Laboratoire de l'Informatique du Parallélisme

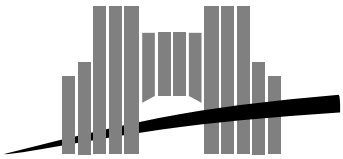
Ecole Normale Supérieure de Lyon
Institut IMAG
Unité de recherche associée au CNRS n°1398

Construction of DO Loops from Systems of Affine Constraints

Jean-François Collard
Paul Feautrier
Tanguy Risset

May 1993

Research Report N^{O} 93-15



Ecole Normale Supérieure de Lyon
46, Allée d'Italie, 69364 Lyon Cedex 07, France,
Téléphone : + 33 72 72 80 00; Télécopieur : + 33 72 72 80 80;
Adresses électroniques :
lip@frensl61.bitnet; lip@lip.ens−lyon.fr (uucp).

Construction of DO Loops from Systems of Affine Constraints

Jean-François Collard
Paul Feautrier
Tanguy Risset

May 1993

Abstract

Most parallelization techniques for `DO` loop nests are based on reindexation. Reindexation yields a new iteration space, which is a convex integer polyhedron defined by a set of affine constraints. Parallel code generation needs thus to scan all the integer points of this convex, thereby requiring the construction of a new `DO` loop nest. We detail an algorithm for this purpose, which relies on a parametrized version of the Dual Simplex. We show how the resulting loop nest and especially the loop bounds can be kept simple and streamlined, so as not to reduce the benefits of parallelization.

Keywords: automatic parallelization, convex integer polyhedron, code generation.

Résumé

La plupart des techniques de parallélisation des nids de boucles `DO` sont basées sur la reindexation. La reindexation génère un nouvel espace d'itération, qui est un polyèdre convexe entier défini par un système d'inéquations affines. La génération du code parallèle nécessite donc d'énumérer tous les points entiers de ce convexe, ce qui s'obtiendra grâce à un nouveau nid de boucles. Nous présentons un algorithme construisant ces boucles, qui repose sur une version paramétrée du Simplexe Dual. Nous montrons comment les boucles générées, et en particulier leurs bornes, peuvent être gardées simples, de façon à obtenir un code efficace où les bénéfices de la parallélisation ne soient pas amoindris.

Mots-clés: parallélisation automatique, polyèdre convexe entier, génération de code.

Construction of DO Loops from Systems of Affine Constraints

Jean-François Collard, Tanguy Risset
LIP
URA CNRS 1398, ENS Lyon
46 Allée d'Italie
F-69364 Lyon Cedex 07
{jfcollar,risset}@lip.ens-lyon.fr

Paul Feautrier
MASI Laboratory
Université de Versailles
45 Avenue des Etats-Unis
F-78035 Versailles Cedex
feautrier@masi.ibp.fr

May 18, 1993

1 Introduction

Parallel computing is one of the most promising way to increase the computational power available to scientists. Recent experiences on parallel machines confirm what has been foreseen: the dramatic need of automation in parallelization [Lam74, Wol89, WL91, ZC90, HKT91, LHS90, SF91, DRR93].

Most parallelization techniques for `DO` loop nests are based on reindexation, i.e. on a change of basis of the iteration space. Since the values of the new loop counters will be enumerated in a `DO` loop nest, innermost loop bounds have to be expressed as functions of the outermost ones. These computations must be exact, so as to preserve the semantics of the original program. Furthermore, they need to be as simple as possible: control overhead should not reduce the benefits of parallelization.

The problem of finding loop bounds of a reindexed loop nest is tightly connected to various problems on integer convex polyhedra [AI91]. Few algorithms have been designed to solve this problem. The Fourier-Motzkin pairwise elimination method is relevant here, but this method generates redundant constraints. An interesting alternative for this particular application is the Simplex method. A parametrized version of the Dual Simplex method has been developed by Feautrier and implemented in the PIP (Parameter Integer Programming [Fea88]) software. Unfortunately, the results produced by the PIP algorithm may sometimes be more intricate than necessary. In this paper, we show that the solution produced by PIP can be simplified.

After setting the notations and recalling general definitions in Section 2, Section 3 illustrates some program transformations we are dealing with in automatic parallelization. The problem of scanning a polyhedron and the loop bounds computation algorithm are explained in Section 4. Section 5

presents the PIP algorithm and gives an extensive example of loop bounds determination, hinting at a simplification possibility. Section 6 formally describes and proves our simplification proposal. The last section summarizes our results and indicates directions for future work.

2 General Definitions

Throughout this paper, arrowed lowercase letters denote "column" vectors with integer or rational entries. The k-th entry of vector $\vec{x} = (x_1, \ldots, x_n)^T$ is denoted by $\vec{x}[k]$ or, if there is no ambiguity, x_k; its subvector built from components k to l is written as: $(x_k, \ldots, x_l)^T$; if $k > l$, then this vector is by convention the vector of dimension 0. Vector concatenation is denoted by $\cdot$, thus for example $\vec{x} = (x_1, \ldots, x_{k-1})^T \cdot (x_k, \ldots, x_n)^T$. The null vector with appropriate dimension is denoted $\vec{0}$. Furthermore, we denote by $\underline{\ll}$ (resp. $\ll$) the lexicographic order (resp. the strict lexicographic order) on such vectors. Finally, if A is a matrix, then $A_{i\bullet}$ represents the i^{th} row of A, and $A_{\bullet j}$ the j^{th} column.

Definition 1 (Structure parameters) *Structure parameters are integer symbolic constants, generally defining array sizes, iteration bounds, etc. Structure parameters may be defined once in the prologue of a program, and may not be modified elsewhere.*

Throughout this paper, the p structure parameters are merged into a p-vector $\vec{z}$.

Definition 2 (Perfect loop nest) *A perfect loop nest is a nest of* `DO` *statements where, for a given counter, lower and upper bounds are affine functions of enclosing loop counters and possibly of structure parameters.*

Perfect loop nests thus follow the scheme below:

```
DO i1 = lb1(z) , ub1(z)
      DO i2 = lb2(i1, z) , ub2(i1, z)
            ⋮
            DO in = lbn(i1, ..., in-1, z) , ubn(i1, ..., in-1, z)
                  S
            ENDDO
      ENDDO
ENDDO
```

Definition 3 (Iteration vector) *For a given statement S in a perfect loop nest, the iteration vector $\vec{x}$ is the vector of the surrounding loop counters.*

For instance, the iteration vector of statement S in the loop nest above is $\vec{x} = (i_1, \ldots, i_n)^T$.

Definition 4 (Operation) *The execution of S for a given iteration vector $\vec{x}$ is called an* operation, *denoted by the pair $(S, \vec{x})$.*

Definition 5 (Iteration space) *The iteration space of a given statement S in a given perfect loop nest is the set of the values taken by its iteration vector when executing the loop nest.*

Since the loop nests we are dealing with are "perfect", the iteration spaces are finite convex polyhedra of $\mathbb{Z}^n$. Such polyhedra can always be defined by a set of inequalities such as:

$$\mathcal{D} = \{\vec{x} \mid \vec{x} \in \mathbb{Z}^n, C\vec{x} + C'\vec{z} + \vec{b} \geq \vec{0}\}$$

where C and C' are *constraint* matrices of size $m \times n$ and $m \times p$ respectively, and $\vec{b}$ is a constant m-vector. This is equivalent to describing a polyhedron as the intersection of a set of half-spaces.

3 Program Transformation

Current parallelizing methods are inherited from automatic systolic design and transform programs through basis changes of iteration spaces. These basis changes are affine or linear, and usually unimodular, transformations. In the case of unimodular linear transformation [BL92b, BL92a], vector coordinates $(y_1, \ldots, y_n)^T$ and $(x_1, \ldots, x_n)^T$, respectively in the old and new bases, are related by:

$$\begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \mathcal{U} \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} \tag{1}$$

where $\mathcal{U}$ is a $n \times n$ matrix such that $\det(\mathcal{U}) = \pm 1$.

The generated code will have to scan the new iteration space, and this will be done using a new `DO`-loop nest. To illustrate the point, consider the following depth-3 perfect loop nest, parametrized by $\vec{z} = (m, n)^T$:

```
DO i = 0 , m
      DO j = 0 , n
            DO k = 0 , i+j
                  S
            ENDDO
      ENDDO
ENDDO
```

The iteration space $\mathcal{D}(m, n)$ is described by:

$$\mathcal{D}(m, n) = \{(i, j, k)^T \mid (i, j, k) \in \mathbb{Z}^3, 0 \leq i \leq m, 0 \leq j \leq n, 0 \leq k \leq i + j\}$$

This set is a convex polyhedron in $\mathbb{Z}^3$, which can be written as:

$$\mathcal{D}(m,n) = \{\vec{x} \mid \vec{x} \in \mathbb{Z}^3, C\vec{x} + C'\vec{z} + \vec{b} \geq \vec{0}\} \quad (2)$$

with:

$$\vec{x} = \begin{pmatrix} i \\ j \\ k \end{pmatrix} \qquad C = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \\ 1 & 1 & -1 \end{pmatrix} \qquad C' = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix} \qquad \vec{b} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

We give below two examples of such basis changes:

Example 1 : Loop interchange. Assume data dependencies allow the loop nest above to be transformed using loop interchange while preserving its semantics. This transformation would rewrite the loop with counters in the order k, j, i instead of i, j, k:

```
DO  k= lb_k(m,n), ub_k(m,n)
      DO  j= lb_j(m,n,k), ub_j(m,n,k)
            DO  i= lb_i(m,n,k,j), ub_i(m,n,k,j)
                  S
            ENDDO
      ENDDO
ENDDO
```

In this case, the corresponding matrix $\mathcal{U}$ is:

$$\mathcal{U} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

Example 2 : the "systolic" point of view. The goal of the "systolic" method is to find a new iteration space basis in which the first coordinate(s) represent the execution time or *schedule* of the operations[1] and the remaining coordinate(s) represent the virtual processor address on which this operation is to be executed. In this new nest, the loops on time have to be executed sequentially, but not the loops on space. The loop above can for example be eventually expressed with respect to counters $(t, x, y)^T$, where t is the schedule and (x, y) the coordinates of virtual processors on a two-dimensional grid.

The code generated by such a transformation will look like:

[1] To see why this time may be multidimensional, see [Fea92b].

```
DO   t= lb_t(m,n), ub_t(m,n)
       DOALL   (x= lb_x(m,n,t) : ub_x(m,n,t))
                 DOALL   (y= lb_y(m,n,t,x) : ub_y(m,n,t,x))
                           S
                 ENDDOALL
       ENDDOALL
ENDDO
```

Methods of dependence analysis [Fea89, Fea91] and schedule computation [Fea92a, Fea92b, QR89, Qui87] are now well understood and provide an appropriate matrix $\mathcal{U}$.

In the transformation examples above, the parallelizer has to compute new loop bounds, and this is all but obvious. The rest of this paper concentrates on the algorithm for this computation and on the application of the PIP algorithm to this purpose.

4 The Polyhedron Scanning Problem

Our motivation is to show how loop bounds can be computed algorithmically from systems of affine constraints. The DO loop nest obtained will scan once and only once all the integer points of the corresponding polyhedron.

4.1 The Input

The input of our computation is a polyhedron $\mathcal{D}(\vec{z})$ expressed in the original basis as a set of m inequations:

$$\mathcal{D}(\vec{z}) = \{\vec{x} \mid C\vec{x} + C'\vec{z} + \vec{b} \geq \vec{0}\} \tag{3}$$

where C and C' are *constraint* matrices of size $m \times n$ and $m \times p$ respectively, and $\vec{b}$ is a constant m-vector.

This expression of polyhedron $\mathcal{D}(\vec{z})$ is obtained by parsing the program text and writing two inequalities $lb_k \leq i_k \leq ub_k$ per loop. Observe that one may still construct $\mathcal{D}(\vec{z})$ if the lower (resp. upper) bound involves ceiling and maximum (resp. floor and minimum) operators. In that case there will be more inequalities than in the simple case.

4.2 The Output

The result must be a loop nest L:

```
DO   x_1 = a_1, b_1
       ⋮
       DO   x_n = a_n, b_n
              S
       ENDDO
ENDDO
```

whose iteration vector is $\vec{x}$ in the new basis, and which enumerates the integer points in $\mathcal{D}(\vec{z})$ in lexicographic order with respect to the new coordinate system. We have to decide whether $\mathcal{D}(\vec{z})$ if empty and, if not, generate loop `L`. `L` is not a perfect loop nest as defined in Section 2: for a given counter x_k, both lower and upper bounds follow the requirements below:

R1 : Bounds may be affine functions of $x_1, \ldots, x_{k-1}$ and $\vec{z}$,

R2 : Bounds may be ceiling/floor functions of R1 expressions,

R3 : Bounds may be maxima/minima of R1 or R2 expressions.

Thus, the output is a list of bounds: first, the bounds of the outermost counter x_1, then the bounds on x_2 as a function of the structure parameters plus x_1, and so on for all entries in $\vec{x}$.

4.3 The Algorithm

The basic method has been presented in [Fea89]. The first question is to know how one can exactly scan the integer points of $\mathcal{D}(\vec{z})$, that is, to build the `L` loop nest. The method proceeds by constructing n polyhedra:

$$\mathcal{D}_k(x_1 \ldots x_k, \vec{z}) = \{(x_{k+1} \ldots x_n)^T \mid C\vec{x} + C'\vec{z} + \vec{b} \geq \vec{0}\}$$

for $k = 0, n-1$. Obviously, $\mathcal{D}(\vec{z}) = \mathcal{D}_0(\vec{z})$. Let:

$$\vec{l}_k(x_1 \ldots x_{k-1}, \vec{z}) = \min_{\ll} \mathcal{D}_{k-1}(x_1 \ldots x_{k-1}, \vec{z}), \tag{4}$$

$$\vec{u}_k(x_1 \ldots x_{k-1}, \vec{z}) = \max_{\ll} \mathcal{D}_{k-1}(x_1 \ldots x_{k-1}, \vec{z}). \tag{5}$$

Note that we use PIP to compute these lexicographic extrema, but the algorithm given here is independant of PIP.

We assert that the bounds of the k-th loop are:

$$a_k = \left\lceil \vec{l}_k(x_1 \ldots x_{k-1}, \vec{z})[1] \right\rceil,$$

$$b_k = \left\lfloor \vec{u}_k(x_1 \ldots x_{k-1}, \vec{z})[1] \right\rfloor.$$

Proof Let $\vec{x}$ be an integer point in $\mathcal{D}(\vec{z})$. For all k, $(x_k \ldots x_n)^T$ is an element of $\mathcal{D}_{k-1}(x_1 \ldots x_{k-1}, \vec{z})$. As a consequence,

$$\vec{l}_k(x_1 \ldots x_{k-1}, \vec{z}) \underline{\ll} (x_k \ldots x_n)^T \underline{\ll} \vec{u}_k(x_1 \ldots x_{k-1}, \vec{z}),$$

which implies:

$$\vec{l}_k(x_1 \ldots x_{k-1}, \vec{z})[1] \leq x_k \leq \vec{u}_k(x_1 \ldots x_{k-1}, \vec{z})[1],$$

which says in effect that x_k is within the bounds of the k-th loop, by the properties of the floor and ceiling functions.

Conversely, let $\vec{x}$ be an iteration vector of the loop nest. $\mathcal{D}_{n-1}(x_1 \ldots x_{n-1}, \vec{z})$ is a one dimensional polyhedra. Obviously:

$$\vec{l}_n(x_1 \ldots x_{n-1}, \vec{z}) \leq x_n \leq \vec{u}_n(x_1 \ldots x_{n-1}, \vec{z}),$$

which implies, by convexity, that $x_n \in \mathcal{D}_{n-1}(x_1 \ldots x_{n-1}, \vec{z})$, which is equivalent to saying that $\vec{x} \in \mathcal{D}(\vec{z})$.

As a consequence, we have proved that the loop nest L scans all integer points of $\mathcal{D}(\vec{z})$ and nothing but integer points of $\mathcal{D}(\vec{z})$. □

We may in fact prove a more precise result. The result above would still be true if we enlarged arbitrarily the bounds of all loops but the last one. The effect would be that most of the time, the bounds for the x_n loop would be undefined, the end result being that only the proper iterations would be executed. The scheme above does not suffer from this waste of processing power:

Let $\mathcal{P}_k(\vec{z})$ be the projection of $\mathcal{D}(\vec{z})$ on the space of the first k loop counters:

$$\mathcal{P}_k(\vec{z}) = \{(x_1 \ldots x_k)^T \mid \exists x_{k+1} \ldots x_n : C\vec{x} + C'\vec{z} + \vec{b} \geq \vec{0}\}.$$

It is well known that all $\mathcal{P}_k(\vec{z})$ are convex polyhedra. Obviously, $\mathcal{P}_n(\vec{z}) = \mathcal{D}(\vec{z})$. We assert that the first k loops of L scan $\mathcal{P}_k(\vec{z})$.

Proof The proof that all points of $\mathcal{P}_k(\vec{z})$ are visited by the loop nest is exactly the same as above. The proof of the reciprocal is by recurrence on k. The property is obvious for $k = 1$, since $\mathcal{P}_1(\vec{z})$ is one dimensional and convex. Suppose the property is true up to $k-1$. Let $(x_1 \ldots x_k)^T$ be a valid iteration vector. By definition of a loop nest, so is $(x_1 \ldots x_{k-1})^T$. By the induction hypothesis,

$$(x_1 \ldots x_{k-1})^T \in \mathcal{P}_{k-1}(\vec{z}).$$

As a consequence, $\mathcal{D}_{k-1}(\vec{z})$ is not empty and $\vec{l}_k(x_1 \ldots x_{k-1}, \vec{z})$ and $\vec{u}_k(x_1 \ldots x_{k-1}, \vec{z})$ are well defined. Both

$$(x_1 \ldots x_{k-1})^T \cdot \vec{l}_k(x_1 \ldots x_{k-1})$$

and

$$(x_1 \ldots x_{k-1})^T \cdot \vec{u}_k(x_1 \ldots x_{k-1})$$

belong to $\mathcal{D}(\vec{z})$, and so does, by convexity, all

$$\begin{aligned} \vec{y}(\mu) &= \mu[(x_1 \ldots x_{k-1})^T \cdot \vec{l}_k(x_1 \ldots x_{k-1})] \\ &\quad +(1-\mu)[(x_1 \ldots x_{k-1})^T \cdot \vec{u}_k(x_1 \ldots x_{k-1})] \end{aligned}$$

for $0 \leq \mu \leq 1$. Now, since:

$$\vec{l}_k(x_1 \ldots x_{k-1})[1] \leq x_k \leq \vec{u}_k(x_1 \ldots x_{k-1})[1],$$

there exists a value of μ such that:

$$\vec{y}(\mu)[k] = x_k.$$

Since $\vec{y}(\mu) \in \mathcal{D}(\vec{z})$, we have proved that $(x_1 \ldots x_k)^T \in \mathcal{P}_k(\vec{z})$. □

4.4 The example, revisited

This section illustrates the use of the Parametric Integer Programming Algorithm (PIP) to get new loop bounds after loop transformation. In the example (Section 3), the first step consists in finding the lexical extrema of:

$$\mathcal{D}_0((m,n)) = \{(k,j,i)^T | i \leq m, j \leq n, k \leq i+j\},$$

where m and n are structure parameters. For this first problem, structure parameters are supposed to be non-negative. PIP has to be called twice (once for upper bounds and once for lower bounds, in any order). The result is:

$$\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \lll \begin{pmatrix} k \\ j \\ i \end{pmatrix} \lll \begin{pmatrix} m+n \\ n \\ m \end{pmatrix}. \tag{6}$$

From the definition of lexical ordering, we know that this entails:

$$0 \leq k \leq m+n.$$

We cannot, however, deduce anything about i or j from the above result.

The second problem is:

$$\mathcal{D}_1(k,(m,n)) = \{(j,i)^T | i \leq m, j \leq n, k \leq i+j\}.$$

This yields the following bounds on j:

$$\left.\begin{array}{l} \textbf{if } k-m-1 \geq 0 \\ \textbf{then if } k-m-n-1 \geq 0 \\ \qquad\qquad\qquad\qquad\qquad \textbf{then } \bot \\ \qquad\qquad\qquad\qquad\qquad \textbf{else } k-m \\ \textbf{else } 0 \end{array}\right\} \leq j \tag{7}$$

and:

$$j \leq (\textbf{if } (k-m-n-1 \geq 0) \textbf{ then } \bot \textbf{ else } n). \tag{8}$$

The bound of j in (7) is called a *quast* (for quasi affine solution tree). These expressions are very intricate because, as we have said earlier, they include information both on the j loop and on the k loop. They can be simplified by noticing that $k \leq m+n$ implies that $k-m-n-1 \geq 0$ is never true. PIP has a mechanism for doing that. The *context* is a set of constraints on the parameters, which are used to simplify (if possible) expressions such as (7) above. If we have $k \leq m+n$ as context, the new result is:

$$(\textbf{if } (m-k \geq 0) \textbf{ then } 0 \textbf{ else } k-m) \leq j \leq n. \tag{9}$$

The last problem is:

$$\mathcal{D}_2(j,k,(m,n)) = \{i | i \leq m, j \leq n, k \leq i+j\}.$$

We may use $k \leq m + n$ and $j \leq n$ as context, but we cannot use the lower bound of j in (9) since it is not in the format of a linear constraint. The solution for i is then:

$$\left.\begin{array}{l} \textbf{if } \ j - k \geq 0 \\ \textbf{then } \ 0 \\ \textbf{else if } \ m - k + j \geq 0 \\ \qquad\qquad\qquad\qquad \textbf{then } \ k - j \\ \qquad\qquad\qquad\qquad \textbf{else } \ \bot) \end{array}\right\} \leq i \leq \left\{\begin{array}{l} \textbf{if } \ m - k + j \geq 0 \\ \textbf{then } \ m \\ \textbf{else } \ \bot \end{array}\right. \tag{10}$$

Had our program been smarter, it could have noticed that (7) is equivalent to:

$$\max(0, k - m) \leq j \tag{11}$$

which implies $j \geq 0$ and $j \geq k - m$. Using these two constraints as context would have given:

$$\left.\begin{array}{l} \textbf{if } \ j - k \geq 0 \\ \textbf{then } \ 0 \\ \textbf{else } \ k - j \end{array}\right\} \leq i \leq m \tag{12}$$

There again, the lower bound is $\max(0, k - j)$. Is this a general property? From the run-time point of view, one may say that the computation of the `max` in (11) is no easier than computation of the `if` in (7). That is true, but the point is not computation time but useless bound intricacy, such as for i in (10).

However, the validity of such a transformation is by no means obvious. In (7), the two inequations were:

$$j \geq 0 \quad ; \quad j \geq k - m \tag{13}$$

These inequations were valid on very specific domains (respectively $m - k \geq 0$ and $m - k < 0$). Extension to the entire space remains to be justified. The intuition is that this extension is valid for convex domains; according to this intuition, a convex polyhedron is defined by a system of inequalities which is satisfied by every point included in it. Thus, every point (k, j) is such that $j \geq 0 \wedge j \geq k - m$, which implies that $j \geq \max(0\ ,\ k - m)$ (see Fig. 1 where the two equations of (13) are drawn).

However, one should note how error-prone this intuition may be. Suppose the lower bound on j had been defined by (see Fig.2):

$$j \geq \left\{\begin{array}{l} \textbf{if } (m - k \geq 0) \\ \textbf{then if } (k - m \geq 0) \\ \qquad\qquad\qquad\qquad \textbf{then } 1 - k \\ \qquad\qquad\qquad\qquad \textbf{else } k - m \\ \textbf{else } 0 \end{array}\right. \tag{14}$$

This is equivalent to the following three equations, with their respective domains:

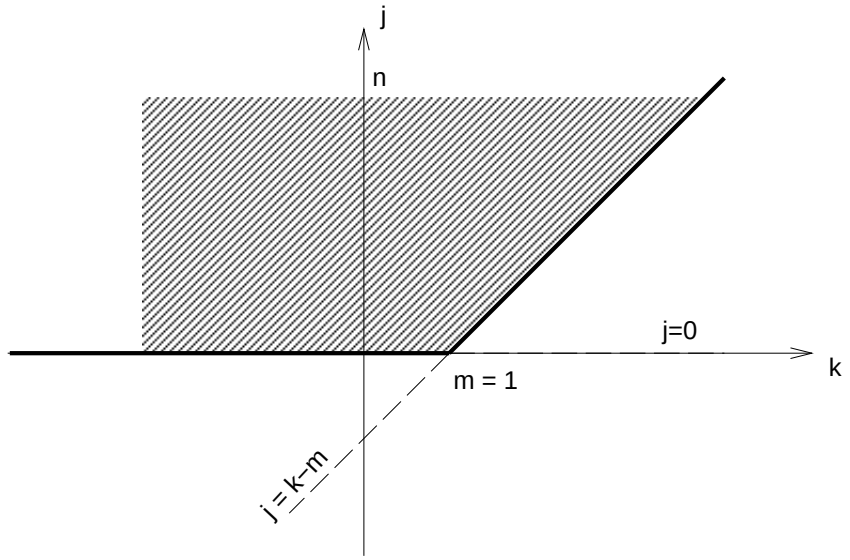


Figure 1: One can take the maximum of the two equations...

$$\begin{cases} m-k \geq 0 \ \wedge \ k-m \geq 0 & : \quad j = 1-k \\ m-k \geq 0 \ \wedge \ k-m < 0 & : \quad j = k-m \\ m-k < 0 & : \quad j = 0 \end{cases}$$

Although this quast is perfectly correct since it describes the same lower bound as (7), the first equation on j ($j = 1-k$) implies that the lower bound can't be simplified as $\max(0, k-m, 1-k)$. In fact, this third equation is unsuitable because the corresponding line cuts through the polyhedron. In the terminology of [Sch86], the hyperplane $\{j | j = 1-k\}$ is not a supporting plane of the polyhedron. The key remark is that the solutions given by PIP will never cut the polyhedron.

5 Parametric Integer Programming

Our problem is now to compute the lexicographical minimum in (4) or maximum in (5). The problems are easily seen to be linear programming problems with two differences:

- the cost function is the lexicographic order;
- parameter values are generally unknown.

PIP handles these two requirements[2]. We will briefly describe the PIP algorithm, but the interested reader will find a more complete description in [Fea88].

[2]PIP furthermore handles integer problems, thanks to the Gomory algorithm [Gom63]. This part of the algorithm is not described here.

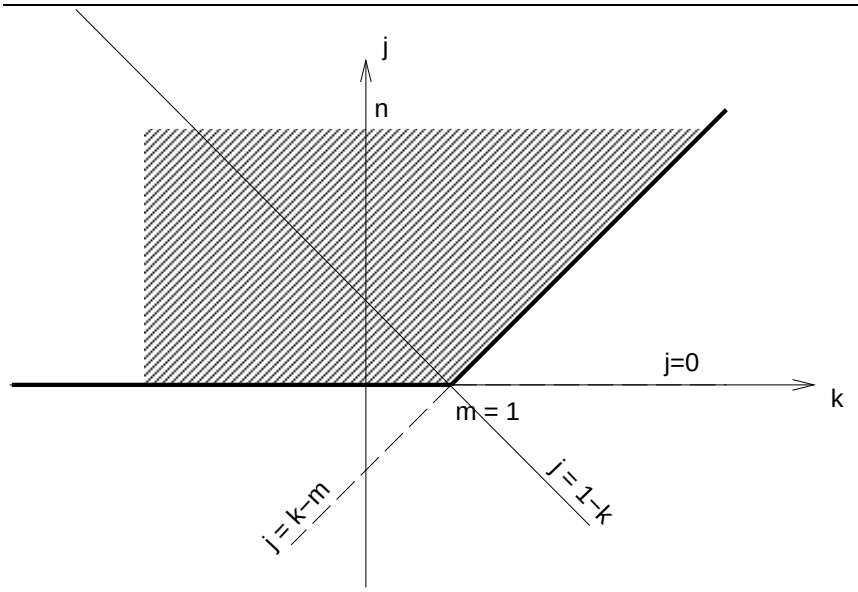


Figure 2: ...but not the maximum of these three.

Suppose that we want the lexicographic minimum[3] of $\mathcal{D}(\vec{z})$. $\vec{z}$ is the vector of structure parameters, submited to the context conditions. $\mathcal{D}(\vec{z})$ is a convex polyhedron whose points are non negative, which can be described as:

$$\mathcal{D}(\vec{z}) = \{\vec{x} \mid C\vec{x} + C'\vec{z} + \vec{c} \geq \vec{0}, \vec{x} \geq \vec{0}\} \tag{15}$$

5.1 PIP Algorithm

In the Dual Simplex algorithm, the problem is solved by a succession of variable changes until we reach a *stopping* condition (to be specified later). Each iteration l changes $\vec{x}^l$ into $\vec{x}^{l+1}$ by an affine invertible transformation[4]. Since this transformation is invertible, we will express $\vec{x}^l$ as a function of $\vec{x}^{l+1}$ so as to be able to replace $\vec{x}^l$ by $\vec{x}^{l+1}$ in (15). Furthermore, since this transformation itself will change from iteration to iteration, it should also be subscripted with l. We can sum this up with the following relation:

$$\vec{x}^l = A^l\vec{x}^{l+1} + A'^l\vec{z} + \vec{a}^l \tag{16}$$

where A^l and A'^l are $|\vec{x}| \times |\vec{x}|$- and $|\vec{x}| \times |\vec{z}|$-matrices respectively, and $\vec{a}^l$ is an $|\vec{x}|$-vector. Initially, A^0 and A'^0 are respectively the unit- and zero-matrices, $\vec{a}^0$ is $\vec{0}$ and $\vec{x}^0 = \vec{x}$. Moreover one should note that this basis change is such that:

[3] Maximum problems can be transformed into minimum ones by a simple trick which is described in [Fea88].

[4] Superscripts such as $\vec{x}^l$ always denote the l^{th} element of a sequence. This convention does not interfere with the power function since the latter is never used in this paper.

- $\vec{x}^l \in \mathcal{D}(\vec{z}) \Rightarrow \vec{x}^{l+1} \geq \vec{0}$;
- $\vec{x}^l$ and $\vec{x}^{l+1}$ differ in only one entry, because each Parametric Dual Simplex Algorithm iteration step changes one and only one basis variable.

Thus $\mathcal{D}(\vec{z})$ is:

$$\mathcal{D}(\vec{z}) = \{A^l\vec{x}^{l+1} + A'^l\vec{z} + \vec{a}^l \left| \begin{array}{l} A^l\vec{x}^{l+1} + A'^l\vec{z} + \vec{a}^l \geq \vec{0} \\ C^l\vec{x}^{l+1} + C'^l\vec{z} + \vec{c}^l \geq \vec{0} \end{array} \right\} \tag{17}$$

where C^l and C'^l are the $m\times|\vec{x}|$- and $m\times|\vec{z}|$-matrices derived from C and C' at iteration l. We may consider the $m+|\vec{x}|$-vector:

$$\vec{t}^l(\vec{z}) = \begin{pmatrix} A'^l \\ C'^l \end{pmatrix}\vec{z} + \begin{pmatrix} \vec{a}^l \\ \vec{c}^l \end{pmatrix} \tag{18}$$

and A^l and C^l as two blocks of an $(m+|\vec{x}|)\times|\vec{x}|$ matrix S^l:

$$S^l = \begin{pmatrix} A^l \\ C^l \end{pmatrix} \tag{19}$$

$[S^l, \vec{t}^l(\vec{z})]$ is the *problem tableau* in the sense of the simplex method. A step of PIP's algorithm is described as follows.

1. Determine the signs of the components of $\vec{t}^l(\vec{z})$.
2. If all components are positive, then the solution is found.
3. If there is a negative component $\vec{t}^l(\vec{z})[i]$, then two cases may occur.
   - If all entries in the i-th row of S^l are negative, then the solution does not exist.
   - If there is at least a positive entry S^l_{ij}, then it gives rise to a pivoting step and thus to iteration $l+1$. The new problem tableau is $[S^{l+1}, \vec{t}^{l+1}(\vec{z})]$ where:

$$\begin{aligned} S^{l+1}_{\bullet j} &= (1/S^l_{ij})S^l_{\bullet j} \\ S^{l+1}_{\bullet k} &= S^l_{\bullet k} - (S^l_{ik}/S_{ij})S^l_{\bullet j},\ k \neq j \\ \vec{t}^{l+1}(\vec{z}) &= \vec{t}^l(\vec{z}) - (t^l_i(\vec{z})/S^l_{ij})S_{\bullet j} \end{aligned}$$

4. In the remaining case, select a $\vec{t}^l(\vec{z})[i]$ whose sign is unknown. This yields **two** new subproblems, according to the signs. Both yield an $(l+1)$-th iteration.

This separation has a crucial consequence: there won't be one and only one solution, but a binary tree whose nodes are predicates with respect to the parameters and whose leaves are either an affine function if a solution exists, or, if not, the undefined solution written as $\perp$. Since each node produces two subproblems, one has to remember the path taken and take account of

the corresponding predicates. This is called the problem *context*. For $l = 0$, this context is a set of inequalities on the parameters; when point 4 of the step algorithm above is reached, the first subproblem must be computed in the previous context augmented with:

$$\vec{t}(\vec{z})[i] \geq \vec{0}$$

Symmetrically, the second context is the parent's one plus:

$$\vec{t}(\vec{z})[i] < \vec{0}$$

In the end, each path in the tree is exactly what the actual simplex algorithm would have done, but here the solution $Sol(\vec{z})$ is expressed as a function of the parameters. The entire binary tree is called a *quasi-affine solution tree* or *quast*, whose leaves are thus associated with a *domain* Dom_i defined by the conjunction of the predicates in the corresponding path. These domains build up a partition of the problem space, since no point can belong to two different domains (this would imply that one predicate in the quast is both true and false), and since every predicate cuts the problem space in two disjoint half-spaces. Moreover, since the quast leaves are affine functions in the variables, they define hyperplanes in the $\mathbb{Z}^p$ space for a given parameter vector $\vec{z}$. A quast can thus be regarded as a multiguard:

$$Sol(\vec{z}) = \begin{cases} (\vec{z}) \in Dom_1 : \vec{f^1}(\vec{z}) \\ \vdots \\ (\vec{z}) \in Dom_q : \vec{f^q}(\vec{z}) \end{cases}$$

In our particular case, we are only interested in the first component of the solution, since it gives us the bound of our loop counter. Thus, we obtain:

$$\begin{pmatrix} (\vec{z}) \in Dom_1 : \vec{f^1}(\vec{z})[1] \\ \vdots \\ (\vec{z}) \in Dom_q : \vec{f^q}(\vec{z})[1] \end{pmatrix} \quad \leq \quad x_1$$

5.2 Understanding PIP's behavior

The aim of this section is to show how the solution (7) in the example of section 4.4 is found and thus to get an intuition of where PIP's solution comes from. We will start at the $\mathcal{D}_1$ problem : i, j, a, b, and c are the variables associated to $\mathcal{D}_1$'s inequations:

$$\begin{aligned} i &\geq 0 \\ j &\geq 0 \\ a = m - i &\geq 0 \\ b = n - j &\geq 0 \\ c = i + j - k &\geq 0 \end{aligned}$$

The initial Simplex tableau is Table 1, where rows are labeled with their respective variables.

	j	i	1	k	m	n
j	1	0	0	0	0	0
i	0	1	0	0	0	0
$a = m - i$	0	-1	0	0	1	0
$b = n - j$	-1	0	0	0	0	1
$c = i + j - k$	1	1	0	-1	0	0

Table 1: First tableau

According to the notations in (18) and (19), we have:

$$A_0 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$C_0 = \begin{pmatrix} 0 & -1 \\ -1 & 0 \\ 1 & 1 \end{pmatrix} \quad C'_0 = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -1 & 0 & 0 \end{pmatrix}$$

Since the bounds on k are known, the solution of $\mathcal{D}_1$ has to be computed in context (6). All rows have non-negative constant terms excepted the c row, whose constant term $(\vec{t}(\vec{z}))[c]$ is $-1 \times k$. A pivoting step is necessary with entry (c, i) as the pivot. Variable c enters the basis and i leaves it, giving Table 2 as the new problem tableau.

Now, all rows have non negative constant terms excepted the a row, whose constant term $(\vec{t}(\vec{z}))[a]$ is $-1 \times k + 1 \times m = m - k$. If $m - k \geq 0$, then the optimum is reached. The basis variables are j and c, whose values are 0 as it can be read in Table 2 in their respective rows. Moreover,

$$c = 0 \wedge j = 0 \Rightarrow i = k$$

and the tentative result is:

$$i = k \quad , \quad j = 0$$

If $m - k < 0$, then another pivoting step is necessary. Entry (a, j) is the new pivot, and variable a enters the basis and j leaves it. The new tableau is the one in Table 3.

	j	c	1	k	m	n
j	1	0	0	0	0	0
i	-1	1	0	1	0	0
a	1	-1	0	-1	1	0
b	-1	0	0	0	0	1
c	0	1	0	0	0	0

Table 2: Second tableau

	a	c	1	k	m	n
j	1	1	0	1	-1	0
i	-1	0	0	0	1	0
a	1	0	0	0	0	0
b	-1	-1	0	-1	1	1
c	0	1	0	0	0	0

Table 3: Third tableau

In the a row, the term $m - k$ is necessarily positive. Basis variables are a and c, and their values are both 0:

$$a = 0 \wedge c = 0 \Rightarrow i = m \wedge j = k - m,$$

The final lower bound on j is thus:

$$(\textbf{if} \quad (m - k \geq 0) \quad \textbf{then} \quad 0 \quad \textbf{else} \quad k - m) \leq j \tag{20}$$

We thus have an illustration of how the Parametric Dual Simplex Algorithm algorithm finds a solution: through a sequence of basis changes, it tests a sequence of trial solution in ascending lexicographic order, until a feasible point is found. The next section will try and explain how this can be used for our purposes.

6 Simplification of the result

As we have stated in section 4.4, we would like to replace the solution given by PIP (a quast) by a maximum upon the quast's leaves. In this section, we prove that this can be done safely.

6.1 An example

Consider the following example:

$$\begin{cases} \text{in context } \{z \geq 0\} \\ \text{get the lexicographic minimum of} \\ \mathcal{D} = \{i \mid i - z \geq 0\} \end{cases}$$

The hatched zone in figure 3 represents the domain as a convex polyhedron of space $(\vec{z}, \vec{i})$. Suppose that, for some reason, one step of the algorithm has to solve the following problem:

$$\begin{cases} \text{in context } \{z \geq 0, z - 1 \geq 0, -z + 1 \geq 0\} & \text{/* thus } z = 1 \text{ */} \\ \text{get the lexicographic minimum of} & \\ \mathcal{D} = \{i \mid i - z \geq 0\} & \text{/* line } d \text{ on Figure 3 */} \end{cases}$$

We can see that the following solution is correct:

$$\begin{cases} \text{in context } \{z = 1\} \\ \text{the lexicographic minimum of } \mathcal{D} = \{i \mid i - z \geq 0\} \text{ is } i = 1 \end{cases} \tag{21}$$

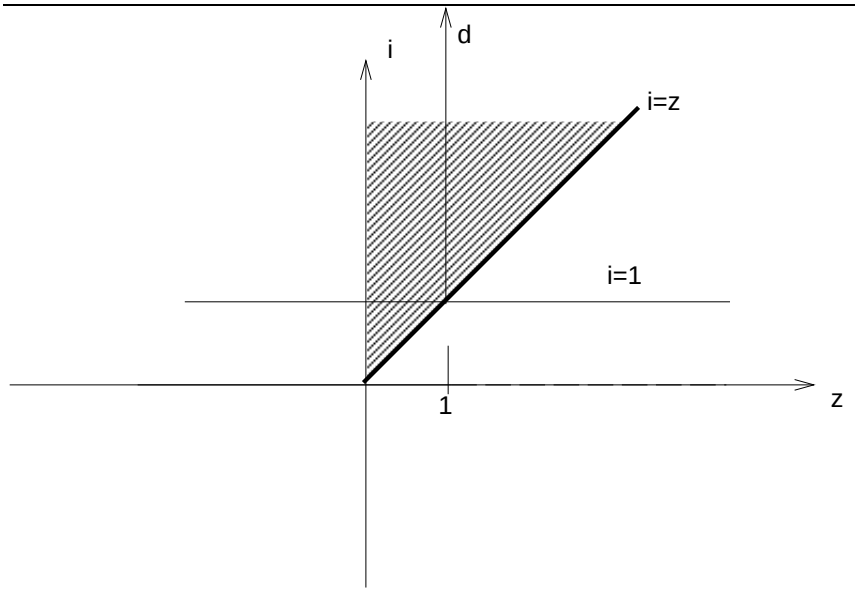


Figure 3: Domain $\mathcal{D}$ of the example, z is a parameter, i a variable.

But this solution cannot be extended to the complete parameter space. Indeed, we do not have $i \geq 1$ for all points of the hatched zone of figure 3. This situation appears when the convex is not *full dimensional* (in this case we do not have a unique minimal representation for domain $\mathcal{D}$ [Sch86]). As it is hard to affirm that such domains will never appear, we will prove that when they appear, the solution given is not (21) because inequalities such as $i \geq 1$ do not define domain $\mathcal{D}$.

In order to obtain this solution, we have taken the system of inequalities:

$$\begin{cases} z - 1 \geq 0 \\ -z + 1 \geq 0 \\ i - z \geq 0 \end{cases}$$

and we have solved (in i) an extracted system of equalities ($\{z = 1\ ,\ i = z\}$ for example). That is what PIP would do: choose n *good* inequalities and solve the corresponding system of equalities. But PIP wouldn't give solution (21), but the following solution:

$$\begin{cases} \text{in context } \{z = 1\} \\ \text{the minimum lexicographic of } \mathcal{D} = \{i \mid i - z \geq 0\} \text{ is } i = z \end{cases}$$

which can be extended to the complete parameter space.

The basic intuitive idea is simple. The Dual simplex algorithm chooses n inequations from m until it reaches a problem $P(\vec{z})$ such that the solution is obvious ($\vec{x} = \vec{0}$). During these transformations, no additional constraint appears (except in the parameter space), but the algorithm is doing two things simultaneously: finding which constraints define the optimum (i.e. finding the basis for which the solution is $\vec{x} = \vec{0}$), and solving the corresponding

system of n equalities (because the solution is directly given by the constant term of $S\vec{x} + \vec{t}(\vec{z})$). As PIP will choose its inequalities from the domain constraints (and not from the context constraints), and as the solution of the equality system will correspond to a positive combination of these equalities (at least for the first variable), this solution will be extensible to the inequalities system. That is what we prove in the next section.

6.2 Setting the problem

PIP's algorithm finds the lexicographic minimum of

$$\mathcal{D}^0(\vec{z}) = \{\vec{x} \in \mathbb{Z}^n \mid C\vec{x} + C'\vec{z} + \vec{c} \geq \vec{0}, \vec{x} \geq \vec{0}\}$$

as a function of $\vec{z}$ in the domain $\{\vec{z} \mid K\vec{z} + \vec{h} \geq \vec{0}\}$ called the *context*. Thus, the general form of a parametrized problem solved by PIP is:

$$P^0(\vec{z}) = \begin{cases} \text{in context: } \vec{z} \in \{K\vec{z} + \vec{h} \geq \vec{0}\} \\ \text{get the lexicographic minimum of the domain: } \mathcal{D}^0(\vec{z}) \end{cases} \tag{22}$$

- n is the dimension of $\vec{x}$, m is the number of inequations of the system $C\vec{x} + C'\vec{z} + \vec{c} \geq \vec{0}$, p is the number of parameters.
- C is a $m \times n$ matrix, C' is a $m \times p$ matrix.

The solution $Sol(\vec{z})$ given by PIP in the context is in the form of a multiguard

$$Sol(\vec{z}) = \begin{cases} \vec{z} \in Dom_1 : \vec{f^1}(\vec{z}) \\ \vdots \\ \vec{z} \in Dom_q : \vec{f^q}(\vec{z}) \end{cases} \tag{23}$$

- $(Dom_1, \ldots, Dom_q)$ being a partition of the context
- $\vec{f^i}(\vec{z})$ being an affine function from $\mathbb{Z}^p$ to $\mathbb{Z}^n$.

We now prove the following proposition:

Proposition 1 *Given a parametrized problem $P^0(\vec{z})$ (22), the solution $Sol(\vec{z})$ (23) given by PIP is such that, for each solution $\vec{f^i}(\vec{z})$, the inequation*

$$\vec{x}[1] - \vec{f^i}(\vec{z})[1] \geq 0 \tag{24}$$

is a positive linear combination of n inequations of $\mathcal{D}^0(\vec{z})$.

Proof

In the following, we will deal with basis changes. Thus it is important to know in which basis the coordinates of $\vec{x}$ are expressed. For the sake of clarity, we introduce the following notations:

- $\vec{x}^0 = (x_1^0, \ldots, x_n^0)$ is the original (canonical) coordinates system of $\mathbb{Z}^n$. We will note $\mathcal{B}_0$ the corresponding canonical basis.
- $\vec{x}^l = (x_1^l, \ldots, x_n^l)$ is another coordinate system of $\mathbb{Z}^n$. We will note $\mathcal{B}_l$ the corresponding basis.
- We will use the *problem tableau* notation:

$$\mathcal{D}^0(\vec{z}) = \{\vec{x}^0 \mid S\vec{x}^0 + \vec{t}(\vec{z}) \geq \vec{0}\}$$

(here, $S = \begin{pmatrix} I \\ C \end{pmatrix}$ $\vec{t}(\vec{z}) = \begin{pmatrix} 0 \\ C' \end{pmatrix} \vec{z} + \begin{pmatrix} \vec{0} \\ \vec{c} \end{pmatrix}$).

- The i^{th} row of $S\vec{x}^0 + \vec{t}(\vec{z})$ represents an expression denoted by $ineq_i(\vec{x}^0, \vec{z})$:

$$ineq_i(\vec{x}^0, \vec{z}) = (S\vec{x}^0 + \vec{t}(\vec{z}))[i] = S_{i\bullet}\vec{x}^0 + t_i(\vec{z})$$

We have seen in section 5 that, when solving (22), PIP's algorithm executes a succession of steps, each step having to solve a problem $P^l(\vec{z})$ whose form is:

$$P^l(\vec{z}) = \left\{ \begin{array}{l} \text{in context: } \vec{z} \in \{K\vec{z} + \vec{h} \geq \vec{0}, K^l\vec{z} + \vec{h}^l \geq \vec{0}\} \\ \text{get the lexicographic minimum of the domain:} \\ \mathcal{D}^l(\vec{z}) = \{A^l\vec{x}^l + A'^l\vec{z} + \vec{a}^l \;\Big|\; \begin{array}{l} A^l\vec{x}^l + A'^l\vec{z} + \vec{a}^l \geq \vec{0}, \\ C^l\vec{x}^l + C'^l\vec{z} + \vec{c}^l \geq \vec{0} \end{array} \} \end{array} \right. \tag{25}$$

Remind that we denote:

$$S^l = \begin{pmatrix} A^l \\ C^l \end{pmatrix} \quad \vec{t}^l(\vec{z}) = \begin{pmatrix} A'^l \\ C'^l \end{pmatrix} \vec{z} + \begin{pmatrix} \vec{a}^l \\ \vec{c}^l \end{pmatrix}$$

We will show that at each step, $P^l(\vec{z})$ has the following properties:

1. $\mathcal{D}^l(\vec{z})$ represents exactly the points of $\mathcal{D}^0(\vec{z})$, $\vec{x}^l$ being expressed in a basis $\mathcal{B}_l$ such that the expressions of $\vec{x}^l$ in terms of $\vec{x}^0$ are given by the left parts of n inequalities defining $\mathcal{D}_0(\vec{z})$. This can be expressed by: $\exists \sigma_l : [1, n] \to [1, m]$ injective, such that:

$$\forall i \in [1, n],\ x_i^l = ineq_{\sigma_l(i)}(\vec{x}^0, \vec{z}).$$

2. Each line of $S^l\vec{x}^l + \vec{t}^l(\vec{z}) \geq \vec{0}$ is the expression in the new basis of the corresponding inequality of $\mathcal{D}^0(\vec{z})$. i.e:

$$\forall i \in [1, m],\ S_{i\bullet}^l\vec{x}^l + t_i^l(\vec{z}) = ineq_i(\vec{x}^0, \vec{z}).$$

3. The coefficients of the first row of S^l are non negative.

As one could argue that PIP's algorithm is not exactly the Dual simplex method (due to the lexicographical cost function), we give a technical demonstration of these intuitive properties, based on the implemented algorithm itself. We will show these properties by recurrence on the steps of the algorithm.

The three properties are true for the first problem $P^0(\vec{z})$ (22). Indeed, we have $x_i^0 = ineq_i(\vec{x}^0, \vec{z})$, thus σ is the identity. Property 2 is verified by definition of $ineq_i(\vec{x}, \vec{z})$ and, as the first line of S^0 is $(1, 0, \ldots, 0)$ it has all its coefficients non negative thus property 3 holds too.

Consider now a problem $P^l(\vec{z})$ generated by PIP from $P_0(\vec{z})$:

$$P^l(\vec{z}) = \left\{ \begin{array}{l} \text{in context: } \vec{z} \in \{K\vec{z} + \vec{h} \geq \vec{0}, K'\vec{z} + \vec{h}' \geq \vec{0}\} \\ \text{get the lexicographic minimum of the domain:} \\ \mathcal{D}^l(\vec{z}) = \{A^l\vec{x}^l + A'^l\vec{z} + \vec{a}^l \mid \ S^l\vec{x}^l + \vec{t}^l(\vec{z}) \geq \vec{0} \ \} \end{array} \right.$$

and suppose that the three properties are true for this problem. We will study the different subproblems that can be generated by PIP in one step and see that the three properties hold for each of these subproblems.

A step of PIP's algorithm is divided into four possible cases:

- if each component of $\vec{t}^l(\vec{z})$ is non negative, then the solution is reached (it is given by the first n components of $\vec{t}^l(\vec{z})$). No subproblem is generated.
- if no component of $\vec{t}^l(\vec{z})$ is known as negative and if there is a $t_i^l(\vec{z})$ whose sign is unknown, then the two subproblems generated are:

$$P_1^{l+1}(\vec{z}) = \left\{ \begin{array}{l} \text{in context: } \vec{z} \in \left\{ \vec{z} \mid \begin{array}{c} K\vec{z} + \vec{h} \geq \vec{0} \\ K'\vec{z} + \vec{h}' \geq \vec{0} \\ t_i^l(\vec{z}) \geq 0 \end{array} \right\} \\ \text{get the lexicographic minimum} \\ \text{of the domain } \mathcal{D}^l(\vec{z}) \end{array} \right.$$

and

$$P_2^{l+1}(\vec{z}) = \left\{ \begin{array}{l} \text{in context: } \vec{z} \in \left\{ \vec{z} \mid \begin{array}{c} K\vec{z} + \vec{h} \geq \vec{0} \\ K'\vec{z} + \vec{h}' \geq \vec{0} \\ t_i^l(\vec{z}) < 0 \end{array} \right\} \\ \text{get the lexicographic minimum} \\ \text{of the domain } \mathcal{D}^l(\vec{z}) \end{array} \right.$$

  $P_1^{l+1}(\vec{z})$ and $P_2^{l+1}(\vec{z})$ have obviously the three properties requested (these properties do not take the context into account).
- if there is a negative $t_i^l(\vec{z})$ such that all the elements of $S_{i\bullet}$ are negative, then there is no solution in this leaf, no subproblem is generated.

- in the remaining case a pivoting step is performed. There is an i_0 such that $t^l_{i_0}(\vec{z})$ is negative and some elements $S_{i_0 j}$ are positive. Among all the possible j, we choose j_0 such that it minimizes (lexicographically speaking) the vector $S_{\bullet j_0}/S_{i_0 j_0}$. Then we consider a new basis $\mathcal{B}_{l+1}$ of $\mathbb{Z}^n$ such that:

$$\begin{aligned} x_i^{l+1} &= x_i^l \text{ for } i \neq j_0 \\ x_{j_0}^{l+1} &= S^l_{i_0 \bullet}\vec{x}^l + t^l_{i_0}(\vec{z}) \end{aligned} \tag{26}$$

(We say that $x^l_{j_0}$ leaves the basis and $x^{l+1}_{j_0}$ enters it). This basis change is valid because $S_{i_0 j_0} \neq 0$. If we note:

$$\mathcal{P}_{i_0 j_0} = \begin{pmatrix} 1 & 0 & \dots & & & 0 \\ 0 & 1 & 0 & \dots & & 0 \\ \vdots & & \ddots & & & \vdots \\ S_{i_0 1} & S_{i_0 2} & \dots & S_{i_0 j_0} & \dots & S_{i_0 n} \\ \vdots & & & & \ddots & \vdots \\ 0 & & \dots & & 0 & 1 \end{pmatrix}, \vec{\mathcal{C}}_{i_0 j_0} = \begin{pmatrix} 0 \\ \vdots \\ t_{i_0}(\vec{z}) \\ \vdots \\ 0 \end{pmatrix}$$

then we have:

$$\vec{x}^{l+1} = \mathcal{P}_{i_0 j_0}\vec{x}^l + \vec{\mathcal{C}}_{i_0 j_0} \Rightarrow \vec{x}^l = \mathcal{P}^{-1}_{i_0 j_0}(\vec{x}^{l+1} - \vec{\mathcal{C}}_{i_0 j_0}).$$

If we want to express the problem tableau in the new basis, we obtain:

$$S^l\vec{x}^l + \vec{t}^l(\vec{z}) = S^l\mathcal{P}^{-1}_{i_0 j_0}(\vec{x}^{l+1} - \vec{\mathcal{C}}_{i_0 j_0}) + \vec{t}^l(\vec{z})$$

The problem $P^l(\vec{z})$ expressed in basis $\mathcal{B}_{l+1}$ is:

$$P^l(\vec{z}) = \left\{ \begin{array}{l} \text{in context: } \vec{z} \in \{K\vec{z} + \vec{h} \geq \vec{0}, K^l\vec{z} + \vec{h}^l \geq \vec{0}\} \\ \text{get the lexicographic minimum of the domain:} \\ \mathcal{D}^l(\vec{z}) = \{A^l\mathcal{P}^{-1}_{i_0 j_0}(\vec{x}^{l+1} - \mathcal{C}_{i_0 j_0}) + A'^l\vec{z} + \vec{a}^l \\ \qquad\qquad \mid S^{l+1}\vec{x}^{l+1} + \vec{t}^{l+1}(\vec{z}) \geq \vec{0} \ \} \end{array} \right.$$

with the relations:

$$\begin{aligned} S^{l+1}_{\bullet j_0} &= (1/S^l_{i_0 j_0})S^l_{\bullet j_0} \\ S^{l+1}_{\bullet k} &= S^l_{\bullet k} - (S^l_{i_0 k}/S_{i_0 j_0})S^l_{\bullet j_0},\ k \neq j_0 \\ \vec{t}^{l+1}(\vec{z}) &= \vec{t}^l(\vec{z}) - (t^l_{i_0}(\vec{z})/S^l_{i_0 j_0})S_{\bullet j_0} \end{aligned}$$

And this is exactly the subproblem generated by PIP as $P^{l+1}(\vec{z})$. This implies several things:

  - $P^l(\vec{z})$ and $P^{l+1}(\vec{z})$ are the same problems expressed in different bases; thus they have the same solution (indeed, PIP takes the solution of $P^{l+1}(\vec{z})$ as the solution of $P^l(\vec{z})$).

– As $P^l(\vec{z})$ has property 2, from (26) we have:

$$S^l_{i_0\bullet}\vec{x}^l + t^l_{i_0}(\vec{z}) = ineq_{i_0}(\vec{x}^0, \vec{z}) \Rightarrow x^{l+1}_{j_0} = ineq_{i_0}(\vec{x}^0, \vec{z})$$

$P^l(\vec{z})$ has also property 1, thus:

$$\forall i \in [1, n],\ i \neq j_0 \Rightarrow x^{l+1}_i = ineq_{\sigma_l(i)}(\vec{x}^0, \vec{z})$$

If we note $\sigma_{l+1}(i) = \sigma_l(i)$ for $i \neq j_0$ and $\sigma_{l+1}(j_0) = i_0$, we have property 1 for $P^{l+1}(\vec{z})$ with σ_{l+1}. We just have to check that σ_{l+1} is injective. This is enforced by the fact that $\mathcal{B}_{l+1}$ is a basis. If σ_{l+1} was not injective, two components of $\vec{x}^{l+1}$ would always be equal (we could also note that, if line i_0 had already been chosen, the corresponding line of $S^l(\vec{z}) + \vec{t^l}(\vec{z}) \geq \vec{0}$ would be: $x^l_j \geq 0$ for some j, thus $t^l_{i_0}(\vec{z})$ wouldn't be negative). Finally: we have property 1 for $P^{l+1}(\vec{z})$.

– Property 2 becomes obvious too:

$$\forall i \in [1, m],\ S^{l+1}_{i\bullet}\vec{x}^{l+1} + t^{l+1}_i(\vec{z}) = S^l_{i\bullet}\vec{x}^l + t^l_i(\vec{z}) = ineq_i(\vec{x}^0, \vec{z})$$

– The coefficients of the first line of S^{l+1} are still non negative because j_0 has been chosen in such a way that the first component of the column that will be subtracted to all columns is minimum. (If $S^l_{1k} - (S^l_{i_0k}/S_{i_0j_0})S^l_{1j_0} < 0$, k would have been chosen instead of j_0.) Thus property 3 holds for $P^{l+1}(\vec{z})$.

We have the partial result:

- Properties 1, 2 and 3 hold for any subproblem generated by PIP

Consider now a subproblem $P^l(\vec{z})$ for which a solution is found (i.e all the components of $\vec{t^l}(\vec{z})$ are positive). We have seen that the solution to $P^l(\vec{z})$ (thus to $P^0(\vec{z})$ with the additional context $\{K^l\vec{z} + \vec{h}^l \geq \vec{0}\}$) is given by the first n components of $\vec{t^l}(\vec{z})$: $\vec{f}(\vec{z}) = (t^l_1(\vec{z}), \ldots, t^l_n(\vec{z}))^T$. From property 2 we know that

$$S^l_{1\bullet}\vec{x}^l + t^l_1(\vec{z}) = ineq_1(\vec{x}^0, \vec{z}) = x^0_1 \tag{27}$$

From Property 1, we know how to express each components of $\vec{x}^l$ in terms of $\vec{x}^0$'s components:

$$x^l_i = ineq_{\sigma_l(i)}(\vec{x}^0, \vec{z}) \tag{28}$$

Thus, from (27) and (28), we get:

$$S^l_{11}ineq_{\sigma_l(1)}(\vec{x}^0, \vec{z}) + \ldots + S^l_{1n}ineq_{\sigma_l(n)}(\vec{x}^0, \vec{z}) = x^0_1 - t^l_1(\vec{z}) \tag{29}$$

Finally, consider the positive linear combination of the n inequations numbered by $\sigma_l(1), \ldots, \sigma_l(n)$ as indicated in equation (29) (remember that, from property 3, each element of $S_{1\bullet}$ is non negative). We get:

$$\begin{cases} ineq_{\sigma_l(1)}(\vec{x}^0, \vec{z}) \geq 0 \\ \vdots \\ ineq_{\sigma_l(n)}(\vec{x}^0, \vec{z}) \geq 0 \end{cases} \quad \Rightarrow \quad x_1^0 - t_1^l(\vec{z}) \geq 0$$

Thus we have proposition 1. □

We are now able to prove our result which is stated in Proposition 2:

Proposition 2 *Given a parametrized problem $P^0(\vec{z})$ (22), the solution $Sol(\vec{z})$ (23) given by PIP is such that the relation:*

$$\forall i \in [1, q], (\vec{z} \in Dom_i,\ \vec{x} \in \mathcal{D}^0(\vec{z})) \Rightarrow \vec{x}[1] \geq \vec{f}^i(\vec{z})[1]$$

can be replaced by:

$$\vec{x} \in \mathcal{D}^0(\vec{z}) \Rightarrow \vec{x}[1] \geq \max_{i=1,q}(\vec{f}^i(\vec{z})[1])$$

Proof Directly from proposition 1:

The inequation

$$\vec{x}[1] \geq \vec{f}^i(\vec{z})[1]$$

is implied by the inequations defining $\mathcal{D}^0(\vec{z})$. Thus, the relation

$$(\vec{z} \in Dom_i,\ \vec{x} \in \mathcal{D}^0(\vec{z})) \Rightarrow \vec{x}[1] \geq \vec{f}^i(\vec{z})[1]$$

can be extended to:

$$\vec{x} \in \mathcal{D}^0(\vec{z}) \Rightarrow \vec{x}[1] \geq \vec{f}^i(\vec{z})[1]$$

□

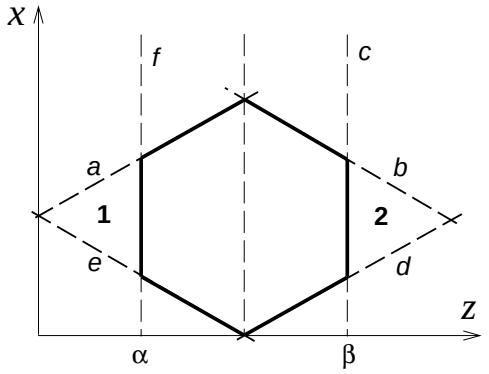
7 Conclusion

In this paper, we have presented a method to solve the loop rewriting problem. This problem is present in most automatic parallelization techniques. Generalizations of the simplex algorithm, like PIP, are particularly well-suited for this problem, except that the output of the PIP algorithm could not be directly transformed into a loop nest conforming to the definition given in Section 4.2. We have shown that, if the parameters $\vec{z}$ are such that the polyhedron $\mathcal{D}(\vec{z})$ is not empty, the result can be rewritten as the maximum of a finite number of affine forms in $\vec{z}$. A symmetrical result holds for the upper bound.

In some languages (notably Fortran), expressing loop bounds as maxima or minima is simpler than using conditional expressions, which must be converted to conditional instructions by the introduction of temporaries.

When rewriting a single loop nest, after a transformation like loop interchange, we have shown (see Section 4.3) that the values of the outer loop bounds are such that the range of inner loop is never empty. Hence, we may directly use the results of PIP in the form of maxima and minima.

In the *systolic* approach, however, we may have to construct the union of two or more iteration domains, and the above property will no longer be true. In that case, each inner loop must be guarded by a test. For example, take an instruction whose iteration space is the one below:



where $a, ..., f$ are affine forms in z. The first application of PIP yields:

$$\alpha \leq z \leq \beta$$

and the second one gives:

$$\max(e, d) \leq x \leq \min(a, b).$$

If the statement is to be executed alone, we may write:

```
DO z = α, β
        DO x = max(e, d), min(a, b)
                S
        ENDDO
ENDDO
```

Suppose however that we are forced to enlarge the z loop, possibly because it represents logical time over the whole program. Since zones 1 and 2 must not be scanned, we must write:

```
DO z = 0, L
        IF α ≤ z ≤ β
                DO x = max(e, d), min(a, b)
                        S
                ENDDO
```

```
            ENDIF
            ... other statements ...
        ENDDO
```

The expressions of the bounds as maxima or minima have the added advantage that they may be converted to linear constraints. They thus can be used as context for the computation of the next inner loop. This is optional, since PIP will find equivalent results whenever the parameters belong to the context. Choice between the two methods must be based on considerations of compile time performance. Results of Chamski in [Cha93] tend to show that omitting the context gives a faster algorithm, but this has to be confirmed by more extensive experiments.

Our aim now is to use this method in the back-end of the PAF parallelizer (*Parallèliseur Automatique pour Fortran*) [Fea91, RWF90, RWF91]. In PAF, the hidden parallelism of the sequential code is expressed according to the *systolic* point of view of section 3. We thus hope to be able to produce, in a mechanical way, streamlined parallel code from PAF's front end systolic descriptions.

Acknowledgments

This work has been partially supported by the Coordinated Research Project on Concurrency, Communication and Cooperation C^3 of the French Council for Research CNRS. The first author (J.F.C.) wishes to acknowledge the support of PRC/MRE contract "ParaDigme" and DRET contract 91/1180. The third author (T.R.) has also been partially supported by the ESPRIT Basic Research Action 6632 "NANA2" of the European Economic Community.

References

[AI91] C. Ancourt and F. Irigoin. Scanning polyhedra with DO loops. In *Proc. ACM SIGPLAN '91*, pages 39–50, June 1991.

[BL92a] M. Barnett and Ch. Lengauer. Loop parallelization and unimodularity. In *Algorithmique parallèle*, pages 369–379, Paris, 1992. Masson.

[BL92b] M. Barnett and Ch. Lengauer. Unimodularity considered nonessential. In *Proc. CONPAR 92 – VAPP V, LNCS 634*, pages 659–664, Lyon, September 1992.

[Cha93] Z. Chamski. *Environnement logiciel de programmation d'un accélérateur de calcul parallèle*. PhD thesis, Univ. Rennes I, Rennes, February 1993.

[DRR93] A. Darte, T. Risset, and Y. Robert. Loop nest scheduling and transformations. In J.J. Dongarra and B. Tourancheau, editors,

Environments and Tools for Parallel Scientific Computing. North Holland, 1993.

[Fea88] P. Feautrier. Parametric integer programming. *RAIRO Recherche Opérationnelle*, 22:243–268, September 1988.

[Fea89] P. Feautrier. Semantical analysis and mathematical programming; application to parallelization and vectorization. In P. Quinton M. Cosnard, Y. Robert and M. Raynal, editors, *Workshop on Parallel and Distributed Algorithms, Bonas*, pages 309–320. North Holland, 1989.

[Fea91] P. Feautrier. Dataflow analysis of scalar and array references. *Int. Journal of Parallel Programming*, 20(1):23–53, February 1991.

[Fea92a] P. Feautrier. Some efficient solution to the affine scheduling problem, part I, one-dimensional time. Technical Report 92.28, IBP/MASI, May 1992.

[Fea92b] P. Feautrier. Some efficient solution to the affine scheduling problem, part II, multidimensional time. Technical Report 92.78, IBP/MASI, October 1992.

[Gom63] R. E. Gomory. An algorithm for integer solutions to linear programs. In R. L. Graves and P. Wolfe, editors, *Recent Advances in Math. Programming*, chapter 34, pages 269–302. Mac-Graw Hill, New York, 1963.

[HKT91] S. Hiranandani, K. Kennedy, and C.-W. Tseng. Compiler optimizations for Fortran D on MIMD distributed-memory machines. In *Supercomputing 91*, pages 86–100. IEEE Computer Society Press, November 1991.

[Lam74] L. Lamport. The parallel execution of do loops. *Communications of The ACM*, 17(2):83–93, February 1974.

[LHS90] L.S. Liu, C.W. Ho, and J.P. Sheu. On the parallelism of nested for-loops using index shift method. In *Proceedings of International Conference on Parallel Processing*, volume 2, pages 119–123, August 1990.

[QR89] P. Quinton and Y. Robert. *Systolic Algorithms and Architectures*. Prentice Hall and Masson, 1989.

[Qui87] P. Quinton. The systematic design of systolic arrays. In Françoise Fogelman Soulie, Yves Robert, and Maurice Tchuente, editors, *Automata Networks in Computer Science*, chapter 9, pages 229–260. Manchester University Press, 1987.

[RWF90] M. Raji-Werth and P. Feautrier. Systematic construction of programs for distributed memory systems. In P. Feautrier and

François Irigoin, editors, *Proc. of the Int. Workshop on Compiler for Parallel Computers, Paris*, December 1990.

[RWF91] M. Raji-Werth and P. Feautrier. On parallel program generation for massively parallel architectures. In M. Durand and F. El Dabaghi, editors, *High Performance Computing II*. North-Holland, October 1991.

[Sch86] A. Schrijver. *Theory of linear and integer programming*. Wiley, New York, 1986.

[SF91] W. Shang and A.B. Fortes. Time optimal linear schedules for algorithms with uniform dependencies. *IEEE Trans. on Computers*, 40(6):723–742, June 1991.

[WL91] M. E. Wolf and M. S. Lam. A loop transformation theory and an algorithm to maximize parallelism. *IEEE Trans. Parallel Distributed Systems*, 2(4):452–471, October 1991.

[Wol89] M. Wolfe. *Optimizing Supercompilers for Supercomputers*. MIT Press, Cambridge MA, 1989.

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