An Access Regularity Criterion and Regularity Improvement Heuristics for Data Transfer Optimization by Global Loop Transformations

Sven Verdoolaege Koen Danckaert Francky Catthoor
Maurice Bruynooghe Gerda Janssens

Abstract

Power consumption for typical embedded multi-media applications is dominated by the storage of and the access to the large multi-dimensional arrays they manipulate. It is now well known that a design methodology for reducing power consumption and improving system performance should apply global loop transformations for increasing locality and regularity of code. Instead of trying to solve a single linear programming problem, which has been proved unfeasible for real world (multi-media) applications, the problem is better divided into several steps. Earlier work used a polyhedral model with a placement step mapping individual polytopes to a common iteration space and an ordering step.

For handling even larger applications with tens of loop nests that should be optimized together, this paper proposes to extend that approach by splitting the placement step into two substeps, the first linearly transforming the polytopes and optimizing for regularity and the second optimizing locality while assigning them an offset in the common iteration space. The main contribution is in proposing an automatable regularity improvement criterion. Improving regularity is beneficial for both power and performance because it enables an even better locality of access. We conclude with some preliminary results.

1 Introduction

Multi-media systems such as medical image processing and video compression algorithms, typically use a very large amount of data storage and transfers. This is especially a problem for embedded systems because the needed memories and bus transfers will consume a lot of power (De Man et al. 1990; Lippens et al. 1993). Wuytack et al. (1996) have shown that between 50 and 80% of the power in embedded multi-media systems is consumed by data storage and transfers (as opposed to the computations which consume much less), both for parallel and single-processor systems. Optimizing the global memory accesses of an application in a so-called Data Transfer and Storage Exploration (DTSE) step is therefore crucial for achieving low power realizations. Moreover, it also has a positive influence on the performance because it reduces the (external) bus traffic and it improves the cache hit rates (Danckaert et al. 2001).

An important factor in optimizing the global memory accesses is the improvement of data locality through loop transformations. Data locality is beneficial in two ways. First, by decreasing the distance between the first and the last access to the same data element, the life-time of that element is shortened, freeing up memory for other data and typically reducing the total memory requirement. Second, when the second access of a pair of accesses to the same data element is closer to the first, the element will typically be in a smaller and faster memory (due to the reduced life-time) that hence can be placed closer to the processor. This reduces the number of accesses to larger and slower memories that are further away from the processor in the memory hierarchy.

Many researchers have recognized the usefulness of data locality. However, in general, the algorithms they propose only perform a limited set of transformations, are of high complexity or make special assumptions on the initial code. Early research (Wolf and Lam 1991) focused on optimizing locality in single perfectly nested loops by applying a single linear transformation. This is clearly insufficient since it ignores the large buffers that exist between loop nests.

A very general way of performing loop transformation is to apply an affine transformation on each statement in a loop (Darte and Robert 1995; Feautrier 1992; Lim and Lam 1997). Although originally formulated for a single loop nest, it equally applies to multiple loop nests. Research on affine loop transformations has mainly focused on parallelization, though. A notable exception is the work of Lim et al. (2001), which first performs affine transformations for parallelization, which is in itself beneficial for locality, and then generalized blocking to further improve locality.

As do other researchers, we apply an affine transformation on each statement, but in order to reduce the complexity, we split the optimization problem into two phases, one determining the linear parts of the affine transformations and one determining the offsets of the affine transformations. This is an extension of our earlier approach (Danckaert et al. 2000) which combines our two steps into one placement step and includes an additional ordering step. This extension allows us to handle larger applications with tens of polytopes, as needed for real-life multi-media applications.

Our second phase, which we call the translation phase, is a general form of loop fusion with loop shifting. Loop fusion with loop shifting techniques typically only consider uniform dependences (Fraboulet et al. 1999) or a very limited form of non-uniform dependences (Song et al. 2001). Although our translation phase accepts any dependence, it will produce better results for dependences that are uniform or close to uniform. In order to increase the level of uniformity of dependences, we try to improve the regularity of accesses. As a (trivial) example, consider the program in Figure 1. The two loops cannot be fused since they access the elements of `a` in a different order. Simply reversing one of the two loops eliminates this obstacle. In general, a combination of reverse, loop permutation and loop skewing may have to be applied to improve

```
for (i = 0; i <= N; ++i)
    a[i] = f(i);
for (i = 0; i <= N; ++i)
    b[i] = a[N-i];
```

Figure 1: Example program with bad regularity.

regularity. An automatable criterion for steering this regularity improvement is the main contribution of this paper.

The transformations are performed on a polyhedral model which expresses algorithms as a set of recurrence equations over polyhedral shaped domains. Before a static loop program can be expressed in this way it may have to be transformed into dynamic single assignment form (Feautrier 1991), which we assume has already been performed in a previous step of our methodology. The polyhedral model has been applied in diverse optimization techniques, including automatic loop parallelization (Feautrier 1996; Darte and Vivien 1997) and memory usage optimization (Quilleré and Rajopadhye 2000; Thies et al. 2001). Tools are also available for extracting a polyhedral model from a given source program (Günz 1998) and for generating efficient loops from (the transformed) polyhedra (Quilleré et al. 2000; Bastoul 2002).

In Section 2 we will first briefly describe the dependence analysis that precedes the loop transformation in order to introduce some concepts required for understanding the remaining sections and explain our previous loop transformation approach, consisting of a placement and an ordering step. After introducing the extension of splitting the placement step, an optimization criterion for regularity is developed in Section 3 and in Section 4 it is shown how it can be used in an automated optimization tool. We conclude with some experiments in Section 5.

2 Loop Transformations in the Polyhedral Model

2.1 Dependence Analysis

We assume that all loop bounds and conditions are manifest affine combinations of the surrounding loop indices and that the programs are in dynamic single assignment form, meaning that each array element is written to at most once during the execution of the program. Under these circumstances, it is well known that a program can be represented as a set of recurrence equations of the following form for each statement in the original program:

$$\{\vec{x} \in \mathbb{Z}^n | C_k\vec{x} \geq \vec{c}\} : U_k(J_k(\vec{x})) = f_k(\ldots, V_k^i(J_k^i(\vec{x})), \ldots)$$

where $\{\vec{x} \in \mathbb{Z}^n | C_k\vec{x} \geq \vec{c}\}$ describes the polytope known as the iteration domain containing the set of surrounding loop indices for which the statement is executed. The array being written to in the statement is U_k, the arrays being read are the V_k^is and the array subscripts are represented by the index functions J_k and J_k^i. An additional assumption is that all the index functions are affine.

Transforming general programs into dynamic single assignment code typically involves the introduction of some storage overhead, but this overhead can be removed again after the loop transformations (De Greef et al. 1997; Verbauwhede et al. 1991; Quilleré and Rajopadhye 2000).

A dependence exists between two statements p and c if the "consumption" statement c reads values written by the "production" statement p, which means that $V_c^i = U_p$ for some i and that the set of elements being written to by p intersects with the set of elements being read by c. In terms of the recurrence equations for an array U_1 and some other array (U_2) defined in terms of U_1:

$$\begin{aligned} P_p &= \{\vec{x} \in \mathbb{Z}^n | C_p\vec{x} \geq \vec{c}_p\} : U_1(J_p(\vec{x})) = f_1(\ldots) \\ P_c &= \{\vec{y} \in \mathbb{Z}^n | C_c\vec{y} \geq \vec{c}_c\} : U_2(J_2(\vec{y})) = f_2(\ldots, U_1(J_c(\vec{y})), \ldots) \end{aligned}$$

the requirement is that there should exist $\vec{x}$ and $\vec{y}$ satisfying

$$C_p\vec{x} \geq \vec{c}_p \wedge C_c\vec{y} \geq \vec{c}_c \wedge J_p(\vec{x}) = J_c(\vec{y}). \tag{1}$$

Note that for simplicity, we will assume that each polytope is embedded in a space of some fixed dimension n. This may require the introduction of extra dimensions and the accompanying inequalities fixing the coordinates in the extra dimensions to some fixed values, but this will not have an influence on our heuristic. The *dependence function*, mapping iteration vectors from the consumption polytope to the iteration vector in the production polytope that writes the array element the consumption point needs, is given by:

$$\vec{x} = J_p^{-1}(J_c(\vec{y})),$$

where J_p is or can be made invertible due to the single assignment requirement. Since J_p and J_c are affine, this can be rewritten as $\vec{x} = D\vec{y} + \vec{d}$, for some $n \times n$ matrix D and some vector $\vec{d}$. With this notation, the set of vectors in P_c that depend on vectors in P_p, i.e., those that satisfy (1), is given by:

$$P_d = \{\vec{y} \in \mathbb{Z}^n | C_d\vec{y} \geq \vec{c}_d\}$$

with

$$C_d = \begin{bmatrix} C_c \\ C_p D \end{bmatrix} \qquad \text{and} \qquad \vec{c}_d = \begin{bmatrix} \vec{c}_c \\ \vec{c}_p - C_p\vec{d} \end{bmatrix}. \tag{2}$$

The set of inequalities $C_d\vec{y} \geq \vec{c}_d$ can imply some set of equalities, known as the *implicit equalities* (Schrijver 1986). That is, we write

$$P_d = \left\{\vec{y} \in \mathbb{Z}^n | G\vec{y} = \vec{g}, F\vec{y} \geq \vec{f}\right\} \tag{3}$$

with $G\vec{y} = \vec{g}$ a maximal set of linearly independent implicit equalities and $F\vec{y} \geq \vec{f}$ the set of remaining inequalities. This form can be obtained with existing techniques (Wilde 1993). The equalities of P_d are composed of those from P_c and P_p, the latter transformed by D, and of possibly additional equalities resulting from inequalities from both polytopes. The equalities of P_c and P_p in turn stem from the dimensions added by embedding the polytopes in an n-dimensional space and from conditions in the program.

2.2 Loop Transformation Steering

Our earlier loop transformation approach (Danckaert et al. 2000) then consists of a placement phase and an ordering phase. In the placement phase, each individual polytope is mapped onto the common iteration space by an affine transformation $\vec{x} \mapsto A\vec{x} + \vec{a}$ and in the ordering phase an ordering is determined in this common iteration space.

After placing both the consumption and production polytope of some dependence in the common iteration space, we can measure the distance between a production point and each depending consumption point. This *distance* $\vec{\delta}$ is the difference between the mapping through $(A_c, \vec{a}_c)$ of a consumption point $\vec{y}$ and the mapping through $(A_p, \vec{a}_p)$ of the corresponding $(D, \vec{d})$ production point.

$$\vec{\delta} = W\vec{y} + \vec{w}$$

with

$$W = A_c - A_p D \qquad \text{and} \qquad \vec{w} = \vec{a}_c - (A_p\vec{d} + \vec{a}_p). \tag{4}$$

The convex hull $\mathcal{D}$ of the set of all such distance vectors generated by a given dependence is obtained by mapping P_d (3) through the affine function $(W, \vec{w})$ and is known as the *dependence polytope* (Yang et al. 1994).

$$\mathcal{D} = \left\{ \vec{\delta} \in \mathbb{Z}^n | \exists \vec{y} \in \mathbb{Z}^n, \vec{\delta} = W\vec{y} + \vec{w}, G\vec{y} = \vec{g}, F\vec{y} \geq \vec{f} \right\} \tag{5}$$

Since prior to the ordering phase, the common iteration space has no concept of time, only geometrical information can be used in cost functions. In particular, Danckaert et al. (2000) consider three geometrical cost functions:

- For locality, the average length of the distance vectors is proposed.
- To optimize data reuse, where array elements are consumed multiple times, the length of the minimum spanning tree over all accesses (1 production and n consumptions) is used as a cost function.
- For regularity, the *dependence cone* (Yang et al. 1994) is used. It is defined as the set of all (strictly) positive combinations of distance vectors.

  $$\mathcal{C}_d = \left\{ \vec{v} \in \mathbb{Z}^n | \vec{v} = \sum_i \lambda_i \vec{\delta}_i, \vec{\delta}_i \in \mathcal{D}, \lambda_i \geq 0, \sum_i \lambda_i \geq 1 \right\}$$

  By first minimizing the dimension of the dependence cone and then its *sharpness* (Danckaert 2001), the variance in direction of the distance vectors will be minimized. This will reduce the number of dependences that cross each other leading to better locality and shorter lifetimes of the array elements involved. Furthermore, it will create more freedom for choosing ordering vectors, which in general will lead to more general and efficient solutions.

The common iteration space that results from the placement phase can be regarded as an iteration space representing a single loop nest. As such, conventional scheduling or space-time mapping techniques can be used as the ordering phase (Feautrier 1992; Lamport 1974; Lengauer 1993). In the simplest case a single ordering vector is selected for scanning all the polytopes in the common iteration space. Alternatively, a fixed ordering can be assumed in the common iteration space, effectively eliminating the ordering phase. We have shown (Verdoolaege et al. 2002) that the latter has a positive effect on the complexity of the placement phase. Although we are not restricted to using only geometrical cost functions in such a context, the concept of improving the regularity as a preprocessing for the actual loop fusion (translation) remains valid. This is illustrated in Figure 1 where the polyhedral ordering vector in both loop nests can remain identical to the one implied from the initial code, but where the mirroring (reversal) of the placement of the polytope in the second loop nest should be done prior to its further translation.

3 An Optimization Criterion for Regularity

The complexity of the placement step described in the previous section is still quite high, and cannot deal with tens of polytopes as needed for real-life multimedia applications from e.g. MPEG4 and MPEG21. Hence, we propose to split it into a linear subphase that determines the linear part A of the transformation for each of the polytopes and a translation subphase that determines the offset $\vec{a}$ of the transformation. In the first subphase only regularity is considered since locality and date reuse depend on the relative offsets of polytopes which are only determined in the second subphase. In the second subphase all three cost functions have to be considered, but the search space is reduced significantly since the linear part is fixed by the previous subphase.

For measuring the regularity we can no longer use the dimension of the dependence cone, since it depends on the relative offsets. This is, however, not the case for the dimension of the dependence polytope. To see this, consider the definition of the dependence polytope (5). It only depends on the translation offsets through the vector $\vec{w}$. Moreover $\mathcal{D} = \vec{w} + \mathcal{D}'$, with

$$\mathcal{D}' = \left\{ \vec{\delta} \in \mathbb{Z}^n | \exists \vec{y} \in \mathbb{Z}^n, \vec{\delta} = W\vec{y}, G\vec{y} = \vec{g}, F\vec{y} \geq \vec{f} \right\} \tag{6}$$

and so $\dim \mathcal{D} = \dim \mathcal{D}'$, independent of $\vec{w}$. The dimension of the dependence cone does depend on $\vec{w}$, namely it will be one more than the dimension of $\mathcal{D}$ if $\vec{w}$ is not in the column space of W and it will equal the dimension of $\mathcal{D}$ otherwise. Figure 2 illustrates this relation. In all the figures in this paper, a distance vector is depicted by an arrow between the production point and the consumption point in the common iteration space and between the origin and the vector point itself in the distance vector space.

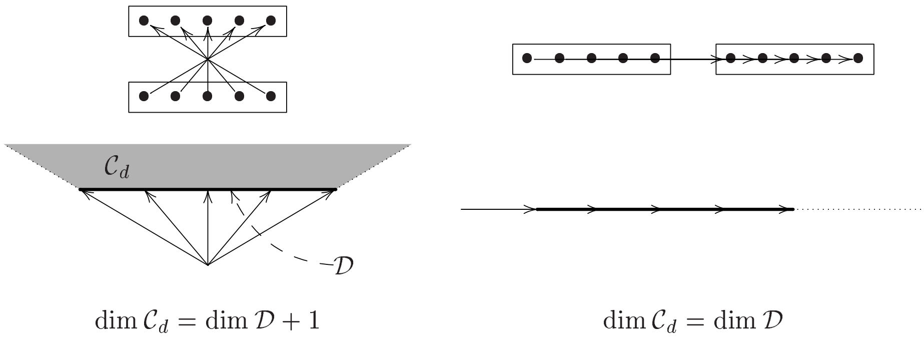


Figure 2: Dependence polytope and cone. The top part of each figure shows the common iteration space and the bottom part the distance vector space. The dimension of the dependence cone $\mathcal{C}_d$ depends on the relative offset of production and consumption polytope, which is the only difference between the figure on the left and the figure on the right, whereas the dimension of the dependence polytope $\mathcal{D}$ does not.

3.1 The Dimension of the Dependence Polytope

We now derive a simple formula for the dimension of the dependence polytope, which we will exploit in our search strategies. The dependence polytope $\mathcal{D}$ is the image of another polytope P_d under a linear transformation. Since this transformation can be singular, it can project out some dimensions and so, intuitively, the dimension of $\mathcal{D}$ equals the number of dimensions of P_d that are not projected out. This is indeed confirmed by the derivation below.

The dimension of a polytope is defined as the dimension of its affine hull (Schrijver 1986), and so, referring to (6),

$$r := \dim \mathcal{D} = \dim \mathcal{D}' = \dim \left\{ \vec{\delta} \in \mathbb{Z}^n | \exists \vec{y} \in \mathbb{Z}^n, \vec{\delta} = W\vec{y}, G\vec{y} = \vec{g} \right\}.$$

This affine hull is the image under W of the affine hull P_d'' of P_d (3).

$$P_d'' = \{ \vec{y} \in \mathbb{Z}^n | G\vec{y} = \vec{g} \} = \vec{y}_0 + \text{lin.space}\, P_d''$$

with $\vec{y}_0$ some point in P_d'' and lin.space P_d'' the lineality space of P_d'' (Schrijver 1986). The last equality holds because P_d'' is an affine hull. The dimension k of P_d'' equals the dimension n of the problem space minus the rank of the matrix defining the equalities,

$$\dim P_d'' = \dim \{ \vec{y} \in \mathbb{Z}^n | G\vec{y} = \vec{g} \} = n - \operatorname{rank} G =: k. \tag{7}$$

Let X be a matrix with a basis for the lineality space of P_d'' as columns, then

$$r = \dim \left(W\vec{y}_0 + \left\{ \vec{\delta} \in \mathbb{Z}^n | \exists \vec{v} \in \mathbb{Z}^k, \vec{\delta} = WX\vec{v} \right\} \right) \tag{8}$$

The dimension of the polyhedron in (8) is given by the dimension of the range of the linear transformation f defined by

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

For linear transformations with a k-dimensional domain, we know that

$$\dim \operatorname{Ker} f + \dim \operatorname{Ran} f = k. \tag{9}$$

The kernel of f are those vectors $\vec{y} \in P_d''$ that map to $\vec{0}$ under W, so

$$\dim \operatorname{Ker} f = \dim \left\{ \vec{y} \in \mathbb{Z}^n | \begin{bmatrix} W \\ G \end{bmatrix} \vec{y} = \begin{bmatrix} \vec{0} \\ \vec{g} \end{bmatrix} \right\} = n - \operatorname{rank} \begin{bmatrix} W \\ G \end{bmatrix}. \tag{10}$$

Substituting (10) and (7) in (9), we obtain

$$r = \dim \operatorname{Ran} f = (n - \operatorname{rank} G) - (n - \operatorname{rank} \begin{bmatrix} W \\ G \end{bmatrix}),$$

or

$$r(W) = \operatorname{rank} \begin{bmatrix} W \\ G \end{bmatrix} - \operatorname{rank} G. \tag{11}$$

This expression can be interpreted as the linear independence in the projection matrix W that is independent of the linear independence that already exists in the source polytope.

3.2 Optimal Dimension

To be able to evaluate a particular choice of transformation matrices, knowledge of the optimal value of the dimension r, which depends on the transformation matrices through W (4), is required. The optimal value is the minimal value of r over all (non-singular) transformation matrices A_c and A_p:

$$R := \min_W r(W) = \min_{A_c, A_p} r(A_c - A_p D).$$

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

$$R = \min_A r(A - D)$$

with $A = A_p^{-1} A_c$.

From (8), we know:

$$r(A - D) = \operatorname{rank}((A - D)X)$$

If we write

$$AX = (A - D)X + DX$$

then we know

$$\operatorname{rank}(AX) \leq \operatorname{rank}((A-D)X) + \operatorname{rank}(DX)$$

and so by applying (11) twice

$$r(A-D) \geq r(A) - r(D) = \operatorname{rank}\begin{bmatrix} A \\ G \end{bmatrix} - \operatorname{rank}\begin{bmatrix} D \\ G \end{bmatrix} = n - \operatorname{rank}\begin{bmatrix} D \\ G \end{bmatrix}$$

since A is non-singular.

This means we have a lower bound for R:

$$R \geq n - \operatorname{rank}\begin{bmatrix} D \\ G \end{bmatrix}.$$

The following argument shows that this lower bound is tight. Take q linearly independent rows $D_q = \{\vec{d}_{i_1} \ldots \vec{d}_{i_q}\}$ from D that are also linearly independent of the rows in G ($q = \operatorname{rank}\begin{bmatrix} D \\ G \end{bmatrix} - \operatorname{rank} G$). Let A_c be composed of $\vec{d}_{i_1} \ldots \vec{d}_{i_q}$ as the first q rows, followed by $(\operatorname{rank} G)$ linearly independent rows from G and $(n - \operatorname{rank}\begin{bmatrix} D \\ G \end{bmatrix})$ vectors that are linearly independent of any of the other rows (A_c is non-singular). Furthermore, let the first q rows of A_p be $\vec{e}_{i_1} \ldots \vec{e}_{i_q}$ and write the remaining rows of D (i.e., those not in D_q) as a linear combination of rows from D_q and G:[1]

$$\vec{d}_j = \sum_{k=1}^{q} \alpha_{jk} \vec{d}_{i_k} + \sum_{k=1}^{\operatorname{rank} G} \beta_{jk} \vec{g}_k \qquad \forall \vec{d}_j \notin D_q.$$

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 non-singular and the reader will easily verify that the first q rows of A_c and $A_p D$ are equal and that the next $(\operatorname{rank} G)$ rows of both $A_c X$ and $A_p D X$ evaluate to $\vec{0}^T$ and hence that $\operatorname{rank}((A_c - A_p D)X) \leq n - \operatorname{rank}\begin{bmatrix} D \\ G \end{bmatrix}$.

Therefore,

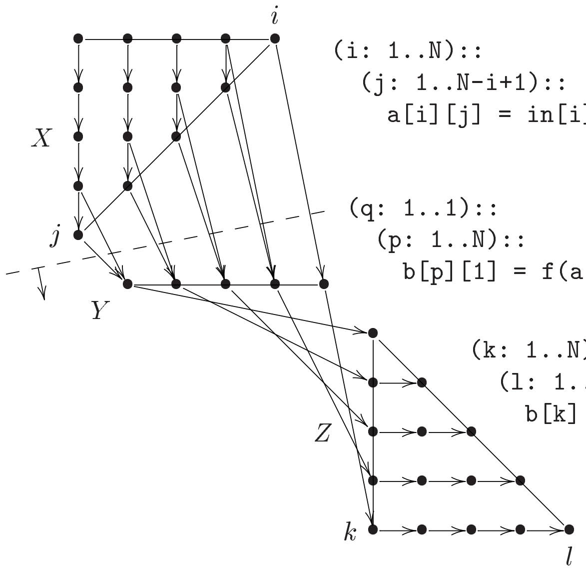
$$R = n - \operatorname{rank}\begin{bmatrix} D \\ G \end{bmatrix}. \tag{12}$$

3.3 Example

Consider the example in Figure 3 and more specifically the first dependence between polytope X writing to `a[i][j]` and Y reading from the same array through `a[N-p+1][p]`. The linear parts of the index functions are as follows:

$$J_X = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad D = (J_X)^{-1} J_Y^1 = J_Y^1 = \begin{bmatrix} 0 & -1 \\ 0 & 1 \end{bmatrix}.$$

[1] e_p is a unit vector with the 1 in the pth position.




```
(i: 1..N)::
  (j: 1..N-i+1)::
    a[i][j] = in[i][j] + a[i-1][j];

(q: 1..1)::
  (p: 1..N)::
    b[p][1] = f(a[N-p+1][p], a[N-p][p]);

(k: 1..N)::
  (l: 1..k)::
    b[k][l+1] = g(b[k][l]);
```

Figure 3: Example program with polyhedral representation. The placement and ordering shown correspond to a procedural interpretation of the code on the right. Since the ordering cuts through $2N-1$ dependences, a buffer of that size is required.

Polytope Y satisfies an equality corresponding to the extra dummy loop iterator q: $[1 \quad 0]\vec{x} = 1$, which is carried over to the P_d of this dependence, so $G = [1 \quad 0]$. According to (12), this means that the optimal dimension is

$$R = 2 - \operatorname{rank}\begin{bmatrix} 0 & -1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} = 0.$$

In general, the optimal dimension will be zero unless the dependence exhibits self-reuse.

Inspection of (11) shows that optimally all non-zero rows of W should be linear combinations of the rows in G. Assume that we do not transform polytope X, i.e., $A_X = I_2$. To make all rows of W (4) zero, we would have to take

$$A_Y = A_X D = \begin{bmatrix} 0 & -1 \\ 0 & 1 \end{bmatrix},$$

with X the producer and Y the consumer. This is a singular matrix, but it can be made non-singular by adding rows from G, without changing the dimension of the dependence polytope. E.g., adding G to the top row results in

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

In fact, since $G\vec{x}$ is constant for any $\vec{x}$ in P_Y, adding any multiple of G to any of the rows in A_Y will result in the exact same transformation for polytope Y.

Similar reasoning on the dependence between polytopes Y and Z, with index and dependence functions

$$J_Y = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \qquad D = (J_Y)^{-1} J_Z^1 = (J_Y)^{-1} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

yields the following optimal value for the transformation matrix of polytope Z:

$$A_Z = A_Y D = \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix}. \tag{13}$$

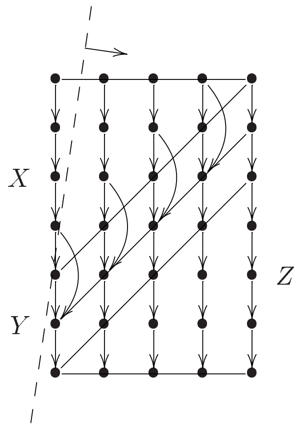
The linear transformations derived above are used in Figure 4 which also includes an appropriate translation and ordering to be determined by the subsequent steps.

The dependence between polytopes Y and Z also allows for fundamentally different optimal transformations of polytope Z. Referring to (2), the linear equalities are in this case:

$$G = G_Y D = [1 \quad 0] \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = [0 \quad 1]$$

with G_Y the implicit equalities in C_Y. Adding multiples of this row to the rows of A_Z (13) can result in the alternative

$$A'_Z = \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix} + \begin{bmatrix} -1 \\ 1 \end{bmatrix} [0 \quad 1] = \begin{bmatrix} -1 & 0 \\ 1 & 1 \end{bmatrix}.$$




```
for (j=1; j<=N; ++j) {
  for (i=1; i<=N-j+1; ++i)
    a[i][j] = in[i][j] + a[i-1][j];
  b[j][1] = f(a[N-j+1][j], a[N-j][j]);
  for (l=1; l<=j; ++l)
    b[j][l+1] = g(b[j][l]);
}
```

Figure 4: Final placement and ordering for the program in Figure 3 with corresponding C code. In this case at most two dependences are cut and so a buffer size of two is sufficient.

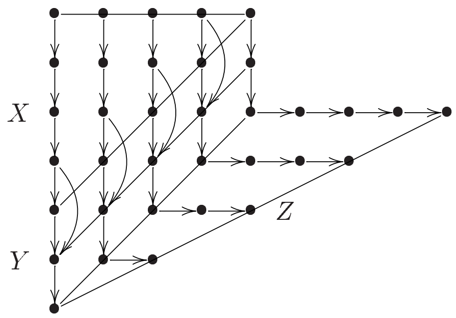


Figure 5: An alternative placement for the program in Figure 3.

A complete mapping based on this linear part is shown in Figure 5. It should be clear that if the ordering step selects a single ordering vector to be used for all polytopes or if the ordering is assumed fixed, then the mapping in Figure 4 is preferable over the one in Figure 5.

4 Search Strategies

In this section we explain how the regularity criterion can be used to find good or even optimal solutions. In all cases, self-dependences are ignored, since the dimension of the corresponding dependence cones is not influenced by applying a linear (non-singular) mapping.

4.1 No Cycles

The dependence graph is a directed graph with the statements with their iteration domain polytopes as vertices and for each dependence, an edge from the production polytope to the consumption polytope. The *undirected* dependence graph is the underlying undirected graph. If the undirected dependence graph contains no cycles, then a solution can be found trivially. After assigning an arbitrary linear transformation, e.g., the identity transformation, to the first polytope, a linear transformation that yields optimal regularity (12) can be assigned to all the other polytopes on the basis of (4) while traversing the dependence graph.

The technique to be used is as shown in the example of Section 3.3. Assuming that W can be made zero, i.e., D is of full rank, we can simply use

$$A_c = A_p D. \tag{14}$$

Otherwise, A_c can be adjusted using rows from G and, in case of self-reuse, where the optimal dimension R is strictly positive, R arbitrary rows.

It is important to realize that it is the *undirected* dependence graph which should be cycle-free. The reason is that two constraints of the form (14) can conflict. At first sight, then, this search strategy does not seem to be applicable to the example, since it has two dependences between polytopes X and Y. As far as the regularity criterion is concerned, however, the two dependences are the same, because the dependence functions are identical and the dependence domains have the same affine hull.

If the undirected dependence graph does contain cycles, the method can be used on the quotient graph based on the strongly connected components (SCCs) of the undirected dependence graph. Within an SCC one of the more complicated techniques below can be used. As a heuristic, the method could also be used on a spanning tree of an SCC. If this yields an optimal solution for the whole SCC, there is no need to use any of the other techniques.

4.2 No Self-Reuse and no Conflicts

If an SCC contains no dependence with self-reuse, then the optimal dimension (12) for each dependence is zero. Due to conflicting constraints, it may not be possible to reach this optimal dimension for each of the dependences. Even if it is possible, then the previous method may still not be able to find it, because it only selects one of the possibly many optimal transformations for a given dependence. To avoid this problem, the method in this section manipulates a compact representation of all possible transformations.

This compact representation is composed of one of the possible transformation matrices and the set of rows that can be used to create the others. We call these the representative and the linear freedom respectively. Suppose the production polytope of a given dependence has representative A'_p and linear freedom L_p, i.e.,

$$A_p = A'_p + M_p L_p, \tag{15}$$

where L_p is an $l \times n$ matrix and M_p any $n \times l$ matrix, then, using (14) and taking into account the implicit equalities of the dependence,

$$A_c = A_p' D + M_p L_p D + M_c G = A_c' + M L_c \tag{16}$$

with

$$L_c = \begin{bmatrix} L_p D \\ G \end{bmatrix} \qquad \text{and} \qquad M = [\, M_p \quad M_c \,],$$

which is again of the same form as (15). It is also possible to traverse a dependence in the opposite direction, by using

$$A_p = A_c D^{-1} \tag{17}$$

with D adjusted with rows from G in case it is singular.

If a polytope is involved in more than one dependence, then several constraints of the form (15) will have to be combined. Suppose we have two constraints

$$A = A_1 + M_1 L_1 \qquad \text{and} \qquad A = A_2 + M_2 L_2.$$

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

$$A_1 + M_1 L_1 = A_2 + M_2 L_2. \tag{18}$$

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 (18), 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 $A_2 + M_a L_a$ (or $A_1 - M_b L_b$; they only differ in a multiple of the common linear freedom) as the new representative of the linear transformation and the new combined constraint will be

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

The Ms are chosen such that the new representative is unimodular, i.e., with determinant ± 1. If no such choice can be made, this is considered to be a conflict.

To assign a unimodular transformation to each of the polytopes in the SCC, an arbitrary polytope is first selected and assigned an identity representative and no linear freedom. This will be the first constrained polytope. The algorithm then iterates over all dependences with at least one constrained polytope, generating constraints using (16) 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 dependences that have been considered previously.

This continues until all polytopes in the SCC have been considered or a conflict has been found. In the former case, the representatives can be used as transformation matrices.

As with the previous algorithm, the method can be used as a heuristic in more general cases, by first considering a subgraph, i.e., one containing only dependences without self-reuse, and afterward evaluating the solution found for the other dependences. If a conflict is found, either during the construction or during the evaluation for the other dependences, we may still be able to identify subregions of the SCC void of conflicts. The general method of the next section can then be used on the reduced graph over those subregions.

4.3 General Case

For the general case of a cyclic dependence graph with dependences exhibiting self-reuse and/or conflicting constraints, we solve a sequence of incremental backtracking searches, each searching for a better solution than the previous one.

In a preprocessing step, we select a set of possible mappings for each polytope. Initially this set $\mathbb{A}$ comprises all unimodular $n \times n$ matrices consisting entirely of 0s and 1s. For each polytope i, the set is reduced over the equivalence relation R_i:

$$\mathbb{A}_i = \mathbb{A}/R_i$$

with

$$A_1 \, R_i \, A_2 \iff A_1 - A_2 = M \cdot G_i,$$

the equivalence relation identifying all matrices that are equal up to a multiple of the linear equalities G_i for the given polytope.

Each of the backtracking searches solves a set of constraints of the following form for each of the dependences between polytopes:

$$r \leq c, \tag{19}$$

where r is the dimension of the dependence cone (11) and c is a constant. Initially all cs are set equal to the dimension of the problem n and "the solution of the previous search" is the solution corresponding to an identity mapping for each polytope. In each step, one of the cs is decreased by one. If the solution of the previous search no longer satisfies the resulting set of constraints (19), a new backtracking search is initiated over the elements of $\mathbb{A}_i$. The algorithm terminates when each c has reached the corresponding optimal value R (12). If one of the backtracking searches does not yield a solution, the c-value decreased last and its corresponding optimal value are set to the current value of c plus one. The order in which the c-values are decreased corresponds to the "importance" of the dependences. A dependence with a larger dependence domain is considered more important.

A number of optimizations are used within the backtracking search. Backjumping is performed and on a jump back, the polytopes are reordered such

Dimension		Count
Initial	Optimal	
0	0	49
1	1	2
1	0	21
2	0	3
2	1	6

Table 1: Overview of the improvement in dependence polytope dimension for the USVD algorithm. The rightmost column shows the count of dependences in the algorithm with dependence polytope dimension for the initial placement and the final placement as given by the first and second column.

that the level where the jump back occurs is moved directly under the level that is jumped back to. The polytopes are also reordered after a fixed number of iterations based on the conflicts that have occurred. This order is kept over the successive searches except that the two polytopes whose dependence constraint was tightened are placed first. The set of possible mappings is ordered such that the solution of the previous search is placed first. This will obviously not directly yield a solution, but is usually a good starting point.

The reasons for performing a sequence of backtracking searches are many. First, directly solving for the optimal dimension, may not yield a solution. Second, transferring the ordering of the polytopes based on the conflicts in one search, may greatly increase the speed of subsequent searches. Finally, if the global search takes too long, it can be interrupted and still yield a (sub-optimal) solution.

5 Experiments

We have implemented the search strategies explained in the previous section in a matlab prototype-tool and in this section we will present the results of the application of the tool to two examples: an updating singular value decomposition (USVD) algorithm and Durbin's algorithm.

The USVD algorithm is frequently used in signal processing applications. Although it is not the largest of algorithms we wish to handle, it was already too large to be handled by the approach described by Danckaert et al. (2000). It consists of four loop nests of maximally three nestings deep with statements at various levels of nesting, making it a three-dimensional problem. The polyhedral representation of the algorithm contain 26 polytopes and 87 dependences, six of which are self-dependences and hence ignored when optimizing the dimensions of the dependence polytopes.

An optimal solution is found using the backtracking search in a total number of only 1990 iterations over all searches. Each iteration corresponds to a choice of one linear mapping for one polytope. Table 1 shows the gains obtained. Eight of the dependences have an optimal dependence polytope dimension of 1, whereas the others have an optimal dimension of 0. The dimensions for the original description were suboptimal for 30 of the 81 dependences, in three cases of which the difference in dimension is even 2. Manually changing the initial algorithm by applying linear transformations, does not change the optimality of the final solution, although it can have an effect on the number of iterations required to reach it. Applying the algorithm from Section 4.2 identifies two subgraphs where it yields a valid solution. The subgraphs are linked by two dependences with self-reuse and an optimal linear mapping for the two subgraphs can be found using the greedy algorithm in Section 4.1.

The USVD example is slightly special in that an optimal solution can be found. An example of an algorithm that does not allow for an optimal solution is Durbin's algorithm, which is two-dimensional. The initial description taken from Quilleré and Rajopadhye (2000) contains six dependences with dependence polytope dimension one, two of which are optimal. Of the other four, only three can be made optimal, because there is a pair of polytopes with dependences that impose conflicting constraints on the linear mappings. The backtracking algorithm finds the best sub-optimal solution in only 50 iterations. The resulting mapping, combined with an appropriate translation and ordering allows for the same memory optimization as is discussed in the article from which the example is taken.

6 Related Work

Many researchers have investigated automatic loop transformations. The initial individual loop transformations were generalized to linear (mostly unimodular) transformations for perfectly nested loop nests and later to affine transformations for each statement in one or multiple loop nests. Most of this research has focused on parallelism (Darte and Robert 1995; Feautrier 1992; Lim and Lam 1997), although recently locality has also received some attention (Lim, Liao, and Lam 2001). Most of these approaches use exact dependence information, which leads to relatively expensive algorithms and limits the scope of applicability.

Rather than splitting up affine transformations into a linear and a translation step as we do, most researchers build up the affine transformation matrices row by row. One such example is the research by Darte and Vivien (1997). Although the heuristic they propose is oriented towards parallelism, their approach is similar to ours since they also use the distance vector abstraction.

Danckaert et al. (2000) propose an approach that consists of an affine transformation of each statement to a common iteration space, followed by an additional linear transformation of this common iteration space, called the ordering phase. We have shown (Verdoolaege et al. 2002) that the first step can be split into a linear and a translation substep and that this translation substep can be performed incrementally, even in this setting with an extra ordering phase. However, we also show that removing the ordering phase, by assuming a fixed ordering in the common iteration space, drastically reduces the complexity of incremental translation and converts it into a generally applicable loop fusion.

As to linear transformations, most researchers only apply a single linear transformation to a whole loop nest, which is often even restricted to being perfectly nested. Moreover, they typically handle loop nests individually. Wolf and Lam (1991) further only consider reuse between uniformly generated references, the number of which can be increased by using our regularity optimization technique. More recently, Kandemir et al. (2001) do not require perfectly nested loops, but report that their algorithm does not obtain very good results for imperfectly nested loops. They only consider self-reuse, whereas the technique in this paper is mostly concerned with group-reuse. An entirely different approach is taken by Olsen and Gao (1992). They do optimize across different loop nests, but they only apply a very limited set of linear transformations, viz. loop reverse and the permutation of two loop levels, and they also require perfectly nested loops.

7 Current Status and Future Work

We have presented a regularity criterion for use in the first step of a two-step global loop transformation approach. The first step linearly transforms individual iteration domains to optimize regularity and the second packs the linearly transformed polytope in a common iteration space mainly optimizing for locality. Verdoolaege et al. (2002) present more details on this second step.

Both steps have been implemented in prototype-tools as part of a precompiler transforming C-code into C-code. To obtain a polyhedral representation, the iteration domains and access functions are extracted from C-code using an in-house tool and after transformation code is generated using publicly available tools (Quilleré et al. 2000). Preliminary experiments have produced promising results for both of the steps, as, for the linear step, we obtain optimal regularity where it is possible and best sub-optimal regularity otherwise on some small to mid-sized problems.

Although the regularity criterion was originally conceived to be used in the context of a subsequent ordering step (Danckaert 2001; Verdoolaege et al. 2001), it is still relevant even in its absence. However, it is clearly not sufficient to steer the linear transformation. Rather, it should be integrated with techniques such as those of Kandemir et al. (2001) to also take into account self-reuse more directly. This integration will also influence the search methods, which we have so far only applied separately and not in the combination explained in the text.

When a full implementation is available, the complete methodology will have to be compared with other (affine) loop transformation methods in terms of scope, speed and quality of the results obtained.

References

Bastoul, C. (2002). Generating loops for scanning polyhedra. Technical Report 2002/23, Versailles University.

Danckaert, K. (2001). *Loop transformations for data transfer and storage reduction on multiprocessor systems*. Ph. D. thesis, KU Leuven.

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.

Danckaert, K., C. Kulkarni, F. Catthoor, H. De Man, and V. Tiwari (2001, January). A systematic approach for system bus load reduction applied to medical imaging. In *Proc. IEEE Int. Conf. on VLSI Design*, Bangalore, India.

Darte, A. and Y. Robert (1995). Affine-by-statement scheduling of uniform and affine loop nests over parametric domains. *Journal of Parallel and Distributed Computing 29*(1), 43–59.

Darte, A. and F. Vivien (1997). Optimal fine and medium grain parallelism detection in polyhedral reduced dependence graphs. *International Journal of Parallel Programming 25*(6), 447–496.

De Greef, E., F. Catthoor, and H. De Man (1997). Memory size reduction through storage order optimization for embedded parallel multimedia applications. *Parallel Computing 23*(12), 1811–1837.

De Man, H., F. Catthoor, G. Goossens, J. Vanhoof, J. Van Meerbergen, S. Note, and J. Huisken (1990, February). Architecture-driven synthesis techniques for VLSI implementation of DSP algorithms. *Proc. of the IEEE, special issue on "The future of computer-aided Design" 78*(2), 319–335.

Feautrier, P. (1991). Dataflow analysis of array and scalar references. *International Journal of Parallel Programming 20*(1), 23–53.

Feautrier, P. (1992, October). Some efficient solutions to the affine scheduling problem. Part I. One-dimensional time. *International Journal of Parallel Programming 21*(5), 313–348.

Feautrier, P. (1996). Automatic parallelization in the polytope model. In *The Data Parallel Programming Model*, pp. 79–103.

Fraboulet, A., G. Huard, and A. Mignotte (1999, November). Loop Alignment for Memory Accesses Optimization. In *Twelfth International Symposium on System Synthesis Proceedings (ISSS'99)*, pp. 71–77. IEEE Computer Society Press.

Günz, R. (1998, August). The new loopo scanner and parser. http://www.infosun.fmi.uni-passau.de/cl/loopo/doc/guenz-p.ps.gz.

Kandemir, M. T., J. Ramanujam, A. N. Choudhary, and P. Banerjee (2001). A layout-conscious iteration space transformation technique. *IEEE Transactions on Computers 50*(12), 1321–1335.

Lamport, L. (1974, February). The parallel execution of DO loops. *Communications of the ACM 17*(2), 83–93.

Lengauer, C. (1993). Loop parallelization in the polytope model. In *International Conference on Concurrency Theory*, pp. 398–416.

Lim, A. W. and M. S. Lam (1997). Maximizing parallelism and minimizing synchronization with affine transforms. In *Proceedings of the Twenty-fourth Annual ACM Symposium on the Principles of Programming Languages*, Paris, France, pp. 201–214. ACM Press.

Lim, A. W., S.-W. Liao, and M. S. Lam (2001). Blocking and array contraction across arbitrarily nested loops using affine partitioning. *ACM SIGPLAN Notices 36*(7), 103–112.

Lippens, P., J. van Meerbergen, W. Verhaegh, and A. van der Werf (1993, November). Allocation of multiport memories for hierarchical data streams. In *Proceedings of the IEEE/ACM international conference on Computer-aided design*, Santa Clara, California, United States, pp. 728–735. IEEE Computer Society Press.

Olsen, R. and G. Gao (1992, April). Collective analysis and transformation of loop clusters. Technical Report ACAPS Technical Memo 24, McGill University.

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.

Quilleré, F., S. Rajopadhye, and D. Wilde (2000, October). Generation of efficient nested loops from polyhedra. In *International Journal of Parallel Programming, vol 28, no 5.*

Schrijver, A. (1986). *Theory of linear and integer programming.* John Wiley & Sons.

Song, Y., R. Xu, C. Wang, and Z. Li (2001). Data locality enhancement by memory reduction. In *International Conference on Supercomputing*, pp. 50–64.

Thies, W., F. Vivien, J. Sheldon, and S. P. Amarasinghe (2001). A unified framework for schedule and storage optimization. In *SIGPLAN Conference on Programming Language Design and Implementation*, pp. 232–242.

Verbauwhede, I., F. Catthoor, J. Vandewalle, and H. De Man (1991). In-place memory management of algebraic algorithms on application-specific IC's. *Journal of VLSI signal processing 3*, 193–200.

Verdoolaege, S., F. Catthoor, M. Bruynooghe, and G. Janssens (2001, November). A heuristic for improving the regularity of accesses by global loop transformations in the polyhedral model. Report CW 325, Department of Computer Science, K.U.Leuven, Leuven, Belgium.

Verdoolaege, S., F. Catthoor, M. Bruynooghe, and G. Janssens (2002, October). Feasibility of incremental translation. Report CW 348, Department of Computer Science, K.U.Leuven, Leuven, Belgium.

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.

Wolf, M. E. and M. S. Lam (1991, June). A Data Locality Optimizing Algorithm. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'91)*, pp. 30–44.

Wuytack, S., F. Catthoor, L. Nachtergaele, and H. De Man (1996, August). Power exploration for data dominated video applications. In *Proc. IEEE Intnl. Symp. on Low Power Design*, Monterey CA, pp. 359–364.

Yang, Y.-Q., C. Ancourt, and F. Irigoin (1994). Minimal data dependence abstractions for loop transformations. In *Languages and Compilers for Parallel Computing*, pp. 201–216.