 3) \text{ by } 3
\end{align*}
\]

To see how the transformed loop nest works, fix \( I'_1 = 6 \). The two inner loops become:

\[
\begin{align*}
\text{for } I'_1 \text{ from } -3 & \text{ to } 6 \text{ by } 1 \\
\text{for } I'_2 \text{ from } \lfloor (I'_2 - 3) / 2 \rfloor + (2I'_2 - \lfloor (I'_2 - 3) / 2 \rfloor) \% 3 & \text{ to } \min(2I'_2 + 3, -I'_2 + 3) \text{ by } 3
\end{align*}
\]

The set of all iterations \((I'_1, I'_2, I'_3)\) is contained in the intersection of the image iteration space and the hyperplane \(\{(I'_1, I'_2, I'_3) \mid I'_1 = 6\}\) (Fig. 4):

\[
\begin{align*}
I'_1 &= 6 \\
-3 &\leq I'_2 \leq 6 \\
(I'_2 - 3) / 2 &\leq I'_3 \leq \min(2I'_2 + 3, -I'_2 + 3)
\end{align*}
\]

It can be easily verified why \( I'_3 \) always starts to iterate at a lattice point. Note that the upper bound of \( I'_2 \) is 6. The iterations when \( I'_2 \) ranges from 4 to 6 are redundant. This is the problem inherent in the Fourier-Motzkin method.
Example 4 Consider the triple loop:

for $I_1$ from 1 to $m$ by 1
for $I_2$ from 1 to $n$ by 1
for $I_3$ from 1 to $p$ by 1

and the following non-unimodular transformation:

\[
\begin{bmatrix}
I'_1 \\
I'_2 \\
I'_3
\end{bmatrix}
= \begin{bmatrix}
2 & 6 & 1 \\
4 & 7 & 7 \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
I_1 \\
I_2 \\
I_3
\end{bmatrix}, \text{ i.e. }
\begin{bmatrix}
I'_1 \\
I'_2 \\
I'_3
\end{bmatrix}
= \begin{bmatrix}
-7/10 & 6/10 & -35/10 \\
4/10 & -2/10 & 10/10 \\
0 & 0 & 10/10
\end{bmatrix}
\begin{bmatrix}
I_1 \\
I_2 \\
I_3
\end{bmatrix}
\]

An application of the Fourier-Motzkin method yields:

\[
L'_{1,1} = 9 \\
U'_{1,1} = 2m+6n+p \\
\max([L'_{2,1}, L'_{2,2}, L'_{2,3}]) = \max(\{((7I'_1+45)/6, 2I'_1-5n+5, 7I'_1-10m-35n\}) \\
\min([U'_{2,1}, U'_{2,2}, U'_{2,3}]) = \min((7I'_1-45, 2I'_1+5p-5, (7I'_1+10m+35p)/6)) \\
\max([L'_{3,1}, L'_{3,2}, L'_{3,3}]) = \max(1, \{(-7I'_1+6I'_2-10m)/35, \{(-4I'_1+2I'_2+10)/10\}) \\
\min([U'_{3,1}, U'_{3,2}, U'_{3,3}]) = \min(p, \{(-7I'_1+6I'_2-10)/35, \{(-4I'_1+2I'_2+10n)/10\})
\]

Following Step 2, we obtain

\[
\Delta = \begin{bmatrix}
1 & 0 & 0 \\
2 & 5 & 0 \\
1 & 0 & 2
\end{bmatrix}
\]

By Step 3, we read off the loop strides from the diagonal of $\Delta$:

\[step'_1=1, \ step'_2=5, \ step'_3=2\]

By Step 4, the lower bound offsets are:

\[
\delta'_{1,1} = \frac{9}{9} \times 1 = 0 \\
\delta'_{2,1} = (2I'_1 - ((7I'_1+45)/6)) \times 5 \\
\delta'_{2,2} = (2I'_1 - (2I'_1-5n+5)) \times 5 = 0 \\
\delta'_{2,3} = (2I'_1 - (7I'_1-10m-35n)) \times 5 = 0 \\
\delta'_{3,1} = (I'_1-1) \times 2 \\
\delta'_{3,2} = (I'_1 - ((-7I'_1+6I'_2-10m)/35)) \times 2 \\
\delta'_{3,3} = (I'_1 - ((-4I'_1+2I'_2+10)/10)) \times 2
\]

The transformed loop nest is:

for $I'_1$ from 9 to $2m+6n+p$ by 1
for $I'_2$ from $\max\{((7I'_1+45)/6, (7I'_1+45)/6) \times 5, 2I'_1-5n+5, 7I'_1-10m-35n\}$
  to $\min((7I'_1-45, 2I'_1+5p-5, (7I'_1+10m+35p)/6))$ by 5
for $I'_3$ from $\max(1, (I'_1-1) \times 2, \{(-7I'_1+6I'_2-10m)/35, (I'_1 - ((-7I'_1+6I'_2-10m)/35)) \times 2, \{(-4I'_1+2I'_2+10)/10, (I'_1 - ((-4I'_1+2I'_2+10)/10)) \times 2\}$
  to $\min(p, \{(-7I'_1+6I'_2-10)/35, \{(-4I'_1+2I'_2+10n)/10\})$ by 2

14
Figure 5: The space of all iterations \((23, I_2', I_3')\) \((m=18, n=7, p=3)\).

To see how the transformed loop nest works, we fix \(I_1' = 23\). The two inner loops become:

\[
\text{for } I_2' \text{ from } 36 \text{ to } 56 \text{ by } 5 \\
\text{for } I_3' \text{ from } \max(1, [(I_2'-41)/5] + (1-[I_2'-41]/5)]/2 \text{ to } \min(3, [(I_2'-171)/35]) \text{ by } 2
\] \hspace{1cm} (17)

The set of all iterations \((23, I_2', I_3')\) is contained in the intersection of the image iteration space and the hyperplane \(\{(I_1', I_2', I_3') \mid I_1' = 23\}\) (Fig. 5):

\[
I_1' = 23 \\
35 \leq I_2' \leq 56 \\
\max(1, (I_2'-41)/5) \leq I_3' \leq \min(3, (6I_2'-171)/35)
\]

Refer to the transformed loop nest above. When \(I_1' = 23\), \(\max([I_2'_{2,1}], [L_2'_{2,2}], [L_2'_{3,3}]) = [L_2'_{2,1}] = 35\) and \(\delta_{2,1}' = 1\). Hence, \([L_2'_{2,1}] + \delta_{2,1}' = 36\). This is why in the loop nest \((17)\), \(I_2'\) starts to iterate at \(36\). Without \(\delta_{2,1}'\), \(I_2'\) would start to iterate at \(35\). The transformed loop nest would be incorrect, because \((23, 35, I_3')\) is always a hole for every integer \(I_3'\).

Let us see how \(I_3'\) always starts to iterate at a lattice point. Consider the space of all iterations \((23, I_2', I_3')\) in Fig. 5. The bottom boundary is \(L_{3,1} = I_3'\). The right boundary is \(I_{3,3}' = (I_2'-41)/5 = I_3'\). When \(I_2' = 36, 41\) and \(46\), the lower bound of \(I_3'\) is \([I_2'_{3,1}] = 1\). Thus, \(I_3'\) starts to iterate at \((23, 36, 1), (23, 41, 1)\) and \((23, 46, 1)\), respectively. These three points are the lattice points on the bottom boundary. When \(I_2' = 46, 51\) and \(56\), the lower bound of \(I_3'\) is \([I_2'_{3,3}] = ([I_2'-41]/5)\), which evaluates to \(1, 2\) and \(3\) and the lower bound offset \(\delta_{3,3}' = (1-[I_2'-41]/5)]/2\) evaluates to \(0, 1\) and \(0\), respectively. Thus, \(I_3'\) starts to iterate at the three lattice points \((23, 46, 1), (23, 51, 3)\) and \((23, 56, 3)\), respectively. Without \(\delta_{3,3}'\), \(I_3'\) would iterate from the hole \((23, 51, 2)\) when \(I_2' = 51\).

9 Conclusions

We have presented an algorithm for automatic rewriting loop nests for any non-singular loop transformations. The rewriting of loop nests for non-unimodular transformations does not seem as difficult as it appears to be. The non-polynomial step of the algorithm lies in the calculation of the loop bounds \(U_{k,i}'\) and \(U_{k,j}'\) using the Fourier-Motzkin method. This step is needed for both unimodular and non-unimodular transformations. The calculation of the loop strides \(\text{step}_k\) and lower bound offsets \(\delta_{k,i}'\) for non-unimodular transformations is polynomial. All information necessary for the calculation is implicitly contained in the loop transformation \(T\). The main contribution of this paper is to present a method that transforms the loop transformation \(T\) into a lower triangular matrix we called \(\Delta\) from which the loop strides and lower bound offsets can be automatically derived.
The usefulness of non-unimodular transformations has been well exploited in regular array
design. Our future work is to investigate how non-unimodular transformations can contribute in
the context of parallelising compilation.

References

SIGPLAN Symp. on Principles & Practice of Parallel Programming (PPoPP)*, pages 39–50.


T. Gross, and D. Padua, editors, *Advances in Languages and Compilers for Parallel Process-


[12] H. Le Verge, C. Mauras, and P. Quinton. The ALPHA language and its use for the design of


