Implementation of Fourier-Motzkin Elimination [*]

Aart J.C. Bik and Harry A.G. Wijshoff
High Performance Computing Division
Department of Computer Science, Leiden University
P.O. Box 9512, 2300 RA Leiden, the Netherlands
ajcbik@cs.leidenuniv.nl

Abstract

Every transformation of a perfectly nested loop consisting of a combination of loop interchanging, loop skewing and loop reversal can be modeled by a linear transformation represented by a unimodular matrix. This modeling offers more flexibility than the traditional step-wise application of loop transformations because we can directly construct a unimodular matrix for a particular goal. In this paper, we present implementation issues arising when this framework is incorporated in a compiler.

1 Introduction

Inherent to the application of program transformations in an optimizing or restructuring compiler is the so-called 'phase ordering problem', i.e. the problem of finding an effective order in which particular transformations must be applied. This problem is still an important research topic [WS90]. An important step forwards in solving the phase ordering problem has been accomplished by the observation that any combination of the iteration-level loop transformations loop interchanging, loop skewing and loop reversal (see e.g. [AK87, Ban93, PW86, Pol88, Wol86, Wol88, Wol89, Zim90]) can be represented by a unimodular matrix [Ban90, Ban93, Dow90, WL91]. The advantage of this approach is that the order and validity of individual transformations becomes irrelevant, because a unimodular transformation can be constructed directly for a particular goal provided that dependence constraints are accounted for. The incorporation of this framework in a compiler requires the implementation of a mathematical technique known as Fourier-Motzkin elimination [AI89, DE73, LP92]. After application of a loop transformation, new loop bounds must be generated. However, derivation of these bounds requires the conversion of a system of inequalities in an arbitrary form into a form from which these loop bounds can be generated. In this paper, we discuss the implementation of Fourier-Motzkin elimination and present techniques to simplify the resulting bounds.

First, some preliminaries are presented in section 2. In section 3, we discuss code generation after application of a transformation represented by a unimodular matrix and present Fourier-Motzkin elimination. In section 4, some conclusions are drawn.

[*]Support was provided by the Foundation for Computer Science (SION) of the Dutch Organization for Scientific Research (NWO) and the EC Esprit Agency DG XIII under Grant No. APPARC 6634 BRA III.

2 Loop Transformations

In this section we give an outline of the general approach to iteration-level loop transformations in terms of unimodular matrices.

2.1 Iteration Spaces

In FORTRAN, the DO-loop is an important construct to define iteration. If individual DO-loops are used within other DO-loops, a so-called nested loop results. Below, a nested loop of depth d is illustrated:

```
DO I_1 = L_1, U_1
  ...
    DO I_d = L_d, U_d
      B(I_1,...,I_d)
    ENDDO
  ...
ENDDO
```

We will use the following terminology [Pol88, Wol89, Zim90]. If no other statements appear in between the individual DO-loops, then the whole loop is called a **perfectly nested loop**.

A loop in which arbitrary statements, or even complete other DO-loops appear in between the individual DO-loops is referred to as a **non-perfectly nested loop**. The **loop-body** $\mathtt{B}(\mathtt{I}_1, \ldots, \mathtt{I}_d)$ of the nested loop shown above consists of a sequence of **indexed statements** at nesting depth d. Each individual indexed statement in this loop-body is denoted by $S_i(\vec{\mathtt{I}})$ for the index vector $\vec{\mathtt{I}} = (\mathtt{I}_1, \ldots, \mathtt{I}_d)$.

If the loop-body is executed for the value $\vec{\mathtt{I}} = (i_1, \ldots, i_d)$, we call $\vec{\imath} = (i_1, \ldots, i_d)$ an **iteration (vector)** of this loop. Substituting $\vec{\imath}$ for $\vec{\mathtt{I}}$ in an indexed statement in this loop-body yields the **instance** of this statement that is executed during this iteration. The set of all iterations for which the loop-body of a nested loop is executed is called the **iteration space** of this loop. Because only integer variables can be used as loop indices, this iteration space is a subset of the discrete Cartesian space $\mathbf{Z}^d$.

Usually we assume that each DO-loop is stride 1, so that the loop index $\mathtt{I}_i$ iterates over all integers in the closed interval $[\mathtt{L}_i, \mathtt{U}_i]$. The value of a lower bound $\mathtt{L}_i$ and upper bound $\mathtt{U}_i$ may depend on the values of indices of more outer DO-loops. A relatively simple kind of loop bounds is formed by **basic bounds**, in which a lower bound $\mathtt{L}_i$ or an upper bound $\mathtt{U}_i$ can be expressed as an affine mapping from $\mathbf{Z}^{i-1}$ to $\mathbf{Z}$ as shown below, where $l_{ij}, u_{ij} \in \mathbf{Z}$:

$$\mathtt{L}_i = l_{i0} + \sum_{j=1}^{i-1} l_{ij} \cdot \mathtt{I}_j \quad \mathtt{U}_i = u_{i0} + \sum_{j=1}^{i-1} u_{ij} \cdot \mathtt{I}_j$$

Although most programmers use such basic bounds, it is possible that during program restructuring other kinds of loops are generated. A slightly more difficult class of bounds is formed by the **simple bounds**, in which a lower bound $\mathtt{L}_i$ and an upper bound $\mathtt{U}_i$ of a loop index $\mathtt{I}_i$ can be expressed as follows, where all $l_{ij}, u_{ij} \in \mathbf{Z}$ and $l_{ii} > 0$ and $u_{ii} > 0$:

$$\mathtt{L}_i = \left\lceil \frac{l_{i0} + \sum_{j=1}^{i-1} l_{ij}\mathtt{I}_j}{l_{ii}} \right\rceil \mathtt{U}_i = \left\lfloor \frac{u_{i0} + \sum_{j=1}^{i-1} u_{ij}\mathtt{I}_j}{u_{ii}} \right\rfloor$$

Obviously, a basic lower or upper bound is also a simple bound for which either $l_{ii} = 1$ or $u_{ii} = 1$. Finally, we will consider **compound bounds**, in which each lower bound consists of the maximum of a number of simple bounds, and each upper bound consists of the minimum of a number of simple bounds:

$$\mathtt{L}_i \texttt{ = MAX(} \mathtt{L}_i^1\texttt{, } \mathtt{L}_i^2\texttt{, ...)} \quad \mathtt{U}_i \texttt{ = MIN(} \mathtt{U}_i^1\texttt{, } \mathtt{U}_i^2\texttt{, ...)}$$

Each simple bound $\mathtt{L}_i^k$ in a compound lower bound gives rise to an inequality of the following form, where the ceiling function has been eliminated using the fact index $\mathtt{I}_i$ is an integer variable and $l_{ii}^k > 0$:

$$l_{i0}^k + \sum_{j=1}^{i-1} l_{ij}^k \cdot \mathtt{I}_j \ \le\ l_{ii}^k \cdot \mathtt{I}_i$$

Because we can take $l_{ij}^k = 0$ for $j > i$, this inequality can be expressed as a scalar product with $\vec{\mathtt{I}} = (\mathtt{I}_1, \ldots, \mathtt{I}_d)$, effectively defining a half-space in $\mathbf{R}^d$:

$$(l_{i1}^k \ldots l_{i,i-1}^k \ -l_{ii}^k \quad \underbrace{0 \ldots 0}_{d-i}) \quad \cdot \vec{\mathtt{I}} \quad \le \quad -l_{i0}^k$$

Similarly, each simple bound $\mathtt{U}_i^k$ in a compound upper bound gives rise to the definition of a half-space in $\mathbf{R}^d$:

$$(-u_{i1}^k \ldots - u_{i,i-1}^k \ u_{ii}^k \quad \underbrace{0 \ldots 0}_{d-i}) \quad \cdot \vec{\mathtt{I}} \quad \le \quad u_{i0}^k$$

Because in a compound lower bound the maximum of a number of simple bounds is taken and, likewise, in each compound upper bound the minimum of some simple bounds, all the corresponding inequalities must be satisfied simultaneously. Therefore, we can represent the bounds of a nested loop having compound bounds as a system of inequalities $A\vec{\mathtt{I}} \le \vec{b}$. This system is obtained by adding one row to A and one component to $\vec{b}$ according to one of the scalar products shown above for each inequality arising from one of the simple bounds.

Basic bounds can also be dealt with by simply setting $l_{ii}^k = 1$ and $u_{ii}^k = 1$. A lower or upper bound expressed without a maximum or minimum function respectively gives rise to only one inequality. In this manner, a system $A\vec{\mathtt{I}} \le \vec{b}$ is obtained, in which each inequality defines a half-space in $\mathbf{R}^d$. Therefore, for a $k \times d$ matrix A, the whole system represents the intersection of k half-spaces, forming a so-called polyhedral set $PS \subset \mathbf{R}^d$. Furthermore, because only finite execution sets can be used in FORTRAN, the polyhedral set PS is bounded and, hence, forms a so-called convex polytope [Grü67]. Finally, because only integer variables are used as loop indices, the iteration space of the loop is defined as $IS = \mathbf{Z}^d \cap PS$, which means that only all discrete points within the convex polytope are taken.

In the following loop, for instance, a compound upper bound is used for loop index $\mathtt{I}_2$:

```
DO I1 = 0, 7
  DO I2 = I1, MIN(I1+3, 8)
    DO I3 = 0, 7-I1
      B(I1,I2,I3)
    ENDDO
  ENDDO
ENDDO
```

The system of inequalities representing the loop bounds, obtained by considering bounds in increasing order of nesting depth, is shown below:

$$\begin{pmatrix} -1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & -1 & 0 \\ -1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \\ 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} \mathrm{I}_1 \\ \mathrm{I}_2 \\ \mathrm{I}_3 \end{pmatrix} \leq \begin{pmatrix} 0 \\ 7 \\ 0 \\ 3 \\ 8 \\ 0 \\ 7 \end{pmatrix}$$

2.2 Unimodular Matrices

Every iteration-level loop transformation of a perfectly nested loop of depth d consisting of a combination of loop interchanging, loop skewing, and loop reversal can be modeled by a mapping between the **original** and **target iteration space**, namely a linear transformation represented by a unimodular matrix. A unimodular matrix U is an $d \times d$ integer matrix (i.e. $u_{ij} \in \mathbf{Z}$) for which $|\det(U)| = 1$ holds. The property of transformations represented by such matrices is that (1) integer points are mapped onto integer points, and that, (2) since the inverse of a unimodular matrix is also a unimodular matrix, every integer point in the target iteration space corresponds to an integer point in the original iteration space. Moreover, the transformations are closed under composition.

Each iteration $\vec{\imath} \in IS$ in the original iteration space is mapped onto an iteration $\vec{\imath}' = U\vec{\imath}$ in the target iteration space. Because iterations in the target iteration space are also traversed in lexicographic order, application of a transformation effectively results in a new execution order on the instances. Each loop reversal, loop interchanging, or loop skewing is represented by an elementary $d \times d$ matrix. Elementary matrices are unimodular matrices obtained from the unit matrix by either multiplying a row by -1, interchanging two rows, or adding an integral multiple of a row to another row. Pre- or post-multiplication of a matrix with an elementary matrix performs an elementary row or column operation respectively.

The results of application of some transformations represented by 2×2 elementary matrices to a double loop are shown below:

Original Loop:

```
DO I1 = 1, M
  DO I2 = 1, N
    L(I1,I2)
  ENDDO
ENDDO
```

Reversal:

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

```
DO I'1 = -M, -1
  DO I'2 = 1, N
    L(-I'1,I'2)
  ENDDO
ENDDO
```

Interchanging:

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

```
DO I'1 = 1, N
  DO I'2 = 1, M
    L(I'2,I'1)
  ENDDO
ENDDO
```

Skewing:

$$\begin{pmatrix} 1 & 0 \\ p & 1 \end{pmatrix}$$

```
DO I'1 = 1, M
  DO I'2 = 1+p*I'1, N+p*I'1
    L(I'1,I'2-p*I'1)
  ENDDO
ENDDO
```

In general, reversal of the ith loop is represented by the unit matrix in which $u_{ii} = -1$ holds for exactly one element. Skewing the jth loop by a factor of p with respect to the ith loop, where $i < j$, is represented by the unit matrix having $u_{ij} = p$ for one off-diagonal element.

Each unimodular matrix can be decomposed into a number of such elementary matrices, and thus loop transformations. Conversely, any combination of iteration level loop transformations is represented by a unimodular matrix. Consequently, this approach offers more flexibility than the traditional step-wise application of loop transformations, where the usefulness and validity of each transformation has to be considered separately.

2.3 Validity of Application

Application of a unimodular transformation U is valid, if each data dependence in the original nesting is satisfied in the resulting nesting.

Dependence distance vectors provide a convenient representation of data dependences. If iteration $\vec{\imath}'$ depends on iteration $\vec{\imath}$, then we have $\vec{\imath} + \vec{d} = \vec{\imath}'$ for some distance vector $\vec{d}$. Induced by the sequential semantics of DO-loops, iterations are executed in lexicographic order. Consequently, each distance vector of a loop-carried data dependence is **lexicographically positive**, denoted by $\vec{d} \succ \vec{0}$, i.e. its first nonzero component is positive. Since U is a linear transformation, $U\vec{\imath}' - U\vec{\imath} = U(\vec{\imath}' - \vec{\imath})$ holds. Hence, application of a unimodular transformation U is valid if and only if $U\vec{d} \succ \vec{0}$ for each nonzero dependence distance $\vec{d}$ in the original nest, since this implies that the dependences are satisfied. In [WL91], dependence direction vectors are incorporated in the validity test.

2.4 Code Generation

Because $\vec{\mathtt{I}}' = U\vec{\mathtt{I}}$ can be rewritten into $\vec{\mathtt{I}} = U^{-1}\vec{\mathtt{I}}'$, the new loop-body and loop bounds are obtained by replacing each index $\mathtt{I}_j$ in the original body according to this equation. Consider, for example, that application of U to the following perfectly nested loop is valid:

```
DO I1= 0, 50
 DO I2= 0, 50-I1
   DO I3= 0, 50
     L(I1,I2,I3)
   ENDDO
 ENDDO
ENDDO
```

$$U = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \qquad U^{-1} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & -1 & -1 \end{pmatrix}$$

The resulting loop-body is obtained by replacing $\vec{\mathtt{I}}$ according to equation $\vec{\mathtt{I}} = U^{-1}\vec{\mathtt{I}}'$:

$$\begin{pmatrix} \mathtt{I}_1 \\ \mathtt{I}_2 \\ \mathtt{I}_3 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & -1 & -1 \end{pmatrix} \begin{pmatrix} \mathtt{I}'_1 \\ \mathtt{I}'_2 \\ \mathtt{I}'_3 \end{pmatrix}$$

The resulting loop bounds are derived by rewriting the system of inequalities that is obtained by substitution of $U^{-1}\vec{\mathtt{I}}'$ for $\vec{\mathtt{I}}$ in the system representing the original loop bounds. After redundant inequalities have been eliminated, the following code is generated:

```
DO I'1 = 0, 100
  DO I'2 = 0, MIN(50,I'1)
    DO I'3 = MAX(0,I'1-I'2-50), MIN(50-I'2,I'1-I'2)
      L(I'2,I'3,I'1-I'2-I'3)
    ENDDO
  ENDDO
ENDDO
```

The conversion of the original iteration space into the target iteration space is illustrated in figure 1. For example, as defined by the first row of U, all iterations in the plane $\mathtt{I}_1 + \mathtt{I}_2 + \mathtt{I}_3 = 1$ are mapped onto iterations in the plane $\mathtt{I}'_1 = 1$ in the target iteration space.

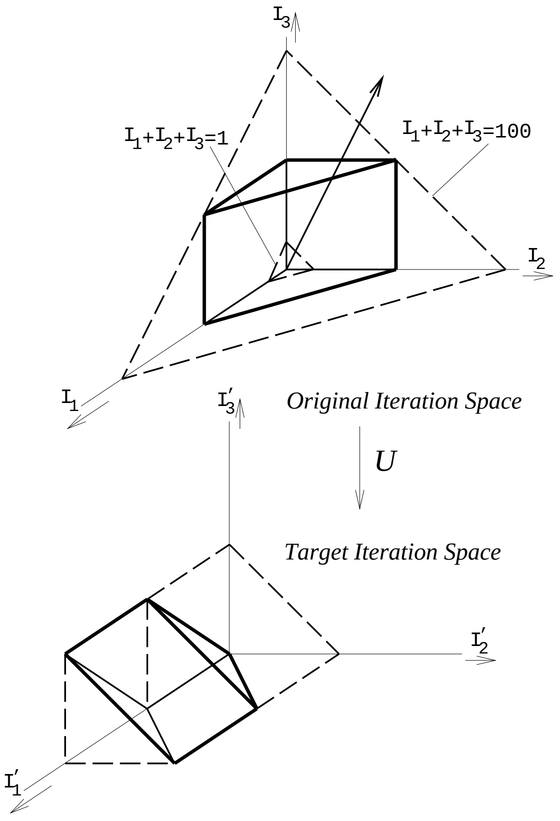

Figure 1: Application of a U

```
DO I1= 1, 3               DO I'1= 2, 4
  DO I2= I1+1, 4            DO I'2= 1, I'1-1
    S(I1,I2)        →         S(I'2,I'1)
  ENDDO                     ENDDO
ENDDO                     ENDDO
```

The new loop-body is obtained by replacing loop indices according to equation $\vec{\mathtt{I}} = U^{-1}\vec{\mathtt{I}}'$ (note that $U = U^{-1}$ in this case):

$$\begin{pmatrix} \mathtt{I}_1 \\ \mathtt{I}_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \mathtt{I}'_1 \\ \mathtt{I}'_2 \end{pmatrix} \qquad (1)$$

A first step towards finding the new loop bounds is to apply the substitution arising from (1) to the system of inequalities defined by the original loop:

$$\begin{array}{rcl} 1 \le & \mathtt{I}_1 & \le 3 \\ 1 + \mathtt{I}_1 \le & \mathtt{I}_2 & \le 4 \end{array} \rightarrow \begin{array}{rcl} 1 \le & \mathtt{I}'_2 & \le 3 \\ 1 + \mathtt{I}'_2 \le & \mathtt{I}'_1 & \le 4 \end{array}$$

The inequality $1 + \mathtt{I}'_2 \le \mathtt{I}'_1$ cannot be used directly to determine the lower bound on index $\mathtt{I}'_1$ of the outermost loop, because this bound is given in terms of the innermost loop index $\mathtt{I}'_2$. In general, substituting $U^{-1}\vec{\mathtt{I}}'$ for $\vec{\mathtt{I}}$ in the system of inequalities defined by the original loop yields a new system that is unsuited for generating the bounds directly.

3 Generating Loop Bounds

The application of a transformation represented by a unimodular matrix U to perfectly nested loop is implemented by rewriting the loop-body and generating new loops with indices $\vec{\mathtt{I}}'$ that induce a lexicographic traversal of the target iteration space. Since the equation $\vec{\mathtt{I}} = U^{-1}\vec{\mathtt{I}}'$ holds for the loop indices $\vec{\mathtt{I}}$ and $\vec{\mathtt{I}}'$ of the original and target iteration space respectively, the new loop-body can be obtained by replacing each index in the original loop-body according to this equation. Unfortunately, generating the new loop bounds is not so straightforward. Consider, for instance, application of a simple interchange to the following double loop:

Each bound can only be a function of outer loop indices and the resulting system is not necessarily of this form. Only the bounds of the *innermost* loop can be determined directly. For instance, in the previous example we have $1 \leq \mathtt{I}'_2$, $\mathtt{I}'_2 \leq 3$ and $\mathtt{I}'_2 \leq \mathtt{I}'_1 - 1$. Hence, the lower and upper bound of the $\mathtt{I}'_2$-loop are 1 and $\mathtt{MIN}(3, \mathtt{I}'_1 - 1)$ respectively. The bounds of $\mathtt{I}'_1$ can be obtained by eliminating $\mathtt{I}'_2$ from the system. This is performed by replacing all inequalities involving the loop index $\mathtt{I}'_2$ by inequalities in which each lower bound of $\mathtt{I}'_2$ is less than or equal to each upper bound of this index. In the example, we obtain:

$$\begin{array}{rcl} 1 & \leq & \mathtt{I}'_2 \\ \mathtt{I}'_2 & \leq & 3 \\ \mathtt{I}'_2 & \leq & \mathtt{I}'_1 - 1 \\ \mathtt{I}'_1 & \leq & 4 \end{array} \quad \rightarrow \quad \begin{array}{rcl} 1 & \leq & 3 \\ 1 & \leq & \mathtt{I}'_1 - 1 \\ \mathtt{I}'_1 & \leq & 4 \end{array}$$

Consequently, the lower and upper bound of $\mathtt{I}'_1$ can be expressed in terms of the constants 2 and 4 respectively, which is the appropriate form for the bounds of an outermost loop. At this point the valid range for index $\mathtt{I}'_1$ is known, and the upper bound of index $\mathtt{I}'_2$ can be simplified into $\mathtt{I}'_1 - 1$.

This example illustrates the problems that occur during the generation of new loop bounds. In the next sections, we present an automatic method which deals with these problems by successively eliminating loop indices in decreasing order of nesting depth. At each step of this method, the bounds for the last index in the system can be determined.

3.1 Fourier-Motzkin Elimination

After applying a transformation represented by a unimodular matrix U to a perfectly nested loop of which the bounds can be described as $A^0\vec{\mathtt{I}} \leq \vec{b}$, we have to derive the new loop bounds from the system of inequalities $A\vec{\mathtt{I}}' \leq \vec{b}$, where $A = A^0U^{-1}$. Since in this system the bounds of an index $\mathtt{I}'_i$ may depend on indices $\mathtt{I}'_j$ for $j > i$, we use Fourier-Motzkin elimination [AI89, AT93, Ban93, DE73, LP92] to derive a system in an appropriate form.

We successively eliminate index $\mathtt{I}'_k$ for decreasing value of k. Starting with $k = d$, an $m \times k$ matrix A and a vector $\vec{b}$ with m components, the system $A\vec{\mathtt{I}}' \leq \vec{b}$ gives rise to the following inequalities, for $1 \leq i \leq m$:

$$\sum_{j=1}^{k} a_{ij} \cdot \mathtt{I}'_j \leq b_i \qquad (2)$$

We can reorder this system according to the value of each coefficient a_{ik}, so that for particular $p, q \in \mathbf{N}$ we have $a_{ik} > 0$ for $1 \leq i \leq p$, $a_{ik} < 0$ for $p < i \leq q$ and $a_{ik} = 0$ for $q < i \leq m$, where $p \leq q \leq m$. This reordering gives rise to the following three sets of inequalities, in which only positive coefficients occur for index $\mathtt{I}'_k$:

$$\begin{array}{rcl} a_{ik} \cdot \mathtt{I}'_k & \leq & b_i - \sum_{j=1}^{k-1} a_{ij} \cdot \mathtt{I}'_j \\ -b_i + \sum_{j=1}^{k-1} a_{ij} \cdot \mathtt{I}'_j & \leq & (-a_{ik}) \cdot \mathtt{I}'_k \\ \sum_{j=1}^{k-1} a_{ij} \cdot \mathtt{I}'_j & \leq & b_i \end{array} \qquad (3)$$

The first p inequalities in this system define the upper bounds on index $\mathtt{I}'_k$. The next $q - p$ inequalities define the lower bounds on this index. Consequently, a minimum and a maximum function of these two sets of bounds are generated for the upper and lower bound respectively. Moreover, because these bounds may evaluate to rational constants and only integer values are allowed for loop indices, we use ceiling functions for lower bounds and floor functions for upper bounds. In case only one upper bound results ($p = 1$), the maximum function is omitted. Likewise, the minimum function is omitted if only one lower bound results ($q = p+1$). Ceiling and floor functions are omitted for all lower or upper bounds having $a_{ik} = 1$ in the corresponding inequality.

After generation of these bounds, index $\mathtt{I}'_k$ is eliminated from the system in order to enable the generation of bounds for more outer loop indices. In the original formulation of Fourier-Motzkin elimination [DE73], index $\mathtt{I}'_k$ is eliminated by replacing each pair of inequalities $L \leq c_1 \cdot \mathtt{I}'_k$ and $c_2 \cdot \mathtt{I}'_k \leq U$, where $c_1 > 0$ and $c_2 > 0$, by the inequality $L/c_1 \leq U/c_2$. However, we will replace the previous pair of inequalities by $c_2 \cdot L \leq c_1 \cdot U$. In this manner, a formulation is obtained in which only integer arithmetic is involved. Moreover, the resulting inequality is divided by $\gcd(c_1, c_2)$ to avoid overflow as much as possible.

This implies that the innermost loop index in the system of inequalities is eliminated by replacing the first q inequalities by the following $p \cdot (q - p)$ inequalities for all $1 \leq i \leq p$ and $p < i' \leq q$, where $g = \gcd(a_{i'k}, a_{ik})$, $c_i = a_{ik}/g$, and $c_{i'} = -a_{i'k}/g$:

$$\sum_{j=1}^{k-1} (c_{i'} \cdot a_{ij} + c_i \cdot a_{i'j}) \cdot \mathtt{I}'_j \leq c_{i'} \cdot b_i + c_i \cdot b_{i'}$$

Together with the last $m-q$ inequalities from (3), a new system of the form (2) has been obtained for a different m and a lower value of k. Repetitive application of the elimination of loop indices in decreasing order of nesting depth eventually yields all bounds in the loop nest resulting after application of a unimodular transformation. If during this process, an inequality of the form '$0 \leq c$' with $c<0$ is encountered, or if the maximum of all lower bounds of index $\texttt{I}_1$ is greater than the minimum of all upper bounds of this index,[1] we are dealing with an inconsistent system. In this context, this implies that the target iteration space and, hence, the original iteration space are empty. In general, it is also possible that some variables in the system are unbounded, although this will not occur for systems that are derived from the loop bounds in a program. Therefore, we exclude this latter possibility from our discussion.

In our implementation of Fourier-Motzkin elimination, at each step $k=d,\ldots,1$, a system of inequalities $A\vec{\texttt{I}}' \leq \vec{b}$, for some $m \times k$ matrix A and $\vec{\texttt{I}}' = (\texttt{I}'_1,\ldots,\texttt{I}'_k)^T$, is represented by the column augmented matrix $(A \mid \vec{b})$. The rows in this matrix are reordered according to the value of the elements in the kth column. For all pairwise combinations $1 \leq i \leq p$ and $p < i' \leq q$, the rows i and i' are added after multiplication with $a_{i'k}/g$ and a_{ik}/g respectively, where $g = \gcd(a_{i'k}, a_{ik})$. The resulting rows together with the last $m-q$ rows of the previous matrix, excluding all elements in the kth column, constitute the rows of the new matrix.

This process is applied recursively to the resulting matrix until *all* loop indices have been eliminated. Consequently, a sequence of column augmented matrices terminated by a column vector is obtained, as is illustrated below:

$$(A^{(d)} \mid \vec{b}^{(d)}) \rightarrow \ldots \rightarrow (A^{(1)} \mid \vec{b}^{(1)}) \rightarrow \vec{b}^{(0)} \quad (4)$$

Each column augmented matrix $(A^{(k)} \mid \vec{b}^{(k)})$ for $1 \leq k \leq d$ in this sequence has its own local parameters $p^{(k)}$, $q^{(k)}$ and $m^{(k)}$. However, as was done in this section, we will omit superscripts if it is clear from the context which matrix in the sequence is considered. The first matrix in a sequence consists of a reordered representation of the original system $A^0 U^{-1} \vec{\texttt{I}}' \leq \vec{b}$.

[1]This additional test is required because $\texttt{I}_1$ is the only loop index that is not eliminated. In our implementation, one additional elimination step is applied to enable a uniform test for consistency the last column vector.

If one of the components in the terminating column vector $\vec{b}^{(0)}$ is negative, the resulting system of inequalities is inconsistent. Furthermore, each column augmented matrix $(A^{(k)} \mid \vec{b}^{(k)})$ represents the bounds of index $\texttt{I}'_k$ according to the following system of inequalities:

$$A^{(k)} \begin{pmatrix} \texttt{I}'_1 \\ \vdots \\ \texttt{I}'_k \end{pmatrix} \leq \vec{b}^{(k)}$$

3.2 Example

Consider, for instance, application of a transformation represented by the following unimodular matrix U to the loop nest shown below:

```
DO I1 = 10, 15
  DO I2 = 1, 3
    DO I3 = 1, 50
      S(I1, I2, I3)
    ENDDO
  ENDDO
ENDDO
```

$$U = \begin{pmatrix} 0 & 6 & 1 \\ 1 & -3 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

The new loop body is obtained by replacing the original loop indices according to the equation $\vec{\texttt{I}} = U^{-1} \cdot \vec{\texttt{I}}'$. The new loop bounds are defined by a system of inequalities represented by the column augmented matrix $(A \mid \vec{b})$, where $A = A^0 U^{-1}$ and the system $A^0 \cdot \vec{\texttt{I}} \leq \vec{b}$ represents the original loop bounds:

$$A = \begin{pmatrix} 0 & -1 & -3 \\ 0 & 0 & -1 \\ -1 & 0 & 6 \\ 0 & 1 & 3 \\ 0 & 0 & 1 \\ 1 & 0 & -6 \end{pmatrix} \qquad \vec{b} = \begin{pmatrix} -10 \\ -1 \\ -1 \\ 15 \\ 3 \\ 50 \end{pmatrix}$$

Application of Fourier-Motzkin elimination to the column augmented matrix $(A \mid \vec{b})$ results in the following sequence of column augmented matrices and a terminating column vector:

$$\left(\begin{array}{rrr|r} -1 & 0 & 6 & -1 \\ 0 & 1 & 3 & 15 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & -1 & -1 \\ 0 & -1 & -3 & -10 \\ 1 & 0 & -6 & 50 \end{array}\right) \rightarrow \left(\begin{array}{rr|r} 0 & 1 & 12 \\ 1 & 2 & 80 \\ -1 & -2 & -21 \\ 0 & -1 & -1 \\ 0 & 0 & 5 \\ 0 & 0 & 49 \\ 0 & 0 & 2 \\ -1 & 0 & -7 \\ 1 & 0 & 68 \end{array}\right)$$

$$\rightarrow \left(\begin{array}{r|r} 1 & 78 \\ 1 & 68 \\ -1 & 3 \\ -1 & -7 \\ 0 & 5 \\ 0 & 49 \\ 0 & 2 \\ 0 & 59 \\ 0 & 11 \end{array}\right) \rightarrow \begin{pmatrix} 81 \\ 71 \\ 71 \\ 61 \\ 5 \\ 49 \\ 2 \\ 59 \\ 11 \end{pmatrix}$$

Because all components of $\vec{b}^{(0)}$ are positive, the resulting system of inequalities is consistent. The column augmented matrices in the sequence define the following loop bounds:

```
  DO I'_1= MAX(-3,7), MIN(78,68)
   DO I'_2= MAX(1,⌈(21-I'_1)/2⌉), MIN(12,⌊80-I'_1)/2⌋)
    DO I'_3= MAX(1,⌈(10-I'_2)/3⌉,⌈(I'_1-50)/6⌉),
+             MIN(3,⌊(15-I'_2)/3⌋,⌊(I'_1-1)/6⌋)
     S(I'_2+3*I'_3,I'_3,I'_1-6*I'_3)
    ENDDO
   ENDDO
  ENDDO
```

Because the lower bound of the outermost loop always evaluates to the value 7, we can simplify this lower bound. Likewise, the upper bound can be replaced by 68. However, in general less obvious simplifications are also possible, which is the topic of the following sections.

3.3 Ad-Hoc Simplification

The matrices arising from Fourier-Motzkin elimination may define bounds on a particular index that are not really necessary because other inequalities also define these bounds. The elimination of these so-called redundant bounds may result in the generation of more efficient code because less expressions need to be evaluated at run-time. Moreover, the evaluation of minimum or maximum functions is not required if a single bound remains. In this section we present a computationally inexpensive method to detect and eliminate some redundant bounds.

For each loop index $\mathtt{I}'_k$ in the resulting loop nest of depth d, we maintain four variables $l_k^{\min}, l_k^{\max}$ and $u_k^{\min}, u_k^{\max}$ that can have values in $\mathbf{Z} \cup \{-\infty, \infty\}$. Variables $l_k^{\min}$ and $l_k^{\max}$ are used to record the minimum and maximum value for the *lower* bounds of this loop index respectively. Likewise, variables $u_k^{\min}$ and $u_k^{\max}$ are used to record the minimum and maximum value of the *upper* bounds of index $\mathtt{I}'_k$ respectively. This implies that index $\mathtt{I}'_k$ can have integer values in the interval $[l_k^{\min}, u_k^{\max}]$. Initially, we set $l_k^{\min} = l_k^{\max} = -\infty$ and $u_k^{\min} = u_k^{\max} = \infty$ for all $1 \leq k \leq d$. Subsequently, more accurate values are determined during a backward scan over all column augmented matrices in the sequence. Hence, the new loop bounds are considered in *increasing order of nesting depth.* For each $(k+1) \times m$ column augmented matrix $(A \mid \vec{b})$ in this sequence, we can find p and q such that the first p rows represent the upper bounds of an index $\mathtt{I}_k$ in terms of indices $\mathtt{I}'_j$ for $j < k$ and the next $q - p$ rows represent the lower bounds of this index.

Lower and upper bounds can be expressed in the following form where $a_{ik} > 0$ for $1 \leq i \leq p$, and $a_{ik} < 0$ for $p < i \leq q$:

$$\frac{b_i + \sum_{j=1}^{k-1} (-a_{ij}) \cdot \mathtt{I}'_j}{a_{ik}} \tag{5}$$

The remaining bounds for $q < i \leq m$ do not define bounds on index $\mathtt{I}'_k$ and are eliminated (bounds arising from these inequalities are accounted for in other matrices in the sequence). Because $\mathtt{I}'_j \in [l_j^{\min}, u_j^{\max}]$ holds for $j < k$, the minimum value l and maximum value u of the numerator in expression (5) can be determined as follows, where we define $a^+ = \max(a, 0)$ and $a^- = \max(-a, 0)$ as done in [Ban88]:

$$\begin{array}{rcl} l &=& b_i + \sum_{j=1}^{k-1} (-a_{ij})^+ \cdot l_j^{\min} - (-a_{ij})^- \cdot u_j^{\max} \\ u &=& b_i + \sum_{j=1}^{k-1} (-a_{ij})^+ \cdot u_j^{\max} - (-a_{ij})^- \cdot l_j^{\min} \end{array}$$

Consequently, an upper bound $(a_{ik} > 0)$ can only have values in $[l', u']$, where $l' = \lfloor l/a_{ik} \rfloor$ and $u' = \lfloor u/a_{ik} \rfloor$. Therefore, if $u_k^{\max} \leq l'$ holds, then this upper bound is redundant with respect to previously considered upper bounds and is eliminated. Similarly, if $u' \leq u_k^{\min}$, this bound replaces all previously considered upper bounds of this index. After an upper bound has been considered, the following assignments are executed:

$$\begin{array}{rcl} u_k^{\min} &:=& \min(u_k^{\min}, l'); \\ u_k^{\max} &:=& \min(u_k^{\max}, u'); \end{array}$$

Likewise, a lower bound $(a_{ik} < 0)$ can only have values in $[l', u']$, where $l' = \lceil u/a_{ik} \rceil$ and $u' = \lceil l/a_{ik} \rceil$. Therefore, if $l_k^{\max} \leq l'$, this bound replaces all previously considered lower bounds of this index. The bound is eliminated if $u' \leq l_k^{\min}$. After consideration of a lower bound, we perform the next statements:

$$\begin{array}{rcl} l_k^{\min} &:=& \max(l_k^{\min}, l'); \\ l_k^{\max} &:=& \max(l_k^{\max}, u'); \end{array}$$

Continuing in this fashion eventually results in a new sequence of matrices in which some redundant bounds (viz. rows) are eliminated.

Consider, for instance, the double loop example presented at the beginning of section 3 again. Below, we show $A\vec{\mathtt{I}}' \leq \vec{b}$, where $A = A^0 U^{-1}$, in column augmented representation:

$$\left(\begin{array}{rr|r} 0 & -1 & -1 \\ -1 & 1 & -1 \\ 0 & 1 & 3 \\ 1 & 0 & 4 \end{array} \right)$$

Application of Fourier-Motzkin elimination yields the following sequence of matrices:

$$\left(\begin{array}{rr|r} -1 & 1 & -1 \\ 0 & 1 & 3 \\ 0 & -1 & -1 \\ 1 & 0 & 4 \end{array}\right) \rightarrow \left(\begin{array}{r|r} 1 & 4 \\ -1 & -2 \\ 0 & 2 \end{array}\right) \rightarrow \left(\begin{array}{r} 2 \\ 2 \end{array}\right)$$

Since all components of the terminating column vector are positive, the target iteration space is non-empty. Subsequently, the ad-hoc simplification method is performed during a backward scan of the column augmented matrices in this sequence. After consideration of the matrix defining the bounds of $\mathtt{I}'_1$, we know that $\mathtt{I}'_1 \in [2,4]$. We have $u_2^{\min} = 1$ and $u_2^{\max} = 3$, after the first upper bound of index $\mathtt{I}'_2$ has been considered. Since $l' = u' = 3$ for the other upper bound of this index, we have $u_2^{\max} \leq l'$ and this upper bound can be eliminated:

$$\left(\begin{array}{rr|r} -1 & 1 & -1 \\ 0 & -1 & -1 \end{array}\right) \rightarrow \left(\begin{array}{r|r} 1 & 4 \\ -1 & -2 \end{array}\right)$$

3.4 Exact Simplification

The ad-hoc method discussed in the previous section can be used as inexpensive method to eliminate some redundant bounds. However, the efficiency of the generated code can be further improved at the expense of a potential increase in compile-time by the incorporation of a more advanced method, which eliminates *all* redundant bounds including those not detected by the ad-hoc method.

The following exact simplification method is based on the observation that a particular inequality is already enforced by other inequalities in a system of inequalities, if the system obtained by negating this inequality is inconsistent [AI89]. For instance, negation of inequality $i \leq 11$ in the following system of inequalities yields $i > 11$, which can be rewritten into $12 \leq i$ for integer variables:

$$\begin{array}{rcl} 1 \leq & i & \leq 10 \\ & i & \leq 11 \end{array} \quad \xrightarrow{\text{Negation}} \quad \begin{array}{rcl} 1 \leq & i & \leq 10 \\ 12 \leq & i & \end{array}$$

Application of Fourier-Motzkin elimination to the resulting system yields $1 \leq 10$ and $12 \leq 10$, revealing the inconsistency of this system. Therefore, the third inequality is not required and can be eliminated from the system.

A matrix of the following form, where each $\vec{0}$ denotes a zero column vector of appropriate size, is constructed incrementally for successive steps $k = 1, \ldots, d$:

$$\left(\begin{array}{cccc|c} A^{(1)} & \vec{0} & \vec{0}\ldots\vec{0} & & \vec{b}^{(1)} \\ & A^{(2)} & \vec{0}\ldots\vec{0} & & \vec{b}^{(2)} \\ & & \ddots & & \vdots \\ & & & A^{(d)} & \vec{b}^{(k)} \end{array}\right) \tag{6}$$

At each step k, this matrix represents the bounds of the first k loop indices. The number of positive elements in column k of the column augmented matrix $(A^{(k)} \mid \vec{b}^{(k)})$ is equal to the number of upper bounds of index $\mathtt{I}'_k$. Similarly, the number of negative elements in column k of this matrix is equal to the number of lower bounds of this index. If there are *several* upper bounds, the first upper bound is negated and Fourier-Motzkin elimination is applied to the resulting system to test consistency. If the resulting system is consistent, the bound is recovered and the next upper bound is considered. Otherwise, the upper bound is eliminated. This process is repeated until all upper bounds have been considered or only one upper bound remains. Similar steps are performed while there are several lower bounds. Thereafter, the value of k is incremented and the next matrix of the form (6) is constructed until $k = d$.

In the matrix representation, negation of either a lower or an upper bound represented by the ith row, is performed as follows:

$$\left(\begin{array}{c|c} \vdots & \vdots \\ a_{i1}\ldots a_{ik} & b_i \\ \vdots & \vdots \end{array}\right) \rightarrow \left(\begin{array}{c|c} \vdots & \vdots \\ -a_{i1}\ldots - a_{ik} & -b_i - 1 \\ \vdots & \vdots \end{array}\right)$$

Consider, for example, a simple interchange of the $\mathtt{I}_2$- and $\mathtt{I}_3$-loop in the following fragment:

```
DO I1= 1, 100              DO I'1= 1, 100
 DO I2= I1, 100             DO I'2= I'1, 100
  DO I3= I1, I2              DO I'3= MAX(I'1,I'2), 100
   S(I1,I2,I3)      →         S(I'1,I'3,I'2)
  ENDDO                      ENDDO
 ENDDO                      ENDDO
ENDDO                      ENDDO
```

Ad-hoc simplification of the sequence arising from Fourier-Motzkin elimination yields the following sequence:

$$\left(\begin{array}{rrr|r} 0 & 0 & 1 & 100 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & -1 & 0 \end{array}\right) \rightarrow \left(\begin{array}{rr|r} 0 & 1 & 100 \\ 1 & -1 & 0 \end{array}\right) \rightarrow \left(\begin{array}{r|r} 1 & 100 \\ -1 & -1 \end{array}\right)$$

The ad-hoc method is not able to detect the fact that, because $\mathtt{I}'_1 \leq \mathtt{I}'_2$ is enforced by the lower bound of the second loop index, the lower bound of the third index can be simplified into the single expression $\mathtt{I}'_2$.

During exact simplification, the following sequence is constructed incrementally:

$$\left(\begin{array}{r|r} 1 & 100 \\ -1 & -1 \end{array}\right) \rightarrow \left(\begin{array}{rr|r} 1 & 0 & 100 \\ -1 & 0 & -1 \\ 0 & 1 & 100 \\ 1 & -1 & 0 \end{array}\right)$$

$$\rightarrow \left(\begin{array}{rrr|r} 1 & 0 & 0 & 100 \\ -1 & 0 & 0 & -1 \\ 0 & 1 & 0 & 100 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 100 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & -1 & 0 \end{array}\right)$$

No actions are performed for the first two matrices, because they define only one lower and upper bound for the first two indices. However, two lower bounds are defined on index $\mathtt{I}'_3$ in the last matrix. The first lower bound is negated, followed by application of Fourier-Motzkin elimination to the resulting matrix. The following sequence of matrices results, where the negated inequality is marked in the first matrix:

$$\left(\begin{array}{rrr|r} 0 & 0 & 1 & 100 \\ \hline -1 & 0 & 1 & -1 \\ \hline 0 & 1 & -1 & 0 \\ 1 & -1 & 0 & 0 \\ 1 & 0 & 0 & 100 \\ -1 & 0 & 0 & -1 \\ 0 & 1 & 0 & 100 \end{array}\right) \rightarrow \left(\begin{array}{rr|r} 0 & 1 & 100 \\ -1 & 1 & -1 \\ 0 & 1 & 100 \\ 1 & -1 & 0 \\ -1 & 0 & -1 \\ 1 & 0 & 100 \end{array}\right)$$

$$\rightarrow \left(\begin{array}{r|r} 1 & 100 \\ 1 & 100 \\ 1 & 100 \\ -1 & -1 \\ 0 & -1 \end{array}\right) \rightarrow \left(\begin{array}{r} 99 \\ 99 \\ 99 \\ -1 \end{array}\right)$$

Because this system is inconsistent, the lower bound can be eliminated. Since only one lower bound remains for the third index, no further actions are performed and the following sequence of matrices results, in which all redundant bounds have been eliminated:

$$\left(\begin{array}{rrr|r} 0 & 0 & 1 & 100 \\ 0 & 1 & -1 & 0 \end{array}\right) \rightarrow$$

$$\left(\begin{array}{rr|r} 0 & 1 & 100 \\ 1 & -1 & 0 \end{array}\right) \rightarrow \left(\begin{array}{r|r} 1 & 100 \\ -1 & -1 \end{array}\right)$$

We have deliberately chosen to consider all matrices of the form (6) separately for increasing values of k, so that only bounds that are redundant with respect to the possible values of more outer loop indices are eliminated. If only the *final* matrix (i.e. matrix (6) for which $k = d$) would be used to test the redundancy of bounds, some bounds could be redundant because additional iterations introduced by the removal of these bounds induce zero trip loops for more inner loops. So, evaluation overhead would be reduced at the expense of an increase of overhead caused by the execution of empty iterations. If the redundancy of *single* bounds would be tested, this approach could result in loops in which some indices become unbounded.

For example, the upper bound of the outermost index is redundant in the following fragment, because the inequality $\mathtt{I}'_1 \leq \mathtt{100}$ is also enforced by the bounds of the second loop index. Any upper bound greater than `100` could be used for $\mathtt{I}'_1$ without introducing additional iterations:

```
DO I'_1 = 1, 100                 DO I'_1 = 1, ∞
  DO I'_2 = I'_1, 100              DO I'_2 = I'_1, 100
    ...                   vs.        ...
  ENDDO                            ENDDO
ENDDO                            ENDDO
```

3.5 Comparisons

In this section, we compare the performance of the two simplification methods on some matrices of size $2 - 5$ having the property that some (but not all) redundant bounds are eliminated by the ad-hoc method.

In table 1, we show the number of remaining bounds after application of Fourier-Motzkin elimination, ad-hoc simplification and exact simplification. In table 2, we present the CPU-time in milli-seconds on an HP-UX 9000/720 for Fourier-Motzkin elimination, the ad-hoc method followed by the exact method, and the exact method without preceding application of the ad-hoc method to the examples. All versions are compiled with optimizations enabled (but have not been fully hand-optimized with respect to e.g. memory allocation).

Some conclusions can be drawn from this experiment. Because application of the exact method can be far more expensive than the actual Fourier-Motzkin elimination, it must be possible to disable the application of this simplification method. Furthermore, if advanced simplification is required, the total computational time can be reduced substantially by preceding application of the ad-hoc method, acting as a filter for the exact simplification. Therefore, the bounds that result after application of a unimodular transformation are always simplified according to the ad-hoc method, followed by the exact method if desired.

	F.M.	Ad-Hoc	Exact
2	6	4	4
3	14	8	6
4	34	16	8
5	138	36	10

Table 1: Number of Remaining Bounds

	F.M.	Ad-Hoc	Exact	Total	Exact Only
2	0.1	0.1	0.1	0.2	1.4
3	0.4	0.2	1.2	1.4	7.4
4	1.3	0.4	7.4	7.8	46.0
5	24.6	2.3	47.8	50.1	1539.0

Table 2: CPU-time in msecs

4 Conclusions

In this paper we have discussed an implementation of Fourier-Motzkin elimination. This method has been incorporated in a prototype restructuring compiler MT1 [Bik92] to support unimodular transformations. An ad-hoc simplification method is used in case compilation-time is at a premium. However, the ad-hoc method can also act as a filter for computationally more expensive exact simplification methods.

Acknowledgments The authors would like to thank Peter Knijnenburg, Arnold Niessen and Remco de Vreugd for proofreading this article.

References

[AI89] Corinne Ancourt and Francois Irigoin. Scanning polyhedra with do loops. In *Proceedings of Third ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 39–50, 1989.

[AK87] Randy Allen and Ken Kennedy. Automatic translation of FORTRAN programs to vector form. *ACM Transactions on Programming Languages and Systems*, Volume 9:491–542, 1987.

[AT93] Eduard Ayguadé and Jordi Torres. Partitioning the statement per iteration space using non-singular matrices. In *Proceedings of the International Conference on Supercomputing*, pages 407–415, 1993.

[Ban88] U. Banerjee. *Dependence Analysis for Supercomputing.* Kluwer Academic Publishers, Boston, 1988.

[Ban90] U. Banerjee. Unimodular transformations of double loops. In *Proceedings of Third Workshop on Languages and Compilers for Parallel Computing*, 1990.

[Ban93] U. Banerjee. *Loop Transformations for Restructuring Compilers: The Foundations.* Kluwer Academic Publishers, Boston, 1993.

[Ban94] U. Banerjee. *Loop Parallelization.* Kluwer Academic Publishers, Boston, 1994.

[Bik92] Aart J.C. Bik. A prototype restructuring compiler. Master's thesis, Utrecht University, 1992. INF/SCR-92-11.

[DE73] George B. Dantzig and B. Curtis Eaves. Fourier-Motzkin elimination and its dual. *Journal of Combinatorial Theory*, Volume 14:288–297, 1973.

[Dow90] Michael L. Dowling. Optimal code parallelization using unimodular transformations. *Parallel Computing*, Volume 16:157–171, 1990.

[Grü67] Branko Grünbaum. *Convex Polytopes.* Interscience Publishers, London, 1967.

[Lam74] Leslie Lamport. The parallel execution of do loops. *Communications of the ACM*, pages 83–93, 1974.

[LP92] Wei Li and Keshav Pingali. A singular loop transformation framework based on non-singular matrices. In *Proceedings of the Fifth Workshop on Languages and Compilers for Parallel Computing*, 1992.

[Pol88] C.D. Polychronoupolos. *Parallel Programming and Compilers.* Kluwer Academic Publishers, Boston, 1988.

[PW86] David A. Padua and Michael J. Wolfe. Advanced compiler optimizations for supercomputers. *Communications of the ACM*, pages 1184–1201, 1986.

[WL91] Michael E. Wolf and Monica S. Lam. A loop transformation theory and an algorithm to maximize parallelism. *IEEE Transactions on Parallel and Distributed Algorithms*, pages 452–471, 1991.

[Wol86] Michael J. Wolfe. Loop skewing: The wavefront method revisited. *International Journal of Parallel Programming*, Volume 15:279–293, 1986.

[Wol88] Michael J. Wolfe. Vector optimization vs. vectorization. *Journal of Parallel and Distributed Computing*, Volume 5:551–567, 1988.

[Wol89] Michael J. Wolfe. *Optimizing Supercompilers for Supercomputers.* Pitman, London, 1989.

[WS90] Debbie Whitfield and Mary Lou Soffa. An approach to ordering optimizing transformations. In *Proceedings of the second ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 137–146, 1990.

[Zim90] H. Zima. *Supercompilers for Parallel and Vector Computers.* ACM Press, New York, 1990.