

**A heuristic for improving the regularity
of accesses by global loop
transformations in the polyhedral
model**

Sven Verdoolaege *Francky Catthoor*
Maurice Bruynooghe *Gerda Janssens*

Report CW 325, November 2001



Katholieke Universiteit Leuven
Department of Computer Science

Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

A heuristic for improving the regularity of accesses by global loop transformations in the polyhedral model

Sven Verdoolaege *Francky Catthoor*
Maurice Bruynooghe *Gerda Janssens*

Report CW 325, November 2001

Department of Computer Science, K.U.Leuven

Abstract

Our approach for global loop transformations aimed at optimizing data transfer and storage is based on an extended polytope model. The transformations are performed in two steps: a placement step that maps the individual polytopes to a common iteration space and an ordering step defining an order in the common iteration space.

Recently, Danckaert has proposed to split the placement step itself into a first substep dealing with the linear part of the mappings and focusing on the regularity of accesses and a second substep dealing with the translation part and focusing on the locality. In this context, he developed a criterion for optimizing the regularity of the dependencies between read and write operations together with a search procedure.

This paper shows that this optimization criterion is in fact an approximation of the dimension of the dependency distance vector polytope. We further present some improvements on the existing search procedure together with two new search procedures.

Keywords : polyhedral model, loop transformations, access regularity, DTSE, dependency distance vector, polytope dimension

CR Subject Classification : D.3.4 Optimization

A heuristic for improving the regularity of accesses by global loop transformations in the polyhedral model

Sven Verdoolaege Francky Catthoor*
Maurice Bruynooghe Gerda Janssens

November 2001

1 Introduction

1.1 Basic Idea

The polyhedral model is widely used for modeling data and control flow. It has its origin in systolic array synthesis and has been applied in diverse optimization techniques, including automatic loop parallelization (Feautrier 1996) and memory usage optimization (Quilleré and Rajopadhye 2000). A key feature is that the model is exact, yet mathematically compact. By mapping each iteration of a loop to a coordinate value in a geometrical space, the model represents each statement enclosed by an n -level loop nest as an n -dimensional polytope in this space, whose size in each dimension is determined by the loop bounds of the corresponding loop in the loop nest.

To perform control flow transformations in the polyhedral model, van Swaaij et al. (1992) propose to work in two phases: a placement step in which the individual polytopes are mapped to a common iteration space and an ordering step defining an ordering vector in this common iteration space. Danckaert et al. (2000) propose to split the placement itself further into a first substep dealing with the linear part of each mapping and focusing on the regularity of accesses and a second substep dealing with the translation part and focusing on locality.

Danckaert (2001) derives a simple criterion, the dimension of the dependency distance vector polytope, that can be used as a first approximation for the *sharpness* of the dependency distance vector cone, which is in itself proposed as a criterion for optimizing the regularity of the dependencies. Dependency distance vector cones of the same dimension can then further be compared to each other using a general notion of “angle”.¹

*IMEC, also prof. at Katholieke Universiteit Leuven

¹Note that “dependency distance vector cone” and “dependency distance vector polytope” are distinct. We will have more to say on this in section 2.

We will first repeat the gist of the derivation, followed by a refinement of the expression for calculating the dimension and a new expression for calculating the smallest reachable dimension. We conclude with three algorithms that try and optimize the dimension. The first of these algorithms is based on the one proposed in Danckaert (2001). Throughout we assume that all accesses, manifest conditions and loop bounds are affine.

1.2 Notation

When it leads to a more compact notation, we will sometimes use homogeneous coordinates, where a point with Cartesian coordinates $\vec{x} \in \mathbb{Z}^n$ is represented by any of a set of homogeneous coordinates $\begin{bmatrix} w\vec{x} \\ w \end{bmatrix} \in \mathbb{Z}^{n+1}$. Since we only need to represent finite points, we can always use the representative with $w = 1$. We will denote this representative by \vec{x}^H .

Any affine transformation on Cartesian coordinates can be represented by a linear transformation on homogeneous coordinates (which we will denote by placing a tilde ($\tilde{\cdot}$) on top of the variable corresponding to the Cartesian transformation), by adding an extra column corresponding to the translation vector and an extra row \vec{e}_{n+1}^T . I.e., we represent the affine transformation (with linear transformation A and translation vector \vec{a})

$$\vec{y} = A\vec{x} + \vec{a}$$

by

$$\vec{y}^H = \tilde{A}\vec{x}^H,$$

where

$$\vec{y}^H = \begin{bmatrix} \vec{y} \\ 1 \end{bmatrix}, \quad \tilde{A} = \begin{bmatrix} A & \vec{a} \\ \vec{0}^T & 1 \end{bmatrix} \quad \text{and} \quad \vec{x}^H = \begin{bmatrix} \vec{x} \\ 1 \end{bmatrix}.$$

Note that this transformation preserves the value 1 of the final coordinate. Furthermore, both the product of two such matrices and the inverse of such a matrix, all have \vec{e}_{n+1}^T as their final row and the linear part of the product (inverse) is the product (inverse) of the linear parts(s) of the argument(s). The main benefit of this notation is that composition of transformations amounts to multiplication of the respective homogeneous transformation matrices. When adding (or subtracting) affine transformations, both the linear transformation matrices and the translation vectors should be added. In the homogeneous notation, this corresponds to adding all but the final row. We will allow ourselves a slight abuse in notation such that whenever we add or subtract homogeneous transformation matrices, the operation should be interpreted in this way.

As with transformation matrices, a set of inequalities $C\vec{x} \geq \vec{c}$ has its homogeneous counterpart in

$$\begin{bmatrix} C & -\vec{c} \\ \vec{0}^T & 1 \end{bmatrix} \begin{bmatrix} \vec{x} \\ 1 \end{bmatrix} \geq \vec{0} \quad \text{or} \quad \tilde{C}\vec{x}^H \geq \vec{0}.$$

2 Iteration space approximation

A dependency exists between two statements when one statement (the consumer) reads a subset of the elements of an array (U_1 in the example below) written by the other (possibly the same) statement (the producer). The element accessed in each iteration is the result of applying the index function (\tilde{J}_p for the producer and \tilde{J}_c for the consumer) to the loop iterator vector.² The loop iterator vectors themselves are constrained by the loop bounds, which can be expressed by a set of linear equations (\tilde{C}_p, \tilde{C}_c), defining the polytope corresponding to each statement. All this information can be described succinctly by the following two recurrence equations,

$$\begin{aligned} P_p &= \{\vec{x} \in \mathbb{Z}^n \mid \tilde{C}_p \vec{x}^H \geq \vec{0}\} : U_1(\tilde{J}_p(\vec{x}^H)) = f_1(\dots) \\ P_c &= \{\vec{x} \in \mathbb{Z}^n \mid \tilde{C}_c \vec{x}^H \geq \vec{0}\} : U_2(\tilde{J}_2(\vec{x}^H)) = f_2(\dots, U_1(\tilde{J}_c(\vec{x}^H)), \dots) \end{aligned}$$

where each statement is labeled with the polytope containing the iteration vectors for which it is executed. The f_i s in this example are arbitrary operations. Since the iteration vector from the production polytope (P_p) and the consumption polytope (P_c) are mapped to array elements by \tilde{J}_p and \tilde{J}_c respectively, the dependency function mapping each point of the consumption polytope to the point of the production polytope that produces the value it needs is given by:

$$\tilde{D} = \tilde{J}_p^{-1} \tilde{J}_c.^3$$

This dependency function is of course only valid for those points of P_c that need values produced by P_p (several polytopes may produce (different parts of) the same array). Therefore, the domain of the dependency function is composed of those points that are both in P_c and are mapped by D to points in P_p , or

$$P'_d = \left\{ \vec{x} \in \mathbb{Z}^n \mid \left[\begin{array}{c} \tilde{C}_c \\ \tilde{C}_p \tilde{J}_p^{-1} \tilde{J}_c \end{array} \right] \vec{x}^H \geq \vec{0} \right\}.^4 \quad (1)$$

After mapping both polytopes to a common iteration space (points \vec{x}_p^H in P_p are mapped to $\tilde{A}_p \vec{x}_p^H$ and points \vec{x}_c^H in P_c to $\tilde{A}_c \vec{x}_c^H$), the dependency function in this common iteration space is given by

$$\tilde{D}' \vec{x}'^H = (\tilde{A}_p \tilde{J}_p^{-1} \tilde{J}_c \tilde{A}_c^{-1}) \vec{x}'^H$$

with domain

$$P''_d = \left\{ \vec{x}' \in \mathbb{Z}^n \mid \left[\begin{array}{c} \tilde{C}_c \\ \tilde{C}_p \tilde{J}_p^{-1} \tilde{J}_c \end{array} \right] \tilde{A}_c^{-1} \vec{x}'^H \geq \vec{0} \right\}. \quad (2)$$

²We assume here that all loop nests and arrays have the same number of dimensions. This results in square transformation matrices which greatly simplifies their manipulation and can be obtained by adding dummy loops or dimensions.

³ \tilde{J}_p is or can be made invertible because we assume single-assignment code.

⁴In all likelihood, of the inequalities in this set many will be redundant. Some of them may even conflict, in which case the domain is empty. We assume here that a dependency analysis has been performed and that the dependency function has a non-empty domain.

The function values are again points in the common iteration space, so we can subtract this value from the input value to form the dependency distance vector pointing from the production point to the consumption point. That is, we have $\vec{v}^H = \tilde{V}\vec{x}^H$ with \tilde{V} given by:

$$\tilde{V} = \tilde{I} - \tilde{A}_p \tilde{J}_p^{-1} \tilde{J}_c \tilde{A}_c^{-1} \quad (3)$$

The dependency distance vector cone is then the cone generated by all the dependency distance vectors, i.e.,

$$\text{Cone}_d = \text{cone } P_d$$

with P_d the dependency distance vector polytope given by:⁵

$$P_d = \left\{ \vec{v} \in \mathbb{Z}^n \mid \exists \vec{x} \in \mathbb{Z}^n, \vec{v}^H = \tilde{V}\vec{x}^H, \begin{bmatrix} \tilde{C}_c \\ \tilde{C}_p \tilde{J}_p^{-1} \tilde{J}_c \end{bmatrix} \tilde{A}_c^{-1} \vec{x}^H \geq \vec{0} \right\}.$$

In figures, the distance vectors in this set will be depicted by arrows. More specifically, in a figure of an iteration space, the arrow will be drawn between the production point and the consumption point and in a figure of the distance vector space, it will be drawn between the origin and the vector point itself.

If the dimension of Cone_d is 1, all the dependency distance vectors point in the same direction. This situation is ideal, because it maximizes the choice in ordering vectors. Even if we cannot obtain dimension 1, a smaller dimension may be preferable, because we then still have complete freedom for the ordering vector in all the other dimensions. The idea is therefore to first minimize

$$\dim \text{Cone}_d$$

over all (regular) transformation matrices \tilde{A}_p and \tilde{A}_c . Among solutions with the same dimension we could then additionally compare the generalized angles of the cones (Danckaert 2001), which we will not discuss further in this document. During the first phase of the placement, where only the linear part is fixed, the dimension of the cone can however not be determined since it depends on the final column of \tilde{V} which is in turn determined by the final columns (i.e., the translation vectors) of the transformation matrices.

What *can* be done is to determine the dimension of P_d while the dimension of Cone_d is at most one more than the dimension of P_d . More specifically, for each \vec{v} in P_d ,

$$\vec{v} = V\vec{x} + \vec{v}_{n+1}$$

and therefore each element of Cone_d will be a linear combination of \vec{v}_{n+1} and columns from V . If, during the translation phase, \vec{v}_{n+1} can be chosen from the column space of V , then $\dim \text{Cone}_d = \dim P_d$ and otherwise $\dim \text{Cone}_d =$

⁵ To see that P_d is a polytope, consider the following. $C = \{\vec{x} \mid C\vec{x} \geq \vec{c}\}$ is a polytope which means that it is the convex hull of finitely many vectors $\{\vec{v}_i\}_{1 \leq i \leq n}$ (Schrijver 1986), i.e., $C = \text{conv.hull}\{\vec{v}_i\}_{1 \leq i \leq n}$. $P_d = W \text{conv.hull}\{\vec{v}_i\}_{1 \leq i \leq n} = \text{conv.hull}\{W\vec{v}_i\}_{1 \leq i \leq n}$ is therefore also a polytope.

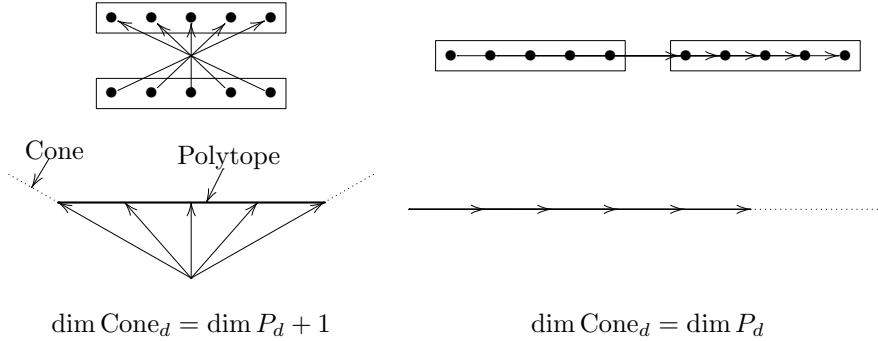


Figure 1: Dependency distance vector polytope and cone

$\dim P_d + 1$. Figure 1 illustrates this relation as well as the concepts dependency distance vector polytope and cone. The figure shows two possible translations for two polytopes with a dependency between them. On the left hand side, the offset is not part of the column space, whereas on the right hand side it is. On each side, the top part shows the common iteration space and the bottom part shows the distance vector space.

The new goal is thus to first minimize

$$r_d = \dim P_d, \quad (4)$$

where we rewrite P_d as follows:

$$P_d = \left\{ \vec{v} \in \mathbb{Z}^n \mid \exists \vec{x} \in \mathbb{Z}^n, \vec{v}^H = \tilde{W} \vec{x}^H, \begin{bmatrix} \tilde{C}_c \\ \tilde{C}_p \tilde{J}_p^{-1} \tilde{J}_c \end{bmatrix} \vec{x}^H \geq \vec{0} \right\} \quad (5)$$

with

$$\tilde{W} = \tilde{A}_c - \tilde{A}_p \tilde{J}_p^{-1} \tilde{J}_c. \quad (6)$$

This is allowed since \tilde{A}_c is regular. While \tilde{V} (3) maps points from the consumption polytope mapped on the common iteration space to points from the production polytope mapped on the same common iteration space, \tilde{W} maps points from the consumption polytope in its original space to points from the production polytope that have been mapped on this consumption space. The domain of this function is then the domain of \tilde{V} (P_d'' (2)), mapped back onto the consumption space, which is the same as the domain of \tilde{D} (P_d' (1)).

As a first approximation, we may drop the constraints on \vec{x} (which amounts to replacing both the production and the consumption polytope by \mathbb{Z}^n) and we get:

$$r_d \leq r_K = \dim \{ \vec{v} \in \mathbb{Z}^n \mid \exists \vec{x} \in \mathbb{Z}^n, \vec{v} = W \vec{x} + \vec{w}_{n+1} \} \quad (7)$$

or

$$\boxed{r_K = \text{rank } W} \quad (8)$$

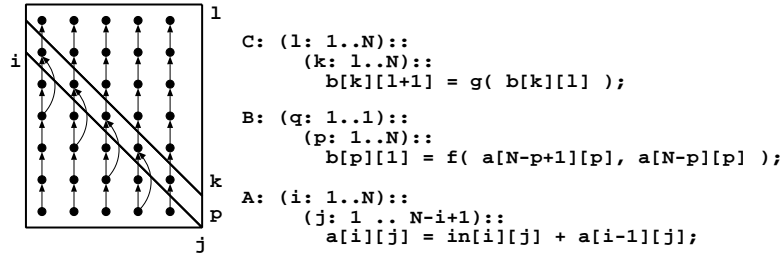


Figure 2: Example 2

which is the optimization criterion derived in Danckaert (2001). From equation 6 it should be clear that the optimal rank is

$$R_K := \min r_K = n - \text{rank } J_p^{-1} J_c \quad (9)$$

which also follows from the more general case given in section 4.2.

3 Affine hull approximation

Before trying to derive a formula for r_d , it may be insightful to see where the approximation given by equation 8 is insufficient and what this means.

Consider the example in figure 2. We will first examine the first dependency between A and B. The index functions are as follows:

$$\tilde{J}_p = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \tilde{J}_p^{-1} \tilde{J}_c = \tilde{J}_c = \begin{bmatrix} 0 & -1 & N+1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \quad (10)$$

From equation 9 it follows that the optimal dimension is 1 since the rank of $J_p^{-1} J_c$ is 1, although, clearly, the placement in the figure is such that the dimension is 0 (there is only one distance vector).

The explanation follows from an inspection of the set of inequalities defining polytope B:

$$\begin{bmatrix} 1 & 0 & -1 \\ -1 & 0 & 1 \\ 0 & 1 & -1 \\ 0 & -1 & N \end{bmatrix} \vec{x}^H \geq \vec{0}.$$

In this example, the top two rows imply: $[1 \ 0 \ -1] \vec{x}^H = \vec{0}$. In general, we have that for any set of inequalities

$$\tilde{C} \vec{x}^H \geq 0$$

there exists a subset of *implicit equalities* $\tilde{C}^=$ of \tilde{C} such that

$$\forall \vec{x} \in P : \tilde{C} \vec{x} \geq \vec{0} \Rightarrow \tilde{C}^= \vec{x} = \vec{0}.$$

As a result, any two transformation matrices that only differ in multiples of rows from $\tilde{C}^=$ really define the same transformation with respect to the given polytope. If we take $A_A = A_p = I_2$, then from equation 6 we would like to have:

$$A_B = A_c = \begin{bmatrix} 0 & -1 \\ 0 & 1 \end{bmatrix}$$

since this would result in a zero rank for W . This A_c cannot be used directly because it is singular, but adding $C_B^= = [1 \ 0]$ to the top row results in the equivalent, but regular

$$A_B = A_c = \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}. \quad (11)$$

The same result can be obtained directly by using

$$\tilde{J}_c = \begin{bmatrix} 1 & -1 & N \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

instead of the equivalent form in equation 10.

Since the dimension should not only be independent of the choice out of equivalent transformation matrices of the consumption polytope, but also of the production polytope, we conjecture that it should be given by the following formula:

$$r_a = \text{rank} \begin{bmatrix} W \\ C^a \end{bmatrix} - \text{rank } C^a \quad (12)$$

with

$$C^a := \begin{bmatrix} C_c^= \\ C_p^= J_p^{-1} J_c \end{bmatrix}.$$

As will be shown in section 4.1, this corresponds to the dimension of the dependency distance vector polytope when both production and consumption polytope are approximated by their affine hull, i.e.,

$$r_a := \dim \left\{ \vec{v} \in \mathbb{Z}^n \mid \exists \vec{x} \in \mathbb{Z}^n, \vec{v}^H = \tilde{W} \vec{x}^H, \begin{bmatrix} \tilde{C}_c^= \\ \tilde{C}_p^= J_p^{-1} J_c \end{bmatrix} \vec{x}^H = \vec{0} \right\}. \quad (13)$$

Reconsidering equation 12 reveals that, even in the case where the optimal rank is 0, there may be several distinct optimal solutions since we can use the implicit equalities of one polytope to arrive at a fundamentally different transformation matrix for the other polytope. Consider the example in figure 2 again and let us turn our attention to the dependency between polytopes B and C. We have

$$\tilde{J}_c = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \tilde{J}_p^{-1} \tilde{J}_c = \tilde{J}_p^{-1} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

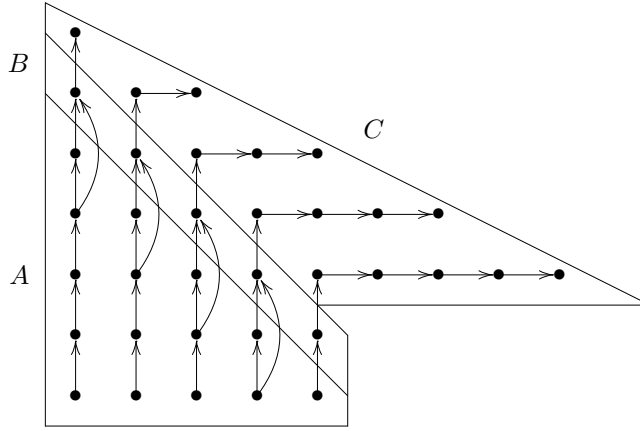


Figure 3: Alternative mapping

and for this dependency B is the production polytope, so we continue with A_p set to the A_B derived above (equation 11). Then equation 6 would result in

$$A_C = A_c = \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix}$$

which is the transformation matrix that was used to get at the placement in figure 2. We can, however, also add multiples of

$$C_p^= J_p^{-1} J_c = [1 \ 0] \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} = [0 \ 1]$$

to the rows of A_c without losing optimality. We can, for example, subtract it from the first row and add it to the second row to arrive at

$$A_C = A_c = \begin{bmatrix} -1 & 0 \\ 1 & 1 \end{bmatrix}.$$

The resulting placement (with a suitable translation) is shown in figure 3 where, indeed, the dimension of the dependency distance vector polytope is zero.

The difference between the affine hull approximation and the real dimension is that some inequalities from both sides of the dependency can conspire to produce extra implicit equalities that are not present in the affine hull approximation. Note also that the optimal dimension R_a is zero if and only if the dependency does not map several consumption points to the same production point.

4 General case

If we take the affine hull of P_d (equation 5) in the definition of r_d (4), then we see that each of r_d (4), r_a (13) and r_K (7) fits the general form⁶

$$r = \dim \left\{ \vec{v} \in \mathbb{Z}^n \mid \exists \vec{x} \in \mathbb{Z}^n, \vec{v}^H = \tilde{W} \vec{x}^H, \tilde{C}^= \vec{x}^H = \vec{0} \right\}. \quad (14)$$

We now derive a formula for the dimension given two transformation matrices and the optimal dimension over all possible transformation matrices. Intuitively, we can see that any row of W (which maps P'_d (1) to P_d) that is a linear combination of rows from $C^=$ will map all points of P'_d to the same value, thus reducing the dimension. The dimension of P_d should then correspond to the number of linearly independent rows in W that do not just differ in linear combinations of $C^=$.

Note that when we actually want to use the criteria derived in this section, we need to be able to find the implicit equalities first. This is discussed briefly in appendix A.

4.1 Dimension

First, we move the polytope to the origin

$$r = \dim \{ \vec{v} \in \mathbb{Z}^n \mid \exists \vec{x} \in \mathbb{Z}^n, \vec{v} = W \vec{x}, C^= \vec{x} = -\vec{c}^= \}$$

and we subsequently rewrite this as

$$r = \dim \{ \vec{v} \in \mathbb{Z}^n \mid \exists \vec{w} \in \mathbb{Z}^k, \vec{v} = WX \vec{w} \} \quad (15)$$

with the columns of X a basis for P_x , the polyhedron containing all the vectors that satisfy $C^= \vec{x} = -\vec{c}^=$.

We denote the dimension of this polyhedron by k which equals the dimension of the vector space minus the rank of the matrix defining the equalities (Schrijver 1986),

$$\dim P_x = \dim \{ \vec{x} \in \mathbb{Z}^n \mid C^= \vec{x} = -\vec{c}^= \} = n - \text{rank } C^= =: k. \quad (16)$$

The dimension of the polyhedron in equation 15 is given by the dimension of the range of the linear transformation f defined by

$$f : \mathbb{Z}^k \rightarrow \mathbb{Z}^n : \vec{w} \mapsto WX \vec{w}.$$

⁶ If the dependency function is given as follows (as in Danckaert (2001)):

$$D : \mathbb{Z}^n \rightarrow \mathbb{Z}^n = \left\{ \begin{bmatrix} \vec{x}_p \\ \vec{x}_c \end{bmatrix} \mid \exists \vec{\alpha} \in \mathbb{Z}^{n_d}, C_d \vec{\alpha} \geq \vec{c}_d, \begin{bmatrix} \vec{x}_p^H \\ \vec{x}_c^H \end{bmatrix} = \begin{bmatrix} \tilde{L}_p \\ \tilde{L}_c \end{bmatrix} \vec{\alpha}^H \right\},$$

then the dependency distance vector polytope is defined as

$$\left\{ \vec{v} \in \mathbb{Z}^n \mid \exists \vec{\alpha} \in \mathbb{Z}^{n_d}, C_d \vec{\alpha} \geq \vec{c}_d, \vec{v}^H = (\tilde{A}_c \tilde{L}_c - \tilde{A}_p \tilde{L}_p) \vec{\alpha}^H \right\}$$

and we can take $W = A_c L_c - A_p L_p$ and $C^= = C_D^=$ in equation 14.

For linear transformations, we know that

$$\dim \text{Ker } f + \dim \text{Ran } f = k. \quad (17)$$

The kernel of f are those vectors \vec{x} that map to $\vec{0}$, so

$$\dim \text{Ker } f = \dim \left\{ \vec{x} \in \mathbb{Z}^n \mid \begin{bmatrix} W \\ C^= \end{bmatrix} \vec{x} = \vec{0} \right\} = n - \text{rank} \begin{bmatrix} W \\ C^= \end{bmatrix}. \quad (18)$$

Substituting (18) and (16) in (17), we get

$$r = \dim \text{Ran } f = (n - \text{rank } C^=) - (n - \text{rank} \begin{bmatrix} W \\ C^= \end{bmatrix}),$$

or

$$r = \text{rank} \begin{bmatrix} W \\ C^= \end{bmatrix} - \text{rank } C^=$$

We can see that (8) and (12) are special cases of this general formula and that it confirms our intuition formulated at the beginning of section 4.

4.2 Optimal dimension

The optimal dimension is given by

$$R := \min_{A_c, A_p} r(A_c - A_p J_p^{-1} J_c).$$

Transforming both production and consumption polytope by the same (regular) A_p^{-1} does not change the dimension of the dependency distance vector polytope, so:

$$R = \min_A r(A - J_p^{-1} J_c)$$

From equation 15, we know:

$$r(A - J_p^{-1} J_c) = \text{rank}((A - J_p^{-1} J_c)X)$$

If we write

$$AX = (A - J_p^{-1} J_c)X + J_p^{-1} J_c X$$

then we know

$$\text{rank}(AX) \leq \text{rank}((A - J_p^{-1} J_c)X) + \text{rank}(J_p^{-1} J_c X)$$

and so

$$r(A - J_p^{-1} J_c) \geq r(A) - r(J_p^{-1} J_c) = n - \text{rank} \begin{bmatrix} J_p^{-1} J_c \\ C^= \end{bmatrix}$$

since A is regular.

This means we have a lower bound for R :

$$R \geq n - \text{rank} \begin{bmatrix} J_p^{-1} J_c \\ C^= \end{bmatrix}.$$

Note that this lower bound is tight, for take q linearly independent rows $D_q = \{\vec{d}_{i_1} \dots \vec{d}_{i_q}\}$ from $J_p^{-1} J_c$ that are also linearly independent of the rows in $C^=$ ($q = \text{rank} \begin{bmatrix} J_p^{-1} J_c \\ C^= \end{bmatrix} - \text{rank } C^=$). Let A_c be composed of $\vec{d}_{i_1} \dots \vec{d}_{i_q}$ as the first q rows, followed by $(\text{rank } C^=)$ linearly independent rows from $C^=$ and $(n - \text{rank} \begin{bmatrix} J_p^{-1} J_c \\ C^= \end{bmatrix})$ vectors that are linearly independent of any of the other rows (A_c is regular). Furthermore, let the first q rows of A_p be $\vec{e}_{i_1} \dots \vec{e}_{i_q}$ and write the remaining rows of $J_p^{-1} J_c$ (i.e., those not in D_q) as a linear combination of rows from D_q and $C^=$:

$$\vec{d}_j = \sum_{k=1}^q \alpha_{jk} \vec{d}_{i_k} + \sum_{k=1}^{\text{rank } C^=} \beta_{jk} \vec{c}_k$$

The remaining rows of A_p are $\vec{e}_j - \sum_{k=1}^q \alpha_{jk} \vec{e}_{i_k}$ for each j corresponding to a \vec{d}_j not in D_q . It is clear that A_p is also regular and the reader will easily verify that $\text{rank}((A - J_p^{-1} J_c)X) \leq n - \text{rank} \begin{bmatrix} J_p^{-1} J_c \\ C^= \end{bmatrix}$.

Therefore,

$$R = n - \text{rank} \begin{bmatrix} J_p^{-1} J_c \\ C^= \end{bmatrix}. \quad (19)$$

Equation 9 is easily seen to follow from this and for the affine hull approximation, we know that the $C_p^= J_p^{-1} J_c$ rows of C^a are linearly dependent of the rows of $J_p^{-1} J_c$, so $R_a = n - \text{rank} \begin{bmatrix} J_p^{-1} J_c \\ C^= \end{bmatrix}$.

5 Backtracking search

This section and the following ones will present algorithms for finding optimal (or “good”) solutions for the problem of minimizing over all dependencies between pairs of polytopes, the dimension of the dependency distance vector polytope as defined in equation 4.

The algorithm in this section is derived from the one in Danckaert (2001) and is the only one (so far) that can give a solution even if the true optimum cannot be reached. Reaching the true optimum means that for each dependency, the corresponding polytope has the optimal dimension as in equation 19. Note that the solution produced by the algorithm depends on the set of mappings described shortly and could in principle be severely non-optimal.

The original description can be summarized as follows. The algorithm first determines a set of possible mappings from which each of the mappings in the

solution will be selected. Currently, this set consists of all the regular matrices containing only 0's and 1's.⁷ Then three phases are undertaken:

1. For each dependency and for each pair of transformation matrices from the set, the rank given in equation 8 is calculated. It also records for each dependency the minimum of the calculated ranks. This minimum (called the *restricted* minimum) differs from equation 9 because the latter optimum might not be reachable with the given set of possible mappings.
2. For each polytope, those mappings are retained that can lead to the minimal rank for each of the dependencies it is involved in.
3. A number of subsequent backtracking searches are initiated where each time the algorithm searches for a solution satisfying a set of constraints of the form

$$r_K \leq c$$

for each dependency between two polytopes. Each level of the search corresponds to a polytope where it enumerates the mappings that were retained for this polytope.

The c -values are taken equal to the dimension for the first search to ensure that a solution can be found and are then gradually decreased until an optimal solution is found or until the algorithm hits upon a set of constraints that is unsatisfiable.

The complexity of phase 1 is dm^2 with d the number of dependencies and m the number of mappings in the initial set. For phase 2, it is dm , but the cost of each operation is a lot smaller than that of phase 1. Phase 3 is exponential in the worst case.

We can wonder whether this separation in three phases is a good idea. Let us first consider phase 2. It is a very cheap step, but it is not certain that it should be performed. In case the restricted optimum is reachable, it may throw away unnecessary mappings and thus speed up phase 3. However, since the solution is invariant over a transformation of the whole common iteration space, we can usually find a counterpart to be used for the other polytope involved in a dependency, for any of the possible transformation matrices. This is especially the case when there are implicit equalities involved implying that there are an infinite number of distinct solutions. In the case the restricted optimum cannot be reached, phase 2 boils down to a rather arbitrary removal of matrices that may severely restrict the “goodness” of the solution found in phase 3. The most pressing reason to remove phase 2 is that it requires phase 1, which is a costly operation and which, as we will see next, is not needed otherwise.

Without phase 2, the only result of phase 1 is the minimal reachable rank, which is used in phase 3 to avoid searching for a solution that is known not to exist. However, we already have a pretty good estimate of this minimal rank, viz. the optimal rank (9). Furthermore, if we arrange the searches in such a way

⁷ We have restricted this a bit further to remove the (few) matrices that are not unimodular.

that the dependency whose constraint has just been tightened is always the first to be checked, then we can see that all the iterations of any search that would have been skipped due to information calculated in phase 1 correspond directly to one set of m^2 calculations of this very phase 1. Therefore, by removing phase 1, we can never end up doing more work (apart from additional overhead that comes with the search), but we will usually save a lot of work (in all cases where the minimal rank equals the optimal rank or where the restricted optimum is not reachable).

As to phase 3, the way in which the constraints are tightened is as follows. Between two successive searches, the c -value corresponding to exactly one dependency is decreased by 1. After the c -value of each dependency has been decreased the algorithm wraps to the first dependency and decreases its c -value once more. If a c -value equals its minimal value it is not further decreased, but instead the algorithm continues with the next dependency. If no solution exists for a given set of constraints, either because the search space has been exhausted or because a maximal number of iterations has been reached, the latest decreased value is increased again and its minimal value is modified never to decrease it again. This process continues until all c -values equal their respective minimal values.

An obvious optimization in this respect is to check whether a constraint that has just been tightened is already satisfied by the solution found in the previous search and to continue with the next dependency if it is. Table 1 shows the (sometimes dramatic) improvement that ensues. The leftmost column shows the number of search iterations that is required after a number of other optimization had already been performed on the search algorithm and the second column shows the number when additionally skipping unnecessary searches. The rows correspond to different algorithms or different versions of the same algorithm. The new value for “Initial APP” is obvious if you consider that the best solution is to leave everything in place and that there are 10 polytopes. The table also shows the result of an additional optimization where the set of possible mappings for each polytope is reordered such that the solution found in the last search is placed first. Naturally, this mapping does not satisfy the new set of constraints, but as the numbers show, it is usually a good starting point.

The change in optimization criterion obviously also had a large effect, both on the number of iterations and on the solution. Table 2 shows the number of iterations required for all three optimization criteria discussed in earlier sections. Empty rows in the r_d column correspond to tests that have not been performed. The main aim of the table is to show that we do not have to pay a huge increase in iterations to get at a (markedly) better solution. The increase that does occur is due to the fact that the more stringent r_K -criterion cuts away more of the search tree. On the other hand, with the r_a -criterion, there are a lot more solutions.

Figures 4 and 5 compare an optimal solution found according to the r_K -criterion to an optimal solution according to the r_d -criterion. For the first solution, both the r_K and the r_d values are plotted. The first figure shows the result for the initial USVD description. As can be seen, 11 out of the

Application	Version	Search Iterations		
		intermediate	skip	reorder
Simple example	Initial	25	12	13
	Modified	25	12	13
APP	Initial	750	10	10
	Modified 1	5834	1028	1028
	Modified 2	7069	1281	1134
Updating SVD	Initial	41371	6965	6337
	Modified 1	40804	10619	6277

Table 1: Removing unnecessary searches and reordering the set of mappings

Application	Version	Search Iterations		
		r_K	r_a	r_d
Simple example	Initial	13	29	
	Modified	13	17	
APP	Initial	10	10	
	Modified 1	92	104	
	Modified 2	1036	475	
Updating SVD	Initial	2536	2056	1990
	Modified 1	2491	2000	1934

Table 2: Effect of change in optimization criterion

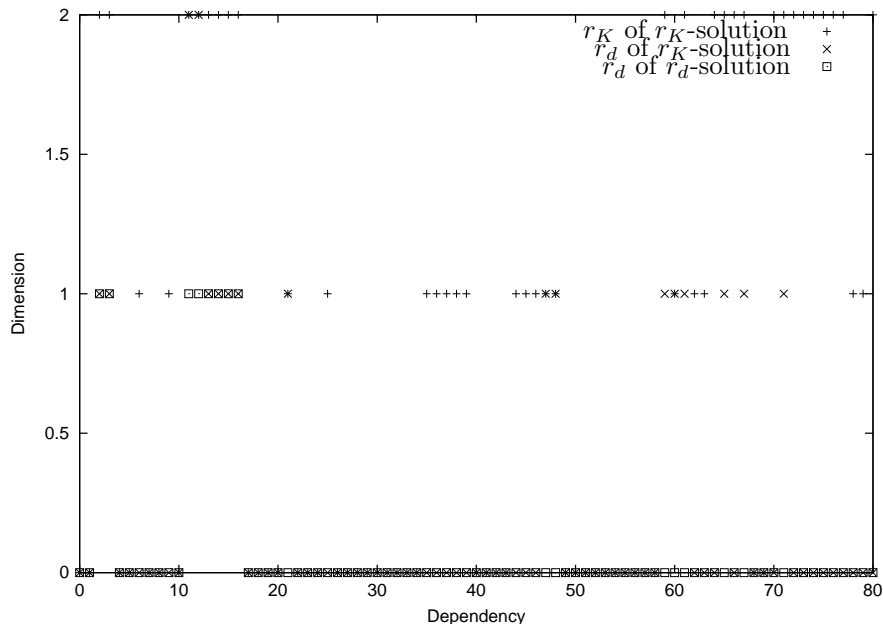


Figure 4: Initial USVD description

81 dependencies have a better dependency distance vector polytope dimension when the r_d criterion is used.

Figure 5 shows the results for a modified description.⁸ This particular description had been modified to result in better r_K -values using the technique discussed in section 3. In this case the r_K -solution is not a restricted optimal solution, because some combinations of constraints were unsatisfiable or exceeded the maximal number of iterations in solving. The figure clearly reflects this, as there are more dependencies with a high dimension among the dependencies whose constraints are tightened later (dependencies with a higher sequence number). It further shows that this solution has some better r_K -values, but that the r_d -values are a lot worse. The number of iterations required changes dramatically in this case. To obtain the (non-optimal) solution presented in the figure, a total number of 7843416 iterations was needed, whereas the r_d -solution only requires 1990, the same number as the initial description since the two are identical with respect to the r_d -criterion.

A number of smaller changes was also made to the algorithm:

- Instead of phase 2 some mappings can be removed from the initial set that are really unnecessary, namely those that only differ in multiples of $C^=$ from some other mapping already in the set. The effect of this change was rather small on our examples. We expect a bigger impact on examples where the true optimum cannot be reached.

⁸It is not the one called “Modified 1” in tables 1 and 2.

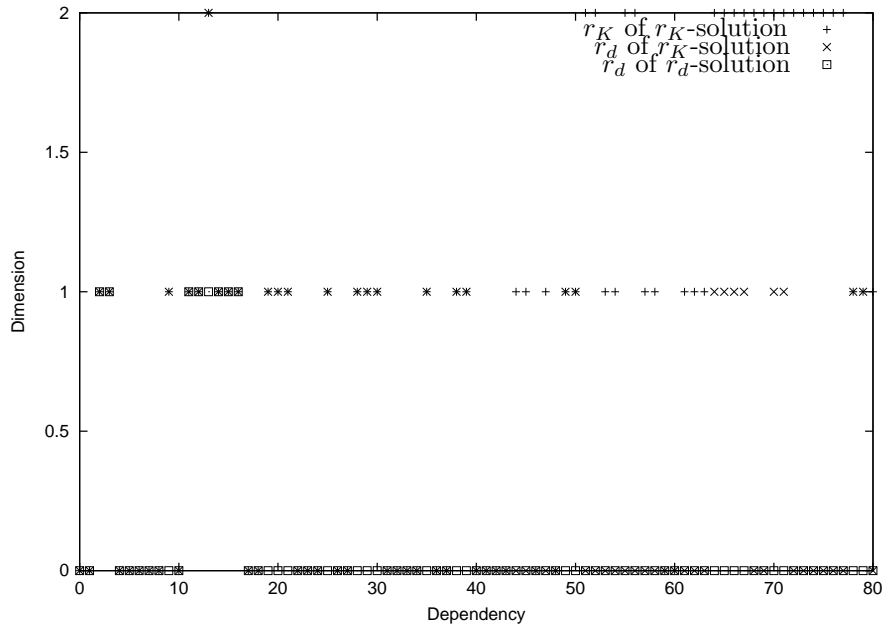


Figure 5: Modified USVD description

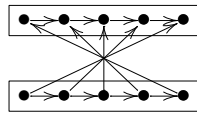


Figure 6: Self-dependencies

- Self-dependencies are ignored when it comes to minimizing the dimension. Obviously, transforming a polytope does not change the dimension of dependency distance vector polytopes of internal dependencies. Care has to be taken though that the orientation of different polytopes does not make the self-dependencies restrict the allowed ordering vectors too much. An extreme case is shown in figure 6. The dimension of the inter-polytope dependency can be made zero by mirroring one of the polytopes over the vertical axis, but this results in intra-polytope dependencies pointing in opposite directions making a valid ordering impossible.
- The algorithm performs backjumping and on a jump back, the polytopes are reordered such that the level where the jump back occurs is moved directly under the level that is jumped back to. The effect is not very large on the final algorithm for the current examples, but, again, we expect a bigger impact on examples that fail to reach the restricted optimum.
- Before initiating the search, not only are the two polytopes involved in

the tightened constraint put first, but the order of the other polytopes is made according to the number of conflicts that have occurred in previous searches.

- The original algorithm had provisions to restart the search with the polytopes reordered based on the number of conflict when some number of iterations had been performed during the search. This was changed to only be applied when the first polytope actually changes, the assumption being that all the other highly conflicting polytopes would move up automatically through the backjump reordering.

6 Greedy search

This section presents a very simple algorithm that cannot be used if the true optimum is not reachable and that is not guaranteed to succeed even if it is. The idea is to pin down one polytope (e.g., by assigning it the identity matrix as transformation matrix) and then successively increasing the set of pinned down polytopes by selecting in each step a dependency between a pinned down polytope and a floating polytope and choosing an appropriate transformation matrix for the floating polytope that satisfies the constraint for the selected dependency. At the end, the constraints for the other dependencies can simply be checked.

An appropriate transformation matrix can be directly determined using equation 6. Let us first assume that $D = J_p^{-1}J_c$ is of full rank. Then, if the floating polytope is the production polytope, we use

$$A_p = A_c D^{-1} \tag{20}$$

and if it is the consumption polytope, we use

$$A_c = A_p D. \tag{21}$$

If D itself is not of full rank but the optimal dimension is zero, then, referring to equation 19, D can be transformed into an equivalent (with respect to the optimality criterion) regular dependency matrix by adding rows from $C^=$. The final case, where the optimal dimension is strictly positive, can be dealt with by additionally adding R arbitrary rows (e.g., rows from the identity matrix), linearly independent of $C^=$, to D to make it of full rank.

It would be prudent in selecting a dependency to pick one with the least amount of freedom. That is, to take one with the smallest optimum rank and, among those, one with the smallest number of implicit equalities. At the end (or along the way), we may wish to minimize the determinants of the calculated transformation matrices as much as possible. This is discussed in appendix B.

7 Constraint propagation search

This section presents a sketch of a possible algorithm that again assumes the true optimum is reachable but that should have a higher chance of succeeding than the algorithm in the previous section. With some limited manual intervention, it works for the USVD example. Instead of just choosing one of the possible mappings that satisfies the constraint of a dependency, the constraint is converted into a constraint on a polytope and then combined with all the other generated constraints for the same polytope. If this combination differs from the original constraint it is propagated to neighbouring polytopes through the dependency constraints.

The constraint imposed on a polytope is a description of all the possible mappings for the polytope with respect to a given fixed polytope, such that for any mapping that complies with this description we corresponding mappings exist for all the neighbouring polytopes that all comply with their respective descriptions and that make sure the dependency constraints are satisfied. The current form only works for dependencies that have zero as optimal dimension. The other dependencies can be checked against the mappings that are found based on the 0D-dependencies.

We know from section 3 that, given a 0D-dependency and a mapping for one of the polytopes involved, we can find all the mappings for the other polytope. They are given by equation 20 or 21 plus any combination of the rows in the $C^=$ -matrix of the dependency. Now, if one of the polytopes, say the production polytope, is given in this form, i.e.,

$$A_p = A'_p + ML_p, \quad (22)$$

where L_p is a $l \times n$ matrix representing the rows that can be added and M any $n \times l$ matrix, we know that that A_c is given by

$$A_c = A'_p D + M_p L_p D + M_c C^= = A'_c + ML_c \quad (23)$$

with

$$L_c = \begin{bmatrix} L_p D \\ C^= \end{bmatrix},$$

which is again of the same form as (22). We call A'_c the representative and L_c the linear freedom.

If a polytope is involved in more than one 0D-dependency, then several constraints of the form (22) will be generated and we will have to combine them. Suppose we have two constraints

$$A = A_1 + ML_1 \quad \text{and} \quad A = A_2 + ML_2.$$

In order for there to exist an A satisfying both constraints an M_1 and an M_2 must exist such that

$$A_1 + M_1 L_1 = A_2 + M_2 L_2. \quad (24)$$

Furthermore, the linear freedom will be restricted to the “intersection” of L_1 and L_2 , that is, a basis for the intersection of the vector spaces generated by L_1 and L_2 . Let L_\cap be this intersection and let L_a and L_b be such that together with L_\cap they form bases for the vector spaces generated by L_1 and L_2 respectively. Then to be able to solve (24), we must have

$$A_1 - A_2 = M_\cap L_\cap + M_a L_a + M_b L_b$$

for some M_\cap , M_a and M_b and we can take $A_2 + M_a L_a$ as the new representative and the new combined constraint will be

$$A = (A_2 + M_a L_a) + M L_\cap.$$

The algorithm now works as follows. An arbitrary polytope is selected and assigned an identity representative and no linear freedom. This will be the first constrained polytope. The algorithm then iterates over all 0D-dependencies with at least one constrained polytope, generating constraints using (23) and/or its backward counterpart. If both polytopes were already constrained the newly generated constraints are combined with the existing ones and if any of the existing ones changes, these changes are propagated through the dependencies that have been considered previously. If there are any 0D-dependencies left (that do not involve any of the polytopes constrained so far), an unconstrained polytope is selected, pinned down. The algorithm then starts over until all 0D-dependencies have been used. The end result will be one or several clusters of polytopes that still need to be connected by the use of the non zero dimensional dependencies. This is a topic for future research.

8 Conclusions and future work

We have presented a heuristic for optimizing the regularity of accesses, viz. the dimension of the dependency distance vector polytope. We have presented a proof for a simple expression for evaluating this dimension and an expression for the optimal dimension. Having a simple expression for the dimension is necessary because it forms the inner core of all the search procedures and is hence frequently evaluated.

Three search procedures were discussed, each with its own set of limitations. The greedy search is exceedingly fast but only works under very special circumstances. Still, these circumstances do occur in real programs, so it is still worthwhile to at least check whether this procedure leads to a good solution.

The constraint propagation search is still rather fast and seems promising, but it has currently only been worked out for cases where most of the dependencies allow a zero optimal dimension. An additional benefit is that it will naturally produce several solutions among which an additional criterion can determine the best.

The backtracking search will produce a solution for any program, but it depends on the set of initial mappings used. The currently used set is both too

limited and too large. It will not find a solution that requires skewing with a skew factor different from 1 and for global dimensions larger than 3 the set is too large to work with. An additional disadvantage of this search procedure is its exponential running time.

The obvious next step is the translation subphase. Some initial ideas are presented in van Swaaij (1992), but in their original context, where linear subphase and translation subphase were still combined, they failed for larger applications. Further research is still required in this area.

A Finding implicit equalities

If we want to use the criteria derived in section 4, we need to be able to find the implicit equalities in a set of inequalities $\tilde{C}\vec{x} \geq \vec{0}$. A simple, but rather inefficient, way is to solve the linear program

$$\max\{\vec{c}_i^T \vec{x} \mid \tilde{C}\vec{x} \geq \vec{0}\}$$

for each row \vec{c}_i^T of \tilde{C} . If the maximum equals 0, then we know the row represents an implicit equality. More efficient techniques are apparently discussed in Telgen (1981).

Polylib (Wilde 1993) can also produce the required set of equalities. In fact, this set is part of its internal representation of polytopes.

B Minimizing determinants

The algorithms in sections 6 and 7 often have a certain choice in determining transformation matrices. Not all transformation matrices are equal, though. We prefer matrices that consist solely of integers and of those we prefer matrices with minimal determinant. This section presents a simple algorithm for decreasing the determinant of a transformation matrix when we have certain linear freedom. It can generate some large coefficients, though, so it is not ideal in its present form.

If row \vec{a}_i^T of A can be written as $\vec{a}_i^T = \vec{b}_i^T + \gamma \vec{c}_i^T$, then it is well known that

$$\det \begin{bmatrix} \vec{a}_1^T \\ \vdots \\ \vec{a}_{i-1}^T \\ \vec{a}_i^T \\ \vec{a}_{i+1}^T \\ \vdots \\ \vec{a}_n^T \end{bmatrix} = \det \begin{bmatrix} \vec{a}_1^T \\ \vdots \\ \vec{a}_{i-1}^T \\ \vec{b}_i^T \\ \vec{a}_{i+1}^T \\ \vdots \\ \vec{a}_n^T \end{bmatrix} + \gamma \det \begin{bmatrix} \vec{a}_1^T \\ \vdots \\ \vec{a}_{i-1}^T \\ \vec{c}_i^T \\ \vec{a}_{i+1}^T \\ \vdots \\ \vec{a}_n^T \end{bmatrix}.$$

We can use this if our transformation matrix is constrained by (22). Let C be A' with some row replaced by a row from L and let A be A' with the same row added

times γ instead of replacing the row from A' , then $\det A = \det A' + \gamma \det C$. If there is an integer γ such that $\det A < \det A'$ then

$$\gamma = \left\lfloor -\frac{\det A}{\det C} \right\rfloor \quad \text{or} \quad \gamma = \left\lceil -\frac{\det A}{\det C} \right\rceil$$

will minimize $\det A$ and we can replace A' by A . If there are several row in L , we can use the gcd of all the determinants of A' with the same row replaced by any of the rows in L . We can repeat this for all the rows of A' .

References

- Danckaert, K. (2001). *Loop Transformations for Data Transfer and Storage Reduction on Multiprocessor Systems*. Ph. D. thesis, Katholieke Universiteit Leuven, Leuven, Belgium.
- Danckaert, K., F. Catthoor, and H. De Man (2000, November). A preprocessing step for global loop transformations for data transfer and storage optimization. In *Compilers, Architectures and Synthesis for Embedded Systems (CASES)*, San Jose, California, United States, pp. 34–40. ACM Press.
- Feautrier, P. (1996). Automatic parallelization in the polytope model. In *The Data Parallel Programming Model*, pp. 79–103.
- Quilleré, F. and S. Rajopadhye (2000, September). Optimizing memory usage in the polyhedral model. In *ACM Transactions on Programming Languages and Systems (TOPLAS), Volume 22, Issue 5*, pp. 773–815.
- Schrijver, A. (1986). *Theory of Linear and Integer Programming*. John Wiley & Sons.
- Telgen, J. (1981). Redundancy and linear programs. *Mathematical centre tracts 137*.
- van Swaaij, M. (1992). *Data Flow Geometry: Exploiting Regularity in System-level Synthesis*. Ph. D. thesis, Katholieke Universiteit Leuven.
- van Swaaij, M., F. Franssen, F. Catthoor, and H. De Man (1992, March). Modelling data and control flow for high-level memory management. In *Proceedings of the 3rd ACM/IEEE European Design Automation Conference*, Brussels, Belgium, pp. 8–13.
- Wilde, D. K. (1993). A library for doing polyhedral operations. Technical Report 785, IRISA, Rennes, France.
<http://www.irisa.fr/EXTERNE/bibli/pi/pi785.html>.