On Reducing Overhead in Loops *

Peter M.W. Knijnenburg Aart J.C. Bik

High Performance Computing Division,
Dept. of Computer Science, Leiden University,
Niels Bohrweg 1, 2333 CA Leiden, the Netherlands.
E-mail: peterk@cs.leidenuniv.nl and ajcbik@cs.leidenuniv.nl

Abstract

In this paper we discuss several techniques for reducing the overhead in Fortran-like DO loops. In particular, we describe two topics:

1. Simplifying loop bounds.
2. Removing *affine* IF-*statements* from loops.

The objective is to isolate the part of the iteration space of the original loop in which the bounds take simpler forms, or in which the condition is satisfied. The main technical tool for determining this subspace is Fourier-Motzkin elimination. We show an easy way to generate code for scanning the different parts of an iteration space that have been isolated in this way. We present our techniques as a number of elementary transformation steps. SInce these steps only partition the iteration space, the steps can always legally be employed. The full effect of these steps is obtained by iterating them until no more changes occur. The result will be a loop structure in which all bounds are simple affine expressions and from which all IF-statements with affine conditions have been eliminated.

1 Introduction

It is well-known that many programs spend most execution time in only a small fraction of the code, namely, in loops [Kuc78]. Since this is particularly true for scientific applications, it is desirable to have the loops structured in such a way that overhead in the execution is minimized. In this paper we consider the following two sources of overhead:

1. *Overhead in computing the loop bounds.* This overhead occurs when bounds are given as the minimum or maximum of a number of elementary affine expressions. While programmers will probably not code their programs using these kind of bounds, they are generated when loops are transformed using unimodular transformations [Ban90, Ban93, Ban94, WL91]. We want to identify that part of the iteration space in which the minimum or maximum of a number of expressions $e_1, \ldots, e_n$ is given by the expression e_i. For in this subspace, we may replace the original bound by e_i thereby reducing the overhead of determining the loop bounds at run time. The techniques discussed in this paper can serve as a post-processing phase of unimodular transformations, in which the resulting loop structures are 'cleaned up'.

2. IF-*statements in the loop body.* These IF-statements cause a portion of the code in the loop to be executed in some iterations, and not in other iterations. At the same time, in each iteration it has to be decided which portion to execute by evaluating the condition of the IF-statement. It is therefore desirable to isolate at compile time the part of the iteration space in which this condition is true, and replace the IF-statement by its THEN-part, and by its ELSE-part in the remainder of the iteration space. In this way, no condition needs to be evaluated at run time. Of course, this cannot always be done. However, we identify a class of IF-statements, called *affine* IF-*statements*, for which this partitioning can be computed at compile time.

The main technical tool we employ for determining the relevant subspaces of the iteration space is *Fourier-Motzkin elimination* [DE73]. Fourier-Motzkin elimination is a way of solving systems of linear inequalities.

*This research was partially supported by Esprit BRA APPARC under grant no. 6634

Using this tool, we have to intersect the original iteration space with a polytope, defined by a system of linear inequalities, in order to obtain a loop structure that scans the iteration space but that executes different code in the different parts.

Although Fourier-Motzkin elimination is exponential in the number of inequalities in the system it solves, it has been observed before [BW95a, Pug92, LP92] that the algorithm has fast implementations for loops with small nesting depth, as is always the case in practical situations. This is particularly true when executed on current work-stations. Moreover, we can view the transformations presented in this paper as transferring computation from run time to compile time. The more effort one wants to spend to restructure loops at compile time, the less run time overhead one is left with. Obviously, the effort is better spend at compile time than at run time. Moreover, since our approach is to define a small collection of elementary transformations, the user can choose in an interactive compilation environment just how much effort he is willing to give.

Since the transformations discussed in this paper only partition the iteration space of the original loop, and hence leave the order in which the iteration points are visited intact, application is always valid. Moreover, under some mild conditions, every transformation step will improve the execution time of the loop. Hence we provide a powerful restructuring technique that is universally applicable.

The paper is organized as follows. In section 2 we give technical preliminaries and introduce some notation. In section 3 we discuss the algorithm for intersecting two polytopes. In section 4, we discuss some methods to simplify loop bounds and illustrate these methods with some small examples. In section 5 we show how affine `IF`-statements can be removed. In section 6 we discuss an extended example. Finally, in section 7 we give a brief discussion and relate the present work to another approach.

2 Preliminaries

In this section, we give some preliminaries used in the rest of this paper. First, we define lower and upper-bounds for the considered loops.

Definition 2.1 *Let $\mathcal{I} = \{\mathtt{I}_1, \ldots, \mathtt{I}_n\}$ be a finite collection of loop indices. With respect to this collection $\mathcal{I}$ we define:*

1. *A* basic lower/upper bound *is an affine expression $a_0 + a_1\mathtt{I}_1 + \cdots + a_n\mathtt{I}_n$ where, for all i, $a_i \in \mathbf{Z}$.*
2. *Let $a \in \mathbf{N}$ and let B be a basic bound. A* simple lower-bound *is an expression $\lceil \frac{1}{a}B \rceil$.*
   A simple upper-bound *is an expression $\lfloor \frac{1}{a}B \rfloor$.*
3. *A* compound lower-bound *is an expression $\max(L_1, \ldots, L_m)$ where each L_i is a simple lower-bound. A* compound upper-bound *is an expression $\min(U_1, \ldots, U_m)$ where each U_i is a simple upper-bound.*

Note that each basic bound is also a simple lower or upper-bound, and, in turn , each simple bound is also a compound bound. In the sequel of the paper we will make this identification when no confusion can arise. Basic, simple and compound bounds are also called *admissible*. All other expressions for bounds, like $\mathtt{I}_1 * \mathtt{I}_2$ or an indirection `IND(I)`, are called *inadmissible*. The reason for this distinction is that admissible bounds can be used and are obtained in the process of Fourier-Motzkin elimination (see below).

Every perfectly nested loop $\mathcal{L}$ with basic bounds gives rise to a system of inequalities $\mathcal{S}(\mathcal{L})$, given by

$$\mathcal{S}(\mathcal{L}) = \left\{ \begin{array}{ccccc} \mathtt{L}_1 & \leq & \mathtt{I}_1 & \leq & \mathtt{U}_1 \\ & & \vdots & & \\ \mathtt{L}_n & \leq & \mathtt{I}_n & \leq & \mathtt{U}_n \end{array} \right. \tag{1}$$

Such a system should be read as the *conjunction* of the individual clauses. This system has the property that every bound $\mathtt{L}_i$ and $\mathtt{U}_i$ only involves variables $\mathtt{I}_1, \ldots, \mathtt{I}_{i-1}$. We call this the *standard form* of a system of inequalities. Note that for loops with compound lower and upper-bounds we also can define such a system of inequalities. Since compound upper-bounds may contain floor and minimum function, we use the following equivalences to obtain a standard form:

- $\mathtt{I} \leq \lfloor \frac{1}{a}B \rfloor$ iff $a\mathtt{I} \leq B$ ($\mathtt{I}$ is integer, $a > 0$).
- $\mathtt{I} \leq \min(U_1, \ldots, U_m)$ iff $\mathtt{I} \leq U_1$ & $\cdots$ & $\mathtt{I} \leq U_m$.

Similar equivalences hold for compound lower-bounds.

Any system of inequalities involving the variables $\mathtt{I}_1, \ldots, \mathtt{I}_n$ can be brought in standard form using *Fourier-Motzkin elimination* [DE73, Ban93, LP92]. We try to give some intuition. Consider a set $\mathcal{C} = \{\varphi_1, \ldots, \varphi_k\}$ of inequalities, where each inequality φ_i is of the form $e \leq e'$ for two affine expressions e and e' over the variables $\mathtt{I}_1, \ldots, \mathtt{I}_n$. Then Fourier-Motzkin elimination consists of the following process. First, rewrite all expressions involving the variable $\mathtt{I}_n$ to the form $L \leq \mathtt{I}_n, \ldots, \mathtt{I}_n \leq U$. Then each inequality obtained in this way bounds $\mathtt{I}_n$ by expressions only involving the variables $\mathtt{I}_1, \ldots, \mathtt{I}_{n-1}$. Hence these expressions can be used to generate loop bounds.

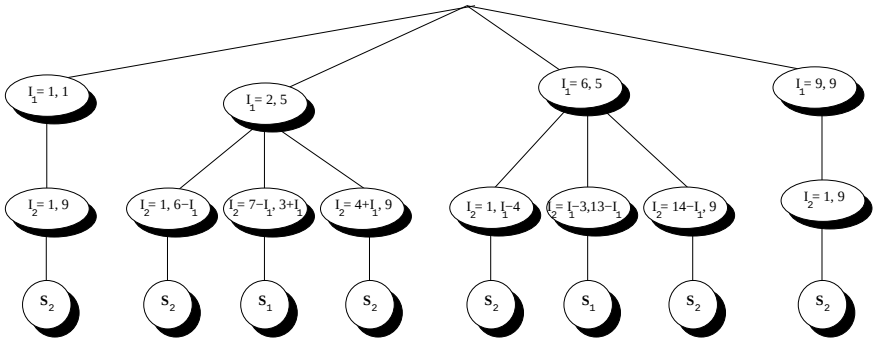

Figure 1: Tree representation of NLS

Now consider the system obtained by forming all inequalities $L \leq U$, for lower-bounds L and upper-bounds U from the previous step, together with all inequalities from the original system not involving $\mathtt{I}_n$. In case the coefficients of $\mathtt{I}_n$ differ from 1, inequalities must be normalized first. After these replacements, a system of inequalities is obtained involving $\mathtt{I}_1, \ldots, \mathtt{I}_{n-1}$ only. Hence we can recursively continue the process, finally ending with a system of inequalities which consist only of the variable $\mathtt{I}_1$ and constants. In this paper we do not elaborate on Fourier-Motzkin elimination further. The reader is referred to [Ban93, BW95a, DE73, LP92] for more details.

Let $[\![\mathcal{C}]\!] = \{x \in \mathbf{R}^n : \varphi_1(x) \wedge \cdots \wedge \varphi_k(x)\}$. We say that $\mathcal{C}$ is *inconsistent* iff $[\![\mathcal{C}]\!] = \emptyset$. We say that an inequality φ is *redundant* for $\mathcal{C}$ iff $[\![\mathcal{C} \cup \{\varphi\}]\!] = [\![\mathcal{C}]\!]$. The Fourier-Motzkin elimination algorithm can be used to decide whether $\mathcal{C}$ is inconsistent. We also have that φ is redundant for $\mathcal{C}$ if and only if $\mathcal{C} \cup \{\neg\varphi\}$ is inconsistent.

Next we define a class of loop structures which will be delivered by the transformations we propose in this paper. This class has been introduced by Chamski [Cha94b].

Definition 2.2 *The class of* nested loop sequences *is inductively defined as the smallest class of loop structures closed under*

1. *Any block of statements is a (trivial) nested loop sequence.*
2. *If $\mathcal{L}$ and $\mathcal{L}'$ are two nested loop sequences, then so is $\mathcal{L}; \mathcal{L}'$ ($\mathcal{L}$ followed by $\mathcal{L}'$).*
3. *If $\mathcal{L}$ is a nested loop sequences, then so is the following loop, where* I *is a new loop variable.*

```
DO  I = L, U
    L
ENDDO
```

A nested loop sequence is called *well-formed* iff all bounds are affine expressions over the loop counters of the surrounding loops. In the sequel we will only consider well-formed nested loop sequences which we will call simply a nested loop sequences, NLS for short.

Note that any perfectly nested loop can be considered as a NLS. Note also that two different NLSs $\mathcal{N}$ and $\mathcal{N}'$ can scan the same iteration space in the same order, executing the same statements in each iteration point. For instance, $\mathcal{N}'$ may be obtained from $\mathcal{N}$ by iteration space partitioning, as is illustrated below with a simple example:

```
N: DO I = 1, 100
     DO J = 1, 50
       S(I,J)
     ENDDO
   ENDDO
```

```
N': DO I = 1, 100
      DO J = 1, 25
        S(I,J)
      ENDDO
      DO J = 26, 50
        S(I,J)
      ENDDO
    ENDDO
```

We call such loop structures $\mathcal{N}$ and $\mathcal{N}'$ *(semantically) equivalent.*

We also need the following notion. Consider a block of statements (which may be a loop) in a nested loop sequence. Then we can construct a perfectly nested loop from the enclosing loops of the block. We call this loop nest the *local nest* of the block. A convenient way of representing nested loop sequences is by means of a finitely branching, finite tree [Cha94b, Cha94a]. We will use these tree structures in our algorithms below. A node in this tree is a tuple $\langle \mathtt{I}, \mathtt{L}, \mathtt{U}, \mathit{body} \rangle$, where

- I is the loop variable;
- L and U are the lower- and upper-bound of the loop, respectively;
- *body* is a list containing (pointers to) representations of the statements in the loop body.

The leaves of the tree contain pointers to (blocks of) assignment statements, IF-statements, etc. Interior nodes contain pointers to representations of nested loop sequences. In figure 1 such a representation for the NLS in figure 5 is given. Using this tree representation of nested loop sequences, a local nest can be represented as a path through this tree.

We call loops (interior nodes in the tree) having the same parent *siblings*. They are loops having the same local nest.

Finally, we need some preliminaries form geometry and linear algebra. A set $H \subseteq \mathbf{R}^d$ consisting of all points represented by the position vector $\vec{x}$ satisfying the linear inequality $\vec{a} \cdot \vec{x} \leq b$ for a fixed nonzero vector $\vec{a} \in \mathbf{R}^d$ and $b \in \mathbf{R}$ is called a *closed half-space* in $\mathbf{R}^d$:

$$H = \{(x_1, \ldots, x_d) \in \mathbf{R}^d | a_1 x_1 + \cdots + a_d x_d \leq b\}$$

Consequently, a half-space consists of all points within and on one side of a hyperplane. For example, in figure 3 we show the half-spaces in $\mathbf{R}$, $\mathbf{R}^2$ and $\mathbf{R}^3$ defined by the inequalities $x_1 \leq 1$, $x_1 + x_2 \leq 3$ and $x_1 + x_2 \leq 3$, respectively. In these cases, hyperplanes $x_1 = 1$, $x_1 + x_2 = 3$ and $x_1 + x_2 = 3$ correspond to a point, line and a plane parallel to the x_3-axis, respectively. In $\mathbf{R}$ and $\mathbf{R}^2$ the corresponding half-space is usually referred to as a half-line and half-plane.

Any set $PS \subseteq \mathbf{R}^d$ consisting of the intersection of a *finite* number of closed half-spaces in $\mathbf{R}^d$ is called a *polyhedral set*. A *bounded* polyhedral set forms a *convex polytope*. Instances of convex polytopes in $\mathbf{R}$, $\mathbf{R}^2$ and $\mathbf{R}^3$ are formed by line segments, convex polygons and convex polyhedra respectively. For example, in figure 2 we show a convex polyhedron formed by the intersection of the half-spaces defined by the inequalities $x_3 \geq 0$, $x_1 \geq 2$, $x_2 \geq 2$ and $x_1 + x_2 + x_3 \leq 8$.

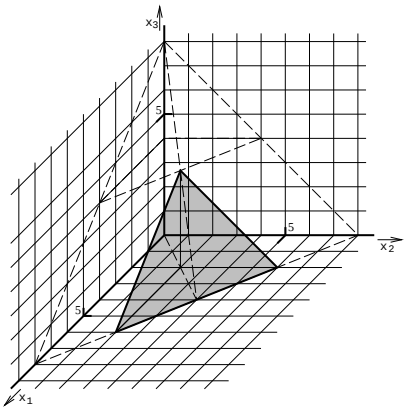

Figure 2: Convex Polyhedron

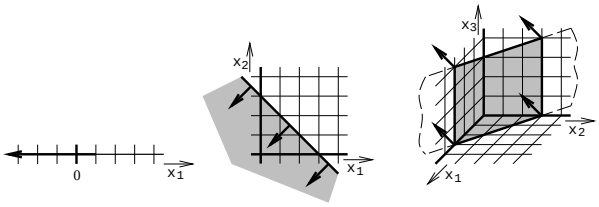

Figure 3: Half-Spaces in $\mathbf{R}$, $\mathbf{R}^2$ and $\mathbf{R}^3$

Obviously, since loops having admissible bounds give rise to systems of linear inequalities, the iteration spaces considered in this paper consist of all the discrete points within a convex polytope.

3 Isolating a polytope inside another polytope

In this section we show how to isolate a polytope $\mathcal{P}'$ defined by a system of linear inequalities arising from a loop $\mathcal{L}'$ inside another polytope $\mathcal{P}$ defined by another loop $\mathcal{L}$. The goal is to generate a loop structure that scans all discrete points in $\mathcal{P}$, but that executes a different block of statements in the intersection of $\mathcal{P}$ and $\mathcal{P}'$ than in the remainder of $\mathcal{P}$. We present an algorithm for computing this loop structure given $\mathcal{L}$ and $\mathcal{L}'$. Together with Fourier-Motzkin elimination, this algorithm forms the backbone of the theory presented in this paper.

We proceed as follows. First we show how to intersect $\mathcal{L}$ with a half-space defined by an inequality of the form $\mathtt{L} \leq \mathtt{I}$, or $\mathtt{I} \leq \mathtt{U}$. The half-space of the first form is called a *lower half-space*, and the half-space of the second form an *upper half-space*. We show for both cases how the isolation of the half-space in $\mathcal{L}$ can be computed. Then we show how a convex bounded polytope, which can be considered as the intersection of a number of half-planes, can be isolated in $\mathcal{L}$.

Assume that we want to execute S_1 for all iterations in the original iteration space outside the half-space, and S_2 for all iterations in the intersection of original iteration space and the given half-space. Then we can generalize index set splitting [BW95b, Wol89, ZC90].

Lower half-spaces Consider a polytope given by $\mathtt{L} \leq \mathtt{I} \leq \mathtt{U}$, and a half-space given by $\mathtt{L}' \leq \mathtt{I}$, where $\mathtt{L}'$ is simple.

Then the following code scans first the part of the polytope which lies outside the half-space, and then the intersection of the polytope and the half-space.

In the first part, we execute a block of statements S_1 and in the second part (the intersection) a block S_2. We call the first part the *pre-loop*, and the second part the *intersection loop*.

```
DO  I = L, min(L' - 1, U)        DO  I = max(L, L'), U
    S1                               S2
ENDDO                            ENDDO
```

Upper half-spaces Likewise, for a half-space given by $\mathtt{I} \leq \mathtt{U}'$ with $\mathtt{U}'$ simple, we generate the following code.

The first loop scans the intersection of the polytope and the half-space, and the second loop scans the remainder of the polytope. In the intersection we execute a block S_2 and in the remainder of the polytope a block S_1. We call the first loop the *intersection loop*, and the second loop the *post-loop*.

```
DO  I = L, min(U, U')            DO  I = max(L, U' + 1), U
    S2                               S1
ENDDO                            ENDDO
```

Since both $\mathtt{L}'$ and $\mathtt{U}'$ are assumed to be simple, the resulting loop bounds are admissible. Please note that irrespective of the position of the half-space and the polytope, the loops as defined above precisely scan the correct portion of the entire space. Note also that some of the loops may be empty. Using these observations we arrive at the following algorithm for isolating a polytope inside another polytope, resulting in a nested loop sequence scanning this space. This algorithm is based on a technique described in [BW95b].

Algorithm
Let $\mathcal{L}$ and $\mathcal{L}'$ be perfectly nested loops, scanning polytopes $\mathcal{P}$ and $\mathcal{P}'$, respectively. Let $\mathcal{L}$ be defined by the system of inequalities

$$\mathtt{L}_1 \leq \mathtt{I}_1 \leq \mathtt{U}_1 \quad \ldots \quad \mathtt{L}_n \leq \mathtt{I}_n \leq \mathtt{U}_n$$

where bounds may be compound. Let $\mathcal{L}'$ be defined by a collection of half-spaces

$$\begin{array}{c} l_1 \leq \mathtt{I}_1, \ldots, l_k \leq \mathtt{I}_1, \mathtt{I}_1 \leq u_1, \ldots, \mathtt{I}_1 \leq u_m \\ \vdots \\ l'_1 \leq \mathtt{I}_n, \ldots, l'_{k'} \leq \mathtt{I}_n, \mathtt{I}_n \leq u'_1, \ldots, \mathtt{I}_n \leq u'_{m'} \end{array}$$

where each bound is simple. Suppose that we want to execute a block of statements S_1 for all discrete points in the intersection of $\mathcal{P}$ and $\mathcal{P}'$, and a block S_2 for the remaining discrete points in $\mathcal{P} - \mathcal{P}'$. We construct a loop structure that performs this task as follows.

For each i from 1 to n do the following.

- Let the *active* loop be $\mathtt{L}_i \leq \mathtt{I}_i \leq \mathtt{U}_i$.
- For each half-space $l \leq \mathtt{I}_i, \ldots, \mathtt{I}_i \leq u$ from $\mathcal{P}'$ do
  - If the half-space is a lower half-space, then partition the active loop accordingly. Set the current active loop to the intersection loop of this partitioning.
  - If the half-space is an upper half-space, then partition the active loop accordingly. Set the current active loop to the intersection loop of this partitioning.
- After this phase we have an ordered list of pre- and post-loops, and an active loop.
  - For each pre- and post-loop, generate as its body the loop defined by the bounds from $\mathcal{L}$

    $$\mathtt{L}_{i+1} \leq \mathtt{I}_{i+1} \leq \mathtt{U}_{i+1}, \ldots, \mathtt{L}_n \leq \mathtt{I}_n \leq \mathtt{U}_n$$

    and body S_2.
  - If $i = n$, then the body of the active loop is given by S_1. Otherwise, its body is given the loop structure obtained in the next iterations over i.

End Algorithm

Example Consider the following loops $\mathcal{L}$ and $\mathcal{L}'$. $\mathcal{L}$ is given by the system of inequalities

$$\begin{cases} 1 \leq \mathtt{I}_1 \leq 9 \\ 1 \leq \mathtt{I}_2 \leq 9 \end{cases}$$

$\mathcal{L}'$ is given by the system of inequalities

$$\begin{cases} 2 \leq & \mathtt{I}_1 & \leq 8 \\ max(7 - \mathtt{I}_1, \mathtt{I}_1 - 3) \leq & \mathtt{I}_2 & \min(\mathtt{I}_1 + 3, 13 - \mathtt{I}_1) \end{cases}$$

The iteration space of $\mathcal{L}$ and $\mathcal{L}'$ is depicted in figure 4.

Then the algorithm from section 3 yields the following pre-, intersection and post-loop, respectively, for $\mathtt{I}_1$:

$$1 \leq \mathtt{I}_1 \leq 1 \quad 2 \leq \mathtt{I}_1 \leq 8 \quad 9 \leq \mathtt{I}_1 \leq 9$$

For the pre- and postloop, the bounds in the second dimension are given by $1 \leq \mathtt{I}_2 \leq 9$. Continuing with the active loop, we obtain the following two pre-, one intersection and two post-loops, respectively, for $\mathtt{I}_2$:

$$\begin{array}{rl} 1 & \leq \mathtt{I}_2 \leq \min(6 - \mathtt{I}_1, 9) \\ \max(7 - \mathtt{I}_1, 1) & \leq \mathtt{I}_2 \leq \\ & \min(\mathtt{I}_1 - 4, 9) \\ \max(7 - \mathtt{I}_1, 1, \mathtt{I}_1 - 3) & \leq \mathtt{I}_2 \leq \\ & \min(3 + \mathtt{I}_1, 9, 13 - \mathtt{I}_1) \\ \max(7 - \mathtt{I}_1, 1, \mathtt{I}_1 - 3, 14 - \mathtt{I}_1) & \leq \mathtt{I}_2 \\ & \leq \min(3 + \mathtt{I}_1, 9) \\ \max(4 + \mathtt{I}_1, 7 - \mathtt{I}_1, 1, \mathtt{I}_1 - 3) & \leq \mathtt{I}_2 \leq 9 \end{array}$$

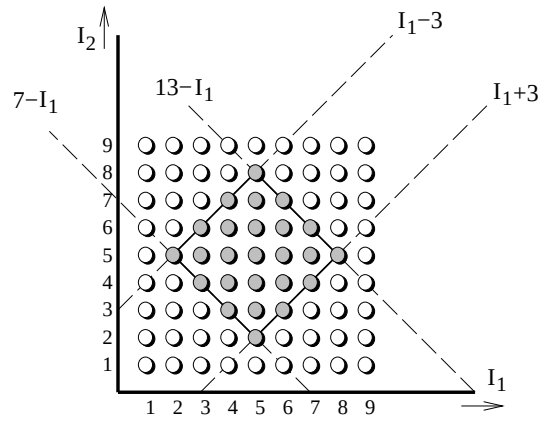

Figure 4: Two intersecting iteration spaces

Please observe that for some values of $\mathtt{I}_1$, some of the above loops over $\mathtt{I}_2$ are empty. For example, for $\mathtt{I}_1 = 2$, the second and the fourth loop are empty. Moreover, a number of bounds are redundant, that is, are always satisfied given the other bounds. For example, since $1 \leq \mathtt{I}_1 \leq 10$, $\min(6 - \mathtt{I}_1, 10) = 6 - \mathtt{I}_1$. In the next section we discuss how to simplify the bound functions and how to remove (partially) empty loops from the structure.

It is clear that the code produced by the algorithm is far from optimal with respect to the overhead associated with the evaluation of loop bounds. It would be possible to generate clean code immediately. However, this would be a costly and difficult procedure (see also the papers by Chamski [Cha94b, Cha94a] where a related technique is discussed). Therefore, we have chosen to generate dirty code, which can be done very inexpensively. This code may be regarded as *intermediate code* that we can clean up afterwards. Obviously, this is a costly operation. Considering that we have to scan a non-convex polytope (namely, $\mathcal{P} - \mathcal{P}'$), this should come as no surprise (*c.f.* [Cha94a]). Nevertheless, the necessary loop structures can all be computed at compile time.

4 Simplifying loop bounds

In this section we discuss techniques for reducing overhead associated with the evaluation of loop bounds. Furthermore, we illustrate each techniques with a simple example.

4.1 Empty loops

Consider a (local) loop nest $\mathcal{L}$, and a loop L in the nest on depth k. Suppose the bounds of this loop are $\mathtt{L}_k$ and $\mathtt{U}_k$, respectively. Then the loop is *empty* iff for no values of $\mathtt{I}_1, \ldots, \mathtt{I}_{k-1}$ within their bounds, the lower-bound of the loop $\mathtt{L}_k$ is smaller than or equal to the upper-bound $\mathtt{U}_k$. This means that the loop is empty iff the following system of inequalities has no integer solutions.

$$\mathtt{L}_1 \leq \mathtt{I}_1 \leq \mathtt{U}_1, \ldots, \mathtt{L}_{k-1} \leq \mathtt{I}_{k-1} \leq \mathtt{U}_{k-1}, \mathtt{L}_k \leq \mathtt{U}_k$$

We can use the Omega test by Pugh [Pug92] to decide whether this system of inequalities has integer solutions. On the other hand, we may check whether the system is inconsistent using Fourier-Motzkin elimination. This provides a conservative test, since inconsistency implies that there are no integer solutions, but not necessarily vice versa.

If the loop L is empty and the nest $\mathcal{L}$ is a stand alone perfectly nested loop, then $\mathcal{L}$ can be removed entirely. Possibly some code needs to be generated for giving the loop counters the correct value after the execution of the loop. For example, since application of Fourier-Motzkin elimination to the system of inequalities $1 \leq \mathtt{I} \leq 10$ and $10 \leq \mathtt{I} - 1$ yields $11 \leq 10$, the original system is inconsistent. Hence, the following perfectly nested loop can be eliminated:

```
DO I = 1, 10
  DO J = 10, I - 1
    ...
  ENDDO
ENDDO
```

If the nest $\mathcal{L}$ is part of some surrounding NLS $\mathcal{N}$, we proceed as follows. Consider the tree representation of $\mathcal{N}$, and the path through this tree that represents the local nest $\mathcal{L}$. The loop L is the kth node on this path. Let k' be the largest number such that $\mathcal{L}$ has siblings in $\mathcal{N}$ on level k', or $k' = 0$ if there are no siblings. Then the path through $\mathcal{N}$ that represents $\mathcal{L}$ can be removed from level k' downwards.

4.2 Redundant bounds

Consider a (local) loop nest $\mathcal{L}$, and a loop L in the nest on depth k. Suppose the bounds of this loop are $\mathtt{L}_k$ and $\mathtt{U}_k$, respectively. Suppose furthermore that the upper-bound $\mathtt{U}_k$ is compound, that is, is of the form $\min(e_1, \ldots, e_m)$ for some simple bounds $e_1, \ldots, e_n$ over the variables $\mathtt{I}_1, \ldots, \mathtt{I}_{k-1}$.

Then the simple bound e_i is *redundant* iff for all values of $\mathtt{I}_1, \ldots, \mathtt{I}_{k-1}$ within their bounds, the value of $e_i(\mathtt{I}_1, \ldots, \mathtt{I}_{k-1})$ is larger than or equal to the values of the other simple bounds. Hence e_i can be removed from $\mathtt{U}_k$, thereby simplifying the evaluation of this bound. We can check whether e_i is redundant by showing that the following system of inequalities has no integer solutions:

$$\mathtt{L}_1 \leq \mathtt{I}_1 \leq \mathtt{U}_1, \ldots, \mathtt{L}_{k-1} \leq \mathtt{I}_{k-1} \leq \mathtt{U}_{k-1}, e_i < \min_{j \neq i} e_j$$

Conservatively, we can check whether the system is inconsistent using Fourier-Motzkin elimination.

Consider, for example, the following loop:

```
DO I = 1, 100
  DO J = 1, MIN(I,100)
    ...
  ENDDO
ENDDO
```

Because the last inequality in the system $1 \leq \mathtt{I} \leq 100$ and $100 < \mathtt{I}$ can be rewritten into $101 \leq \mathtt{I}$, application of Fourier-Motzkin elimination yields the inconsistent system $101 \leq 100$. Hence, the upper bound of the J-loop can be simplified into I.

The case for redundant expressions in lower-bounds is treated analogously. More examples are given in [BW95a].

4.3 Partially empty loops

Consider a (local) loop nest $\mathcal{L}$, and a loop L in the nest on depth k. Suppose the bounds of this loop are $\mathtt{L}_k$ and $\mathtt{U}_k$, respectively. Suppose that L is not empty, but neither that for all values of $\mathtt{I}_1, \ldots, \mathtt{I}_{k-1}$ it is the case that $\mathtt{L}_k \leq \mathtt{U}_k$. In this case, we want to isolate that part of the iteration space defined by $\mathcal{L}$ in which it holds that $\mathtt{L}_k \leq \mathtt{U}_k$. In this subspace, the loop L has to be executed, in the remainder of the iteration space L can be removed. Hence this transformation ensures that the bounds of L are only evaluated in iterations in which L is non-empty.

We can compute the relevant part of the iteration space by solving the following system of inequalities:

$$\mathtt{L}_1 \leq \mathtt{I}_1 \leq \mathtt{U}_1, \ldots, \mathtt{L}_{k-1} \leq \mathtt{I}_{k-1} \leq \mathtt{U}_{k-1}, \mathtt{L}_k \leq \mathtt{U}_k$$

The solution defines a loop $\mathcal{L}'$. We now have two situations.

1. If the nest $\mathcal{L}$ is a stand alone perfectly nested loop, then we may safely replace $\mathcal{L}$ by $\mathcal{L}'$. In this way we have removed spurious iterations for which some inner-loop is empty.

2. If the nest $\mathcal{L}$ is part of a surrounding NLS $\mathcal{N}$, then we may proceed as follows. First, we isolate $\mathcal{L}'$ in $\mathcal{L}$ as described in section 3. We obtain, for each level, a collection of pre- and postloops, and an intersection loop. We have to adapt the algorithm with respect to the code generated in these loops. Note that in the pre- and postloops the loop L is empty and hence these loops should contain code in which L is deleted. In the intersection loop the loop L does contain iterations and hence has to be executed.

   Consider the tree representation of $\mathcal{N}$ and the path leading to L. For each node on this path, let $\mathcal{N}_l$ be the subtree of $\mathcal{N}$ starting with the node on level l of this path. Let $\mathcal{M}_l$ be the tree obtained by deleting L from this subtree.

   We want to generate the following code. Level 1 of $\mathcal{N}$ contains a number of nodes, one of which is the root of $\mathcal{L}$. The isolation algorithm has partitioned this node into a number of other nodes, namely, the collection of preloops, intersection loop, and postloops. The pre- and postloops should be copies of $\mathcal{M}_1$ with the loop bounds on level 1 obtained from the pre- and postloops. Now consider the intersection loop on level 1. On level 2 in this intersection loop, all level 2 siblings of $\mathcal{L}$ in $\mathcal{N}$ should be present, ordered as they are in $\mathcal{N}$.

   The pre- and postloops should be copies of $\mathcal{M}_2$ with bounds obtained from these pre- and postloops, etc. Only on the lowest level the loop L is generated in the intersection loop, together with its siblings. We leave the details to the reader.

Consider, for example, the following perfectly nested loop in which one iteration is empty:

```
DO I = 1, 100
  DO J = 1, I - 1
    ...
  ENDDO
ENDDO
```

Since one step of Fourier-Motzkin elimination to $1 \leq \mathtt{I} \leq 100$ and $1 \leq \mathtt{J} \leq \mathtt{I} - 1$ yields $\max(1,2) \leq \mathtt{I} \leq 100$, the previous loop can be safely rewritten into the following loop:

```
DO I = 2, 100
  DO J = 1, I - 1
    ...
  ENDDO
ENDDO
```

4.4 Compound bounds

Consider a (local) loop nest $\mathcal{L}$, and a loop L in the nest on depth k. Assume that the upper-bound of L is given by a compound bound

$$U = \min(e_1, \ldots, e_m)$$

where each e_i is a simple upper-bound over the index variables $\{\mathtt{I}_1, \ldots, \mathtt{I}_{k-1}\}$. For each $1 \leq i \leq m$, we want to isolate that part of the iteration space of $\mathcal{L}$ in which the value of U is equal to the value of e_i. We obtain this subspace by solving the following system of inequalities:

$$\mathtt{L}_1 \leq \mathtt{I}_1 \leq \mathtt{U}_1, \ldots, \mathtt{L}_{k-1} \leq \mathtt{I}_{k-1} \leq \mathtt{U}_{k-1},$$
$$e_i \leq e_1, \ldots, e_i \leq e_m$$

This gives us a loop $\mathcal{L}'$, in which the upper-bound U of L can be replaced by e_i. We proceed by isolating $\mathcal{L}'$ in $\mathcal{L}$. In the pre- and post-loops we have a version of L with upper-bound

$$U' = \min(e_1, \ldots, e_{i-1}, e_{i+1}, \ldots, e_m)$$

In the active part we have a version of L with upper-bound $U' = e_i$.

For example, this idea enables the following rewriting:

```
DO I = 1, 100                    DO I = 1, 50
 DO J = 1,MIN(I,100-I)            DO J = 1, I
  ...                              ...
 ENDDO                            ENDDO
ENDDO              →             ENDDO
                                 DO I = 51, 100
                                  DO J = 1, 100-I
                                   ...
                                  ENDDO
                                 ENDDO
```

The case for compound lower-bounds containing max functions is treated analogously.

The above procedure constitutes the elementary step for removing min and max functions. Please observe what this step has achieved. First, we have partitioned the iteration space in such a way that in one part the complexity of evaluating a compound upper-bound is reduced to the complexity of evaluating one simple bound. Second, in the remainder of the iteration space, the complexity is reduced to evaluating $n-1$ simple bounds, instead of n.

Please observe also that by using this procedure we may have introduced compound bounds on the levels 1 through $k-1$. This means that we have 'pushed out' the complexity of the bounds to higher levels. Hence, by iterating the above procedure, we can remove all occurrences of compound bounds altogether.

This procedure of 'iterating until no more changes' may be expensive. This reflects that we have, in a certain sense, moved the cost of evaluating compound bounds from run time to compile time. On the other hand, it is easy to see that each step reduces the total cost of evaluating compound bounds, if the kth dimension of the loop has enough iterations. It is easy to see that this number will be small in general. Hence each step reduces the overhead in the computation of the loop bounds. We can easily control the number of times this step may be executed, and still gaining execution time.

We arrive at the following proposition.

Proposition 4.1 *Given a nested loop sequence $\mathcal{N}$, we can construct a semantically equivalent NLS $\mathcal{N}'$ such that $\mathcal{N}'$ contains no zero trip loops, and all bounds in $\mathcal{N}'$ are simple.*

5 Removing affine IF statements

In this section we show how to remove certain kinds of IF-statements from nested loop sequences.

We call IF-statements having affine conditions *affine IF-statements*.

Definition 5.1 *Let $\mathcal{L}$ be a perfectly nested loop with loop indices $\mathtt{I}_1, \ldots, \mathtt{I}_n$.*

1. *A condition of the form $e \leq e'$ where e and e' are affine expressions over $\mathtt{I}_1, \ldots, \mathtt{I}_n$, is called a* simple affine condition *for $\mathcal{L}$.*
2. *The conjunction of one or more simple affine conditions for $\mathcal{L}$ is called an* affine condition *for $\mathcal{L}$.*
3. *An IF-statement of which the condition is an affine condition is called an* affine IF-statement *for $\mathcal{L}$.*

Note that we can express other comparison operations on integers by the following identifications:

$n < m$ iff $n+1 \leq m$; $n = m$ iff $n \leq m$ and $m \leq n$; and $n \geq m$ iff $m \leq n$.

Suppose we have a (local) loop nest containing an affine IF-statement.

```
DO  I_1 = L_1, U_1
    ...
    DO  I_n = L_n, U_n
        IF (e_1 ≤ e'_1 & ··· & e_m ≤ e'_m) THEN
           S_1
        ELSE
           S_2
        ENDIF
    ENDDO
    ...
ENDDO
```

Then the affine condition $(e_1 \leq e'_1 \;\&\cdots\&\; e_m \leq e'_m)$ determines a subspace of the iteration space of this loop. The basic idea is to extract this subspace from the iteration space. In this subspace, the statement S_1 has to be executed. In the rest of the iteration space statement S_2 needs to be executed. We proceed as follows.

Consider the system of inequalities given by the loop bounds and the affine conditions:

$$\mathtt{L}_1 \leq \mathtt{I}_1 \leq \mathtt{U}_1, \ldots, \mathtt{L}_n \leq \mathtt{I}_n \leq \mathtt{U}_n, e_1 \leq e'_1, \ldots, e_m \leq e'_m$$

We have three possibilities.

1. The system is inconsistent. This means that for no value of the loop variables within the loop bounds, the condition holds. Hence we may replace the IF-statement with its ELSE-part without affecting its semantics.

2. The affine condition is redundant with respect to the system of inequalities defined by the loop bounds. This means that for every value of the loop variables within the loop bounds the condition holds. Hence we can replace the IF-statement with its THEN-part without affecting the semantics of the nest.

3. The system is neither inconsistent, nor is the condition redundant. This is the most interesting and probably most frequent case. In this case, the THEN-part of the IF-statement has to be executed in part of the iteration space of the loop, and the ELSE-part in the remainder.

   Using Fourier-Motzkin elimination we can rewrite the system of inequalities given by the loop bounds and the affine conditions to the form

   $$\mathtt{L}'_1 \leq \mathtt{I}_1 \leq \mathtt{U}'_1, \ldots, \mathtt{L}'_n \leq \mathtt{I}_n \leq \mathtt{U}'_n$$

   This system defines a convex polytope within the original iteration space. In this subspace the condition holds. Hence it defines that part of the iteration space in which S_1 has to be executed. We can use the algorithm from section 3 to isolate $\mathcal{L}'$ in $\mathcal{L}$ and hence to isolate the THEN and the ELSE part of the conditional.

Until now we have discussed how to eliminate affine IF-statement in perfectly nested loops. However, the result of this elimination is a nested loop sequence. Hence, if we want to continue the process of eliminating affine IF-statement, we have to extend the theory to the case of nested loop sequences. Briefly, we can apply the following strategy.

- Given an affine IF-statement in a nested loop sequence $\mathcal{N}$, first construct its local nest. This is a perfectly nested loop $\mathcal{L}$.
- Eliminate the affine IF-statement from $\mathcal{L}$ using the above procedure, yielding a nested loop sequence $\mathcal{N}'$.
- Replace the local nest $\mathcal{L}$ in $\mathcal{N}$ by the nested loop sequence $\mathcal{N}'$. Since $\mathcal{N}'$ scans exactly the same iteration space as $\mathcal{L}$ does, this is straightforward: Consider the path through $\mathcal{N}$ that defines $\mathcal{L}$. Each node on this path on level k has to be replaced by the collection of nodes in $\mathcal{N}'$ on level k. Moreover, if a node on the path has children not on the path, these children have to be copied below each of the nodes (from $\mathcal{N}'$) that replace this node.

We arrive at the following proposition.

Proposition 5.2 *Given a nested loop sequence $\mathcal{N}$ containing affine* IF*-statements, we can construct a semantically equivalent nested loop sequence $\mathcal{N}'$ without affine* IF*-statements.*

6 Example

In this section, some of the techniques presented in this paper are illustrated with the following loop $\mathcal{L}$:

```
DO  I_1 = 1, 9
    DO  I_2 = 1, 9
        IF I_2 ≥ max(7 − I_1, I_1 − 3)
           & I_2 ≤ min(I_1 + 3, 13 − I_1) THEN
           S_1
        ELSE
           S_2
        ENDIF
    ENDDO
ENDDO
```

First, we show how we can remove the IF-statement from this loop. That is, we have to isolate the part of the iteration space in which the condition of the IF-statement holds. This gives rise to a loop $\mathcal{L}'$. Using the techniques of section 3, we have that $\mathcal{L}$ and $\mathcal{L}'$ are given by the following systems of inequalities. $\mathcal{L}$ is given by $1 \leq \mathtt{I}_1 \leq 9,\ 1 \leq \mathtt{I}_2 \leq 9$, and $\mathcal{L}'$ is given by $2 \leq \mathtt{I}_1 \leq 8,\ \max(7-\mathtt{I}_1, \mathtt{I}_1-3) \leq \mathtt{I}_2 \leq \min(3+\mathtt{I}_1, 13-\mathtt{I}_1)$.

In the example in section 3 we have shown that isolating $\mathcal{L}'$ in $\mathcal{L}$ gives rise to the following system of inequalities.

$$\begin{array}{rl} 1 & \leq \mathtt{I}_2 \leq \min(6-\mathtt{I}_1, 9) \\ \max(7-\mathtt{I}_1, 1) & \leq \mathtt{I}_2 \leq \\ & \min(\mathtt{I}_1-4, 9) \\ \max(7-\mathtt{I}_1, 1, \mathtt{I}_1-3) & \leq \mathtt{I}_2 \leq \\ & \min(3+\mathtt{I}_1, 9, 13-\mathtt{I}_1) \\ \max(7-\mathtt{I}_1, 1, \mathtt{I}_1-3, 14-\mathtt{I}_1) & \leq \mathtt{I}_2 \\ & \leq \min(3+\mathtt{I}_1, 9) \\ \max(4+\mathtt{I}_1, 7-\mathtt{I}_1, 1, \mathtt{I}_1-3) & \leq \mathtt{I}_2 \leq 9 \end{array}$$

We now show how to clean up the generated structure, which will result in the loop structure given in figure 5.

The body of the pre- and post-loops consists of the block S_2; the body of the intersection loop consists of S_1. We now show how the techniques from sections 4 and 5 can be used to obtain a simple nested loop sequence. First, observe that a number of bounds is redundant. For instance, the upper bound of the first $\mathtt{I}_2$-loop above is redundant.

After removing all redundant bounds, we obtain the following bounds for the intersection loop.

$$\begin{array}{rcl} 1 \leq & \mathtt{I}_1 & \leq 8 \\ \\ 1 \leq & \mathtt{I}_2 & \leq 6-\mathtt{I}_1 \\ \max(7-\mathtt{I}_1, 1) \leq & \mathtt{I}_2 & \leq \mathtt{I}_1-4 \\ \max(7-\mathtt{I}_1, \mathtt{I}_1-3) \leq & \mathtt{I}_2 & \leq \min(3+\mathtt{I}_1, 13-\mathtt{I}_1) \\ 14-\mathtt{I}_1 \leq & \mathtt{I}_2 & \leq \min(3+\mathtt{I}_1, 9) \\ 4+\mathtt{I}_1 \leq & \mathtt{I}_2 & \leq 9 \end{array}$$

Now we remove the max function from the intersection loop. That is, we have to determine the values for $\mathtt{I}_1$ such that

$$\mathtt{I}_1 - 3 \leq 7 - \mathtt{I}_1$$

or, $\mathtt{I}_1 \leq 5$. We obtain the following two loops.

$$\begin{array}{rcl} 2 \leq & \mathtt{I}_1 & \leq 5 \\ \\ 1 \leq & \mathtt{I}_2 & \leq 6-\mathtt{I}_1 \\ \max(7-\mathtt{I}_1, 1) \leq & \mathtt{I}_2 & \leq \mathtt{I}_1-4 \\ 7-\mathtt{I}_1 \leq & \mathtt{I}_2 & \leq \min(3+\mathtt{I}_1, 13-\mathtt{I}_1) \\ 14-\mathtt{I}_1 \leq & \mathtt{I}_2 & \leq \min(3+\mathtt{I}_1, 9) \\ 4+\mathtt{I}_1 \leq & \mathtt{I}_2 & \leq 9 \end{array}$$

and

$$\begin{array}{rcl} 6 \leq & \mathtt{I}_1 & \leq 8 \\ \\ 1 \leq & \mathtt{I}_2 & \leq 6-\mathtt{I}_1 \\ \max(7-\mathtt{I}_1, 1) \leq & \mathtt{I}_2 & \leq \mathtt{I}_1-4 \\ \mathtt{I}_1-3 \leq & \mathtt{I}_2 & \leq \min(3+\mathtt{I}_1, 13-\mathtt{I}_1) \\ 14-\mathtt{I}_1 \leq & \mathtt{I}_2 & \leq \min(3+\mathtt{I}_1, 9) \\ 4+\mathtt{I}_1 \leq & \mathtt{I}_2 & \leq 9 \end{array}$$

Now we can compute that in the first loop structure, the second and fourth loop on $\mathtt{I}_2$ are empty. Likewise, we can remove two empty loops from the second loop structure. After removing redundant bounds, we arrive at the nested loop sequence equivalent to $\mathcal{L}$ depicted in figure 5.

7 Discussion

In this paper we have discussed several techniques for reducing overhead in loops. We have shown that these techniques can restructure a non-trivial loop containing an affine IF-statement into a nested loop sequence with only simple bounds and no IF-statement. This is achieved by partitioning the iteration space of the loop. Note that this process has the drawback that it may cause code explosion. It even may (partially) unroll loops. However, the operations of removing empty loops and redundant bounds can always be applied, since these do not copy code, and even may remove code. So it seems to be a good strategy to always try and remove empty loops after isolating one polytope inside another. Another strategy is given by always trying to transform local loop nests with the largest iteration space and compound bounds or affine IF-statements in their inner loops. In these loops most benefit will be gained by the techniques presented here.

As far as the authors are aware, the only other paper dealing with simplifying loops in the manner presented in this paper, is a paper by Chamski [Cha94b]. In that paper, an algorithm based on the Parametric Integer Programming tool by Feautrier [Fea88] is presented. Like in our approach, his algorithm partitions the iteration space in blocks where the minimum or maximum functions appearing in bounds can be replaced by one of their arguments, and iterating this step until no more changes occur. The advantage of our approach over the approach by Chamski is that we define a number of elementary steps which have to be iterated in order to obtain the full effect of the transformation. Hence the application of the transformation discussed in this paper can easily be controlled in an interactive compilation environment.

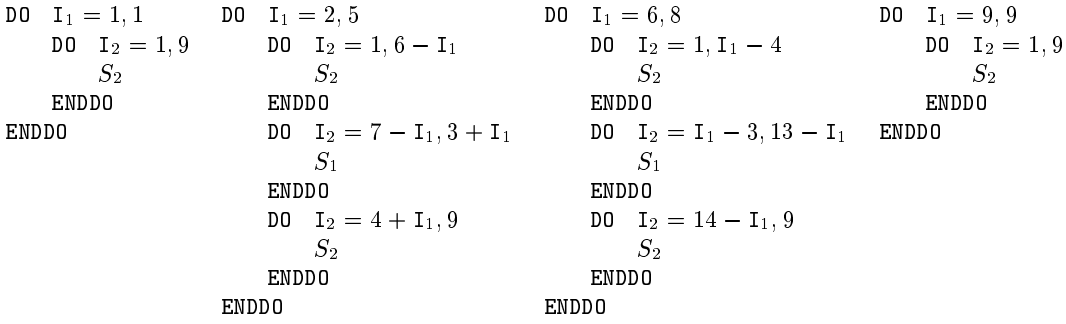

Figure 5: Resulting loop structure

References

[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.* Kluwer Academic Publishers, Norwell, 1993.

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

[BW95a] Aart J.C. Bik and Harry A.G. Wijshoff. Implementation of Fourier-Motzkin elimination. In *Proceedings of the ASCI 95 Conference*, 1995. to appear.

[BW95b] Aart J.C. Bik and Harry A.G. Wijshoff. On strategies for generating sparse codes. Technical Report no 95-01, Department. of Computer Science, Leiden University, 1995.

[Cha94a] Z. Chamski. Enumeration of non-convex sets???? Manuscript, 1994.

[Cha94b] Z. Chamski. Nested loop sequences: Towards efficient loop structures in automatic parallelization. In *Proc. 27th Hawaii Int. Conf. on System Sciences*, pages 14–22, 1994.

[DE73] G.B. Dantzig and B.C. Eaves. Fourier-Motzkin elimination and its dual. *J. of Combinatorial Theory*, 14:288–297, 1973.

[Fea88] P. Feautrier. Parametric integer programming. *Operations Research*, 22(3):243–268, 1988.

[Kuc78] David J. Kuck. *The Structure of Computers and Computations.* John Wiley and Sons, New York, 1978. Volume 1.

[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.

[Pug92] W. Pugh. The Omega test: A fast and practical integer programming algorithm for dependence analysis. *Comm. of the ACM*, 8:102–114, 1992.

[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.

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

[ZC90] H. Zima and B. Chapman. *Supercompilers for Parallel and Vector Computers.* ACM Press, New York, 1990.