The 5th IEEE Symposium on Parallel and Distributed Processing, IEEE Computer Society Press, Dallas, December 1993

# An Algorithm to Automate Non-Unimodular Transformations of Loop Nests

Jingling Xue
School of Electrical and Electronic Engineering
Nanyang Technological University
Singapore 2263

## Abstract

*This paper provides a solution to the open problem of automatic rewriting loop nests for non-unimodular transformations. We present an algorithm that rewrites a loop nest under any non-singular (unimodular or non-unimodular) transformation. The algorithm works nicely with unimodular transformations being treated as a special case. The first step of the algorithm calculates the loop bounds using the Fourier-Motzkin elimination method. The second step relies on a method based on the theory of Hermite normal form and lattice and is only needed for non-unimodular transformations. 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 polynomial, which is the time complexity of calculating the Hermite normal form of an $n \times n$ matrix, where $n$ is the depth of the loop nest. The adjusted loop bounds are the simplest that can be expected.*

## 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, 23] and regular array design [6, 7, 15, 16, 19, 25]. 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, 11, 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 [10, 20] for algorithms in regular array design and [1, 21] in parallelising compilation. However, these algorithms work only for unimodular loop transformations.

In the context of regular array design, loop transformations (often referred to as space-time mappings) can be unimodular or non-unimodular. In fact, non-unimodular transformations have been used to advantage in the synthesis of fixed-size arrays [7]. However, the issue of rewriting loop nests for loop transformations has not been adequately addressed. There have been some work on generating systolic code from loop nests for programmable 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 restricted to be unimodular [1, 3, 21]. This restriction was relaxed in [12] to increase the space of loop transformations. But how to rewrite loop nests for non-unimodular transformations was not mentioned.

In this paper, we present an algorithm that rewrites a loop nest under any non-singular (unimodular or non-unimodular) loop transformation in a mechanical manner. This algorithm works by first calculating the loop bounds using the Fourier-Motzkin elimination method that has been previously promoted in the literature, and then adjusting these loop bounds and calculating the loop strides using a method based on the theory of Hermite normal form and lattice.

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 loop transformations. Sect. 7 examines our loop rewriting algorithm from a geometrical perspective. Sect. 8 demonstrates further the algorithm through the rewriting of two more example loop nests. Sect. 9 concludes the paper.

## 2 Unimodularity, lattice, hermite normal form

We recall several concepts in linear algebra that are central to the paper [14, 18].

**Definition 1** A square matrix is *unimodular* if it is integral and has determinant $\pm 1$.

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 $\mathcal{L}(A) = \{y \mid y = Ax \wedge x \in \mathbb{Z}^n\}$ 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 is an $n \times n$ unimodular matrix $C$ such that $AC = [H\ 0]$ and $H$ is in Hermite normal form.*

By Thm. 1, $\mathcal{L}(A) = \mathcal{L}(H)$. Every integer matrix of full row rank has a unique Hermite normal form.

## 3 Problem statement

$\mathbb{Z}$ denotes the set of integers. As is customary, we write $\lceil x \rceil$ for the *ceiling* of $x$, i.e., the smallest integer not smaller than $x$, and we write $\lfloor x \rfloor$ 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 $0 \le r < |y|$.

We represent a loop nest of depth $n$ as follows:

$$\begin{array}{ll} \textbf{for } I_1 & \textbf{from } \max(\lceil L_{1,1} \rceil, \lceil L_{1,2} \rceil, \cdots) \\ & \textbf{to } \quad \min(\lfloor U_{1,1} \rfloor, \lfloor U_{1,2} \rfloor, \cdots) \ \textbf{by } step_1 \\ & \cdots \\ \textbf{for } I_n & \textbf{from } \max(\lceil L_{n,1} \rceil, \lceil L_{n,2} \rceil, \cdots) \\ & \textbf{to } \quad \min(\lfloor U_{n,1} \rfloor, \lfloor U_{n,2} \rfloor, \cdots) \ \textbf{by } step_n \end{array} \quad (1)$$

where the lower bounds $L_{k,i}$ and upper bounds $U_{k,j}$ are of the form:

$$\begin{array}{rcl} L_{k,i} & = & (\ell^0_{k,i} + \ell^1_{k,i} I_1 + \cdots + \ell^{k-1}_{k,i} I_{k-1}) / \ell^k_{k,i} \\ U_{k,j} & = & (u^0_{k,j} + u^1_{k,j} I_1 + \cdots + u^{k-1}_{k,j} I_{k-1}) / u^k_{k,j} \end{array} \quad (2)$$

where $\ell^p_{k,i}, u^q_{k,j} \in \mathbb{Z}$ 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 $step_k$. 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. 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:

$$\begin{array}{c} \max(L_{1,1}, L_{1,2}, \cdots) \le I_1 \le \min(U_{1,1}, U_{1,2}, \cdots) \\ \cdots \\ \max(L_{n,1}, L_{n,2}, \cdots) \le I_n \le \min(U_{n,1}, U_{n,2}, \cdots) \end{array} \quad (3)$$

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.

A loop transformation is an $n \times n$ non-singular integer matrix and is denoted by $T$. To be *legal*, a loop transformation must respect the dependences of the loop nest. Still, the space of legal loop transformations is infinite. Finding optimal transformations from this space is a difficult task. We refer to [19, 24] for methods on the minimisation of the latency and processor count of a regular array and to [8] for the maximisation of parallelism in a loop nest. The issues of legality and optimality of loop transformations are of no concern here. Our loop rewriting algorithm applies for any non-singular loop transformation.

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

$$\ll \quad = \quad \{(I, J) \mid I < J \wedge I, J \in \Phi\}$$

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

$$\ll_T \quad = \quad \{(TI, TJ) \mid TI < TJ \wedge I, J \in \Phi\}$$

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*. 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 $step_k = 1$. A loop transformation $T$ maps an iteration $(I_1, \cdots, I_n)$ to an iteration $(I'_1, \cdots, I'_n)$:

$$\begin{bmatrix} I'_1 \\ \vdots \\ I'_n \end{bmatrix} = T \begin{bmatrix} I_1 \\ \vdots \\ I_n \end{bmatrix}, \text{ i.e. } \begin{bmatrix} I_1 \\ \vdots \\ I_n \end{bmatrix} = T^{-1} \begin{bmatrix} I'_1 \\ \vdots \\ I'_n \end{bmatrix} \quad (4)$$

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

$$\begin{array}{ll} \textbf{for } I'_1 & \textbf{from } \max(\lceil L'_{1,1} \rceil + \delta'_{1,1}, \lceil L'_{1,2} \rceil + \delta'_{1,2}, \cdots) \\ & \textbf{to } \quad \min(\lfloor U'_{1,1} \rfloor, \lfloor U'_{1,2} \rfloor, \cdots) \ \textbf{by } step'_1 \\ & \cdots \\ \textbf{for } I'_n & \textbf{from } \max(\lceil L'_{n,1} \rceil + \delta'_{n,1}, \lceil L'_{n,2} \rceil + \delta'_{n,2}, \cdots) \\ & \textbf{to } \quad \min(\lfloor U'_{n,1} \rfloor, \lfloor U'_{n,2} \rfloor, \cdots) \ \textbf{by } step'_n \end{array} \quad (5)$$

For convenience, we still refer to $L'_{k,i}$ as the lower bounds. 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 $step'_k$ 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 Irigoin proposed to use the Fourier-Motzkin method. See [18] for details about the Fourier-Motzkin method and [1] about Ancourt and Irigoin'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 the linear system defining $\Phi'$ along $I'_n, \cdots, I'_2$ 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 $I'_{n-k+1}$:

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

where $L'_{n-k+1,i}$ and $U'_{n-k+1,j}$ conform to the format of (2) and contain $I'_1, \cdots, I'_{n-k}$ only. The new system with $I'_{n-k+1}$ eliminated is the input in the $(k+1)$-st projection. After the $(n-1)$-st, i.e., the last projection along $I'_2$, 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:

$$\begin{aligned} &\max(L'_{1,1}, L'_{1,2}, \cdots) \le I'_1 \le \min(U'_{1,1}, U'_{1,2}, \cdots) \\ &\qquad \cdots \\ &\max(L'_{n,1}, L'_{n,2}, \cdots) \le I'_n \le \min(U'_{n,1}, U'_{n,2}, \cdots) \end{aligned}$$

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. From now on, whenever we speak of *lattice points* we mean the lattice points in $\mathcal{L}(T)$. We refer to the integer points in $\mathbb{Z}^n \setminus \mathcal{L}(T)$ as the *non-lattice points*. When we say $(I'_1, \cdots, I'_k, *)$ is a lattice point with its last $n-k$ components unspecified, we mean that there are $X'_{k+1}, \cdots, X'_n$ in $\mathbb{Z}$ such that $(I'_1, \cdots, I'_k, X'_{k+1}, \cdots, X'_n)$ is a lattice point. That is, if $(I'_1, \cdots, I'_k, *)$ is a lattice point, then $(I'_1, \cdots, I'_k)$ belongs to the projection of $\mathcal{L}(T)$ onto $I'_1, \cdots, I'_k$.

If the loop transformation $T$ is unimodular, then $\mathcal{L}(T) = \mathbb{Z}^n$. 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 $\mathcal{L}(T) \neq \mathbb{Z}^n$. 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.

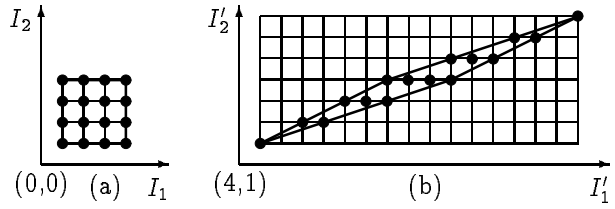


Figure 1: The iteration space $\Phi$ (a) and the image iteration space $\Phi'$ (b) of Ex. 1 ($n=4$).

Let $(I'_1, \cdots, I'_k, *)$ be a lattice point. The loop stride $step'_k$ of $I'_k$ is simply the positive integer such that $(I'_1, \cdots, I'_k + step'_k, *)$ is the smallest (in the lexicographical sense) lattice point larger than $(I'_1, \cdots, I'_k, *)$. Since $\mathcal{L}(T)$ is a lattice, the existence of $step'_k$ is guaranteed. To specify the lower bound offset $\delta'_{k,i}$, we consider the set of lattice points $(I'_1, \cdots, I'_{k-1}, *)$. If $(I'_1, \cdots, I'_{k-1}, \lceil L'_{k,i} \rceil, *)$ is a lattice point whenever $(I'_1, \cdots, I'_{k-1}, *)$ is, we set $\delta'_{k,i} = 0$. In general, we choose $\delta'_{k,i}$, which is an expression of $I'_1, \cdots, I'_{k-1}$, such that $(I'_1, \cdots, I'_{k-1}, \lceil L'_{k,i} \rceil + \delta'_{k,i}, *)$ is the smallest lattice point not smaller than $(I'_1, \cdots, I'_{k-1}, \lceil L'_{k,i} \rceil, *)$. 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. In this case, we simply set $step'_k = 1$ and $\delta'_{k,i} = 0$. To calculate $L'_{k,i}$ and $U'_{k,j}$, we apply the Fourier-Motzkin method.

**Example 1** Consider the double loop:

**for** $I_1$ **from** 1 **to** $n$ **by** 1
**for** $I_2$ **from** 1 **to** $n$ **by** 1

and the following unimodular transformation:

$$\begin{bmatrix} I'_1 \\ I'_2 \end{bmatrix} = \begin{bmatrix} 3 & 2 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} I_1 \\ I_2 \end{bmatrix}, \text{ i.e. } \begin{bmatrix} I_1 \\ I_2 \end{bmatrix} = \begin{bmatrix} 1 & -2 \\ -1 & 3 \end{bmatrix} \begin{bmatrix} I'_1 \\ I'_2 \end{bmatrix} \quad (6)$$

Fig. 1(a) depicts the corresponding iteration space $\Phi$:

$$\begin{aligned} 1 \le I_1 \le n \\ 1 \le I_2 \le n \end{aligned} \quad (7)$$

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:

$$\begin{aligned} 1 \le \quad I'_1 - 2I'_2 \le n \\ 1 \le -I'_1 + 3I'_2 \le n \end{aligned} \quad (8)$$

This system of linear constraints is not triangular, since the loop bounds of $I'_1$ are not readily available.

We use the Fourier-Motzkin method to rewrite this system into the following triangular system:

$$\begin{array}{l}(I'_1-n)/2 \le I'_2 \le (I'_1-1)/2\\ (I'_1+1)/3 \le I'_2 \le (I'_1+n)/3\end{array}$$

which is equivalent to:

$$\max((I'_1-n)/2, (I'_1+1)/3) \le I'_2 \le \min((I'_1-1)/2, (I'_1+n)/3) \quad (9)$$

This gives rise to the lower and upper bounds of $I'_2$. Eliminating $I'_2$ from (9), we obtain

$$\begin{array}{l}(I'_1-n)/2 \le (I'_1-1)/2\\ (I'_1-n)/2 \le (I'_1+n)/3\\ (I'_1+1)/3 \le (I'_1-1)/2\\ (I'_1+1)/3 \le (I'_1+n)/3\end{array}$$

which simplifies to:

$$5 \le I'_1 \le 5n \quad (10)$$

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:

**for** $I'_1$ **from** 5 **to** $5n$ **by** 1
**for** $I'_2$ **from** $\max(\lceil (I'_1-n)/2 \rceil, \lceil (I'_1+1)/3 \rceil)$
**to** $\min(\lfloor (I'_1-1)/2 \rfloor, \lfloor (I'_1+n)/3 \rfloor)$ **by** 1

# 6 Non-unimodular transformations

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

**Example 2** Consider 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:

**for** $I'_1$ **from** $5+\delta'_{1,1}$ **to** $5n$ **by** $step'_1$
**for** $I'_2$ **from** $\max(1+\delta'_{2,1}, \lceil (I'_1-2n)/3 \rceil + \delta'_{2,2})$ (11)
**to** $\min(n, \lfloor (I'_1-2)/3 \rfloor)$ **by** $step'_2$

Fig. 2 depicts is 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 $step'_1$ and $step'_2$ and the lower bound offsets $\delta'_{1,1}$, $\delta'_{2,1}$ and $\delta'_{2,2}$ in the transformed loop nest (11) are to be determined. If we simply set

$$step'_1=1,\ step'_2=1,\ \delta'_{1,1}=0,\ \delta'_{2,1}=0,\ \delta'_{2,2}=0$$

The transformed loop nest thus obtained is incorrect because of the presense of the non-lattice points in

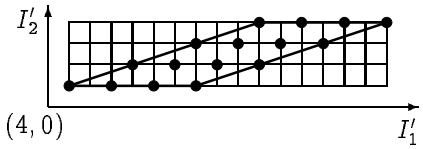


Figure 2: The image iteration space $\Phi'$ of Ex. 2 ($n=4$).

$\Phi'$. In fact, our algorithm returns the following loop strides and lower bound offsets:

$step'_1=1, step'_2=2,$
$\delta'_{1,1}=0, \delta'_{2,1}=(I'_1-1)\%2, \delta'_{2,2}=(I'_1-\lceil (I'_1-2n)/3 \rceil)\%2$

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 $step'_1$ is set to 1, because if $(I'_1, I'_2) \in \mathcal{L}(T)$, then $(I'_1+1, *) \in \mathcal{L}(T)$. The loop stride $step'_2$ is set to 2, because if $(I'_1, I'_2) \in \mathcal{L}(T)$ then $(I'_1, I'_2+1) \notin \mathcal{L}(T)$ and $(I'_1, I'_2+2) \in \mathcal{L}(T)$. It is easy to understand why $\delta'_{1,1}=0$. To see why $\delta'_{2,1}=(I'_1-1)\%2$, we consider the bottom boundary of the image iteration space, i.e., $L'_{2,1}=1=I'_2$. $L'_{2,1}=1$ is a constant. It represents the second components of the integer points on the bottom boundary. Some of these integer points are non-lattice points. This is where the lower bound offset $\delta'_{2,1}=(I'_1-1)\%2$ comes into play. It evaluates to 0 and 1 alternatively starting from the first iteration $I'_1=5$. Thus, $\lceil L'_{2,1} \rceil + \delta'_{2,1}$ evaluates to 1 and 2 alternatively, giving rise to the second components of the lattice points closest to the bottom boundary, as intended. The analysis for $\delta'_{2,2}$ is similar and is 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 implicitly 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 $T$:

$$\Delta = TC$$

($C$ is an $n \times n$ unimodular matrix.) By Thm. 1, $\mathcal{L}(T) = \mathcal{L}(\Delta)$. I.e. 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 use the following notations for $\Delta$. $\Delta_{i,j}$ denotes its $(i,j)$-th element. $\Delta_k$ denotes its $k$-th column.

## 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 $\mathcal{L}(T)$ 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:

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

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

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

**Proof.** By the definition of $\mathcal{L}_k(T)$, $0 \in \mathcal{L}_k(T)$. It suffices to show that (1) $\Delta_{k,k} \in \mathcal{L}_k(T)$ and (2) $x \notin \mathcal{L}_k(T)$, for $0 < x < \Delta_{k,k}$. The columns of $\Delta$ belong to $\mathcal{L}(T)$. The first $k-1$ components of the $k$-th column of $\Delta$ is 0 and the $k$-th component is $\Delta_{k,k}$. Hence, $\Delta_{k,k} \in \mathcal{L}_k(T)$. This proves (1). Let $(0, \cdots, 0, x, *)$ 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 $(0, \cdots, 0, x, *) \notin \mathcal{L}(T)$. $\Delta$ is a lower triangular matrix. If $(0, \cdots, 0, x, *) \in \mathcal{L}(T)$, $\Delta_{k,k}$ must divide $x$. This is impossible, since $0 < x < \Delta_{k,k}$. □

**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. □

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

$$\Delta = \begin{bmatrix} 1 & 0 \\ 1 & 2 \end{bmatrix} = \begin{bmatrix} 3 & 2 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ -1 & -3 \end{bmatrix} = TC \quad (12)$$

So $step'_1 = 1$ and $step'_2 = 2$ are the desired loop strides.

The loop strides $step'_k$ are unique, since the Hermite normal form $\Delta$ is.

## 6.2 The derivation of lower bound 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 $\Delta$, $(I'_1, \cdots, I'_{k-1}, \lceil L'_{k,i} \rceil, *)$ becomes $(X'_{1,i}, \cdots, X'_{k,i}, *)$, where $X'_{1,i}, \cdots, X'_{k,i}$ are the solutions to the equation:

$$\Delta \begin{bmatrix} X'_{1,i} \\ \vdots \\ X'_{k,i} \\ * \end{bmatrix} = \begin{bmatrix} I'_1 \\ \vdots \\ I'_{k-1} \\ \lceil L'_{k,i} \rceil \\ * \end{bmatrix} \quad (13)$$

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

$$\delta'_{k,i} = (-X'_{k,i}\Delta_{k,k}) \% \Delta_{k,k} \quad (14)$$

By this formula, $\delta'_{1,i} = \lceil L'_{1,i} \rceil \% \Delta_{1,1}$. $\Delta_{1,1}$ is the greatest common divisor of the elements in the first row of $T$. Hence, $\Delta_{1,1}$ divides $L'_{1,i}$. This implies that $\delta'_{1,i} \equiv 0$.

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

**Proof.** The transformed loop nest is correct if it has the lexicographical order $\ll_T$. 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 two facts. (1) With the adjusted lower bounds, every loop variable $I'_k$ starts to iterate at a lattice point $(I'_1, \cdots, I'_{k-1}, \max(\lceil L'_{k,1} \rceil + \delta'_{k,1}, \lceil L'_{k,2} \rceil + \delta'_{k,2}, \cdots), *)$, which is the smallest lattice point not smaller than $(I'_1, \cdots, I'_{k-1}, \max(\lceil L'_{k,1} \rceil, \lceil L'_{k,2} \rceil, \cdots), *)$, and (2) all lattice points whose first $k-1$ components are $I'_1, \cdots, I'_{k-1}$ have the form $(I'_1, \cdots, I'_{k-1}, \max(\lceil L'_{k,1} \rceil + \delta'_{k,1}, \lceil L'_{k,2} \rceil + \delta'_{k,2}, \cdots) + step'_k, *)$ (Thm. 3). □

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

$$\begin{aligned} \delta'_{1,1} &= \lceil L'_{1,1} \rceil \% \Delta_{1,1} = 5 \% 1 = 0 \\ \delta'_{2,1} &= (I'_1 - \lceil L'_{2,1} \rceil) \% \Delta_{2,2} = (I'_1 - 1) \% 2 \\ \delta'_{2,2} &= (I'_1 - \lceil L'_{2,2} \rceil) \% \Delta_{2,2} = (I'_1 - \lceil (I'_1 - 2n)/3 \rceil) \% 2 \end{aligned}$$

## 6.3 The loop rewriting algorithm

Our loop rewriting algorithm is given below.

**ALGORITHM** *loop_Rewrite*

**INPUT:** a loop transformation $T$, a loop nest in the form of (1) where $step_k = 1$.

**OUTPUT:** a loop nest in the form of (5) whose lexicographical order is $\ll_T$.

1. Calculate the loop bounds $L'_{k,i}$ and $U'_{k,i}$ using the Fourier-Motzkin method.
2. Calculate the Hermite normal form $\Delta$ of $T$.
3. Set $step'_k = \Delta_{k,k}$.
4. Set $\delta'_{k,i} = (-X'_{k,i}\Delta_{k,k}) \% \Delta_{k,k}$, where $X'_{k,i}$ 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 $L'_{k,i}$ and $U'_{k,j}$. Ancourt and Irigoin' 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 $L'_{k,i}$ and $U'_{k,j}$ 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, $\Delta$ is the identity matrix. We have $\Delta_{k,k} \equiv 1$. Hence, $step'_k = 1$ and $\delta'_{k,i} = 0$. In this case, Steps 2 – 4 can be dispensed with.

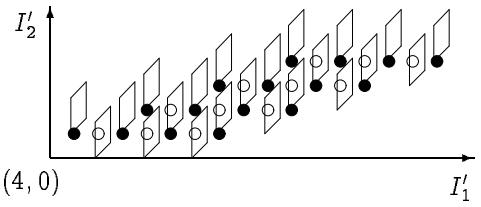


Figure 3: The tiling of the image iteration space of Ex. 2 ($n=4$).

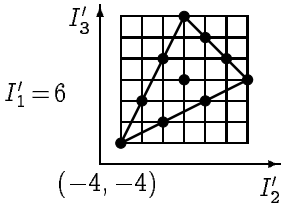


Figure 4: The space of all iterations $(6, I'_2, I'_3)$ ($n=4$).

## 7 A geometrical interpretation of the algorithm

This sections provides a geometrical insight into the development of our 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:

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

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

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 holes are bypassed.

## 8 Two examples

**Example 3** Consider the triple loop:

**for** $I_1$ **from** 1 **to** $n$ **by** 1
**for** $I_2$ **from** 1 **to** $n$ **by** 1
**for** $I_3$ **from** 1 **to** $n$ **by** 1

The following non-unimodular transformation specifies Kung-Leiserson's two-dimensional systolic array for band-matrix product [9]. This array is appealing 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:

$$\begin{aligned} L'_{1,1} &= 3 \\ U'_{1,1} &= 3n \\ (L'_{2,1}, L'_{2,2}, L'_{2,3}) &= (1-n, -I'_1+3, I'_1-3n) \\ (U'_{2,1}, U'_{2,2}, U'_{2,3}) &= (n-1, I'_1-3, -I'_1+3n) \\ (L'_{3,1}, L'_{3,2}, L'_{3,3}) &= (I'_1+2I'_2-3n, (-I'_1+I'_2+3)/2, I'_1-I'_2-3n) \\ (U'_{3,1}, U'_{3,2}, U'_{3,3}) &= (I'_1+2I'_2-3, (-I'_1+I'_2+3n)/2, I'_1-I'_2-3) \end{aligned}$$

Following Step 2, we obtain

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

By Step 3, the loop strides are its diagonal elements:

$$step'_1 = 1, \ step'_2 = 1, \ step'_3 = 3$$

The calculation of the lower bound offsets follows Step 4. Trivially, $\delta'_{1,1} = \delta'_{2,1} = \delta'_{2,2} = \delta'_{2,3} = 0$, because $\Delta_{1,1} = \Delta_{2,2} = 1$. The lower bound offsets of $I'_3$ are:

$$\begin{aligned} \delta'_{3,1} &= (I'_1+2I'_2-(I'_1+2I'_2-3n))\%3 = 0 \\ \delta'_{3,2} &= (I'_1+2I'_2-\lceil(-I'_1+I'_2+3)/2\rceil)\%3 \\ \delta'_{3,3} &= (I'_1+2I'_2-(I'_1-I'_2-3n))\%3 = 0 \end{aligned}$$

Due to space limitations, the transformed loop nest is not shown here explicitly. It follows from (5). To see how the transformed loop nest works, we fix $n=4$ and $I'_1=6$. The two inner loops become:

**for** $I'_2$ **from** $-3$ **to** 6 **by** 1
**for** $I'_3$ **from** $\lceil(I'_2-3)/2\rceil + (2I'_2 - \lceil(I'_2-3)/2\rceil)\%3$
**to** $\min(2I'_2+3, -I'_2+3)$ **by** 3

The set of all iterations $(6, I'_2, I'_3)$ 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):

$$\begin{aligned} I'_1 &= 6 \\ -3 \le I'_2 &\le 6 \\ (I'_2-3)/2 \le I'_3 &\le \min(2I'_2+3, -I'_2+3) \end{aligned}$$

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:

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

Following Step 2, we obtain

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

By Step 3, the loop strides are its diagonal elements:

$$step'_1 = 1,\ step'_2 = 5,\ step'_3 = 2$$

By Step 4, the lower bound offsets are:

$$\begin{aligned}
\delta'_{1,1} &= 9\%1 = 0 \\
\delta'_{2,1} &= (2I'_1 - \lceil (7I'_1+45)/6 \rceil)\%5 \\
\delta'_{2,2} &= (2I'_1 - (2I'_1 - 5n + 5))\%5 = 0 \\
\delta'_{2,3} &= (2I'_1 - (7I'_1 - 10m - 35n))\%5 = 0 \\
\delta'_{3,1} &= (I'_1 - 1)\%2 \\
\delta'_{3,2} &= (I'_1 - \lceil (-7I'_1 + 6I'_2 - 10m)/35 \rceil)\%2 \\
\delta'_{3,2} &= (I'_1 - \lceil (-4I'_1 + 2I'_2 + 10)/10 \rceil)\%2
\end{aligned}$$

The transformed loop nest is not shown here; it follows from (5). To see how the transformed loop nest works, we fix $m = 18$, $n = 7$, $p = 3$, and $I'_1 = 23$. The two inner loops become:

**for** $I'_2$ **from** 36 **to** 56 **by** 5
**for** $I'_3$ **from** $\max(1, \lceil (I'_2-41)/5 \rceil + (1 - \lceil (I'_2-41)/5 \rceil)\%2$
**to** $\min(3, \lfloor (6I'_2-171)/35 \rfloor)$ **by** 2

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):

$$\begin{aligned}
I'_1 &= 23 \\
35 \le I'_2 &\le 56 \\
\max(1, (I'_2-41)/5) \le I'_3 &\le \min(3, (6I'_2-171)/35)
\end{aligned}$$

Refer to the transformed loop nest above. When $I'_1 = 23$, $\max(\lceil L'_{2,1} \rceil, \lceil L'_{2,2} \rceil, \lceil L'_{2,3} \rceil) = \lceil L'_{2,1} \rceil = 35$ and $\delta'_{2,1} = 1$. Hence, $\lceil L'_{2,1} \rceil + \delta'_{2,1} = 36$. This is why in the loop nest, $I'_2$ 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 $I'_3$.

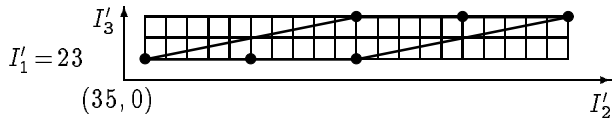


Figure 5: The space of all iterations $(23, I'_2, I'_3)$ ($m = 18$, $n = 7$, $p = 3$).

Let us see how $I'_3$ always starts to iterate at a lattice point. Consider Fig. 5. The bottom boundary is $L'_{3,1} = 1 = I'_3$. The right boundary is $L'_{3,3} = (I'_2-41)/5 = I'_3$. When $I'_2 = 36, 41$ and 46, the lower bound of $I'_3$ is $\lceil L'_{3,1} \rceil = 1$. 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 $\lceil L'_{3,3} \rceil = \lceil (I'_2-41)/5 \rceil$, which evaluates to 1, 2 and 3 and the lower bound offset $\delta'_{3,3} = (1 - \lceil (I'_2-41)/5 \rceil)\%2$ 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 $I'_1 = 23$ and $I'_2 = 51$.

## 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 $L'_{k,i}$ 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 $step'_k$ 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 relevance of non-unimodular transformations to regular array design has been 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 [13]. The advantage is that control signals [25], which would be required if differing transformations are used, are dispensed with. If the iteration space is infinite [15], and if, in addition, the goal of loop transformation is to maximise parallelism, then the space of legal transformations may contain non-unimodular transformations only.

In parallelising compilation, different transformations lead to syntactically different loop nests. It is

evident that once scheduled for execution in parallel machines, these loop nests can differ in many ways, say, in the amount of data transfer required among the memory hierarchy [1]. So far, loop transformations have been confined to be unimodular [1, 3, 21, 22]. Now, we have had an automatic method for the rewriting of loop nests for non-unimodular transformations. Our future work is to investigate how non-unimodular transformations can contribute to loop parallelisation.

## Acknowledgement

Thanks to the referees for comments and to Irigoin and Ancourt for answering my questions about their papers.

## References

[1] C. Ancourt and F. Irigoin. Scanning polyhedra with DO loops. In *Proc. Third ACM SIGPLAN Symp. on Principles & Practice of Parallel Programming (PPoPP)*, pages 39–50. ACM Press, Apr. 1991.

[2] U. Banerjee. *Dependence Analysis for Supercomputing.* The Kluwer Int. Series in Engineering and Computer Science: Parallel Processing and Fifth Generation Computing. Kluwer Academic Publishers, 1988.

[3] U. Banerjee. Unimodular transformations of double loops. In A. Nicolau, D. Gelernter, T. Gross, and D. Padua, editors, *Advances in Languages and Compilers for Parallel Processing*, chapter 10, pages 192–219. MIT Press, 1991.

[4] M. Barnett. *A Systolizing Compiler.* PhD thesis, Department of Computer Sciences, The University of Texas at Austin, Feb. 1992. Technical Report TR-92-13.

[5] M. Barnett and C. Lengauer. Unimodularity considered not essential. In L. Bouge, M. Cosnard, Y. Robert, and D. Trystram, editors, *Parallel Processing: CONPAR92–VAPP V*, Lecture Notes in Computer Science 634, pages 659–664. Elsevier (North-Holland), 1992.

[6] P. Clauss, C. Mongenet, and G. R. Perrin. Calculus of space-optimal mappings of systolic algorithms on processor arrays. In S. Y. Kung and E. E. Swartzlander, editors, *Application Specific Array Processors*, pages 5–18. IEEE Computer Society Press, 1990.

[7] A. Darte. Regular partitioning for synthesizing fixed-size systolic arrays. *Integration*, 12(3):293–304, Dec. 1991.

[8] F. Irigoin. Loop reordering with dependence direction vectors. Technical Report EMP-CAI-I A/184, Ecole Nationale Superieure des Mines de Paris, Nov. 1988.

[9] H. T. Kung and C. E. Leiserson. Algorithms for VLSI processor arrays. In C. Mead and L. Conway, editors, *Introduction to VLSI Systems*, chapter 8.3. Addison-Wesley, 1980.

[10] H. Le Verge, C. Mauras, and P. Quinton. The ALPHA language and its use for the design of systolic arrays. *J. VLSI Signal Processing*, 3:173–182, 1991.

[11] Z. Li, P. C. Yew, and C. Q. Zhu. An efficient data dependence analysis for parallelising compiler. *IEEE Trans. on Parallel and Distributed Systems*, 1(1):26–34, Jan. 1990.

[12] L.-C. Lu and M. Chen. New loop transformation techniques for massive parallelism. Technical Report YALEU/DCS/TR-833, Department of Computer Science, Yale University, Oct. 1990.

[13] D. I. Moldovan. On the design of algorithms for VLSI systolic arrays. *Proc. IEEE*, 71(1):113–120, Jan. 1983.

[14] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization.* Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, 1988.

[15] P. Quinton and V. van Dongen. The mapping of linear recurrence equations on regular arrays. *J. VLSI Signal Processing*, 1(2):95–113, Oct. 1989.

[16] S. K. Rao. *Regular Iterative Algorithms and their Implementations on Processor Arrays.* PhD thesis, Department of Electrical Engineering, Stanford University, Oct. 1985.

[17] H. B. Ribas. *Automatic Generation of Systolic Programs from Nested Loops.* PhD thesis, Department of Computer Science, Carnegie-Mellon University, June 1990. Technical Report CMU-CS-90-143.

[18] A. Schrijver. *Theory of Linear and Integer Programming.* Series in Discrete Mathematics. John Wiley & Sons, 1986.

[19] W. Shang and W. A. B. Fortes. Time optimal linear schedules for algorithms with uniform dependences. *IEEE Trans. on Computers*, C-40(6):723–742, June 1991.

[20] V. van Dongen and M. Petit. PRESAGE: A tool for the parallelization of nested loop programs. In L. J. M. Claesen, editor, *Formal VLSI Specification and Synthesis (VLSI Design Methods-I)*, pages 341–359. North-Holland, 1990.

[21] M. Wolf and M. Lam. A loop transformation theory and an algorithm to maximize parallelism. *IEEE Trans. on Parallel and Distributed Systems*, 2(4):452–471, Oct. 1991.

[22] M. J. Wolfe. *Optimizing Supercompilers for Supercomputers.* Research Monographs in Parallel and Distributed Computing. MIT Press, 1989.

[23] M. J. Wolfe. Data dependence and program restructuring. *J. of Supercomputing*, 4(4):321–344, Jan. 1991.

[24] Y. Wong and J. M. Delosme. Optimization of processor count for systolic arrays. Technical Report YALEU/DCS/RR-697, Department of Computer Science, Yale University, May 1989.

[25] J. Xue and C. Lengauer. The synthesis of control signals for one-dimensional systolic arrays. *Integration*, 14(1):1–32, Nov. 1992.