INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTÈMES ALÉATOIRES

PUBLICATION INTERNE

N° 830

LOOP NEST SYNTHESIS USING THE POLYHEDRAL LIBRARY

HERVÉ LE VERGE, VINCENT VAN DONGEN AND DORAN K. WILDE

IRISA

CAMPUS UNIVERSITAIRE DE BEAULIEU - 35042 RENNES CEDEX - FRANCE

IRISA
INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTÈMES ALÉATOIRES
Campus de Beaulieu – 35042 Rennes Cedex – France
Tél. : (33) 99 84 71 00 – Fax : (33) 99 84 71 71

Loop Nest Synthesis using the Polyhedral Library

Hervé Le Verge, Vincent Van Dongen* and Doran K. Wilde**

Programme 1 — Architectures parallèles, bases de données, réseaux et systèmes distribués
Projet API

Publication interne n° 830 — May 1994 — 7 pages

Abstract: A new method to synthesis loop nests given a polyhedral domain, the context domain, and the loop nesting order is described. The method is based on functions in the IRISA polyhedral library.

Key-words: Polyhedral scanning problem

(Résumé : tsvp)

*email: `vandonge@drebin.crim.ca` Working at CRIM (Centre de recherche Informatique de Montréal), 1801 Mc Gill college suite 800, H3A 2N4, Montreal, Canada

**email: `wilde@irisa.fr`This work was partially supported by the Esprit Basic Research Action NANA 2, Number 6632 and by NSF Grant No. MIP-910852.

Centre National de la Recherche Scientifique
(URA 227) Université de Rennes 1 – Insa de Rennes

INRIA
Institut National de Recherche en Informatique
et en Automatique – unité de recherche de Rennes

La synthèse de nids de boucles avec la bibliothèque polyédrique

Résumé : Une nouvelle méthode de synthèse de nids de boucles est décrite à partir d'un domaine polyédrique, d'un domaine contextuel, et de l'ordre sur les boucles imbriquées. Cette méthode est basée sur les fonctions de la bibliothèque polyédrique de l'IRISA.

Mots-clé : Problème du parcours d'un polyèdre

1 Introduction

The spatial point of view of a loop nest goes back to the work of Kuck [7] who showed that the domain of nested loops with affine lower and upper bounds can be described in terms of a polyhedron. Loop nest synthesis grew out of the older loop transformation theory [6, 11] where it was shown that all loop transformations could be performed by doing a change of basis on the underlying index domain, followed by a rescanning (perhaps in a different order) of the domain. Loop nest synthesis is based on the polyhedral scanning problem which poses the problem of finding a set of nested do–loops which visit each integral point in a polyhedron.

Ancourt et al. [1] were the first to solve the polyhedral scanning problem. They used a method to compute the loop nests which is based on a Fourier-Motzkin pairwise elimination procedure. This method involves the projection of polyhedron along an axis to find the loop bounds. The main difficulty is that the Fourier–Motzkin elimination method creates redundant bound equations which must be eliminated afterward. Le Fur et al. [8] also use this method for their Pandore II compiler.

The traversal of a polyhedron by a set of nested loops can be thought of as a lexicographical ordering of the integer points in the polyhedron, where a point a is executed before a point b if $a \prec b$ (a precedes b lexicographically). Thus given a polyhedron, loop bound expressions can be derived by finding the lexical minimum and maximum of the polyhedron in a given set of directions and in terms of parameters and outer loop variables. A technique to do this uses the Parametric Integer Program (PIP) developed by Feautrier [5, 4]. PIP finds the lexicographic minimum of the set of integer points which lie inside a convex polyhedron which depends linearly on one or more parameters. PIP is called twice for each loop in the loop nest, once for the lower and once for the upper bound. The loop expressions must then be extracted out of the PIP output, which is a quasi-affine expression tree, where each branch is guarded by a constraint and the terminal nodes hold either an index expression or $\perp$ (bottom) meaning that that branch is infeasible. The extraction of the loop bound expressions from PIP output is not an easy problem and requires post processing. Collard et al.[3] show how PIP can be used to find loop bounds and how PIP output can be simplified. Chamski [2] reviews the PIP method of finding loop bounds and gives timing comparisons between the PIP method and the Fourier-Motzkin method.

In this paper, we describe a method to scan parameterized polyhedra using the polyhedral library [10] which is based on the computation of the dual representation of a polyhedron. The proposed method uses the library function *DomainAddRays* to project out inner loop indices in a manner similar to the Fourier–Motzkin elimination method, but without generating any redundant inequalities. The expressions derived from the projection are further reduced by considering the *context domain* of each expression, and eliminating preestablished conditions using the library *DomainSimplify* function. Thus, we will show how the synthesis of loop nests can be nicely done using the polyhedral library.

2 The Polyhedron Scanning Problem

2.1 Introduction to parameterized polyhedra

This section quickly introduces the concept of polyhedra and parameterized polyhedra to help in the understanding of the polyhedron scanning problem. A polyhedron is defined to be the set of points bounded by a set of hyperplanes. Each hyperplane is associated with an affine inequality ($ax \geq b$) which divides space into two halfspaces: a closed halfspace which satisfies the inequality and an open halfspace which does not. A system of such inequalities induces a polyhedron $\mathcal{D} = \{\ x\ :\ Ax \geq b\ \}$ where A and b are a constant matrix and vector respectively.

Often we are interested in describing a whole family of polyhedra $\mathcal{D}(p)$, one polyhedron per instance of the parameters p. This can be done by replacing vector b above with an affine combination of a set of parameters p. By so doing, we obtain a parameterized polyhedron:

$$\mathcal{D}(p) = \{\ x\ :\ Ax \geq Bp + b\ \}$$

where A and B are constant matrices and b is a constant vector. This parameterized polyhedron can be rewritten in the form of a non-parameterized polyhedron in the combined data and parameter space as:

$$\begin{aligned} \mathcal{D}(p) &= \{\ x\ :\ \begin{pmatrix} A & -B \end{pmatrix} \begin{pmatrix} x \\ p \end{pmatrix} \geq b\ \} \\ \mathcal{D}' &= \{\ \begin{pmatrix} x \\ p \end{pmatrix}\ :\ A' \begin{pmatrix} x \\ p \end{pmatrix} \geq b\ \} \end{aligned}$$

2.2 The polyhedron scanning problem

To generate sequential code for operations and variables declared over polyhedra, a loop nest which scans the given polyhedral region must be generated. The *polyhedron scanning problem* is formally stated as:

Given a parameterized polyhedral domain $\mathcal{D}(p)$ in terms of a parameter vector p and a set of k constraints:

$$\mathcal{D}(p) = \{\ x\ :\ Ax \geq Bp + b\ \}$$

where A and B are constant matrices of size $k \times n$ and $k \times m$ respectively, and b is a constant k-vector,

produce the set of loop bound expressions $L_1, U_1, \cdots L_n, U_n$ such that loop nest:

$$
\begin{array}{lll}
\texttt{DO} & x_1 = L_1, U_1 & \\
 & \vdots & \\
 & \texttt{DO} & x_n = L_n, U_n \\
 & & \text{body} \\
 & \texttt{END} & \\
\texttt{END} & &
\end{array}
$$

will visit once and only once all integer points in the domain $\mathcal{D}(p)$ in lexicographic order of the elements of $x = (x_1, \cdots, x_n)$.

When talking about a particular loop variable x_i, we use the terminology *outer loops* to refer to loops which enclose the x_i–loop, that is, the loops of variables $x_j, j < i$. We use *inner loops* to refer to the loops contained in the x_i–loop, that is, the loops of variables $x_j, j > i$.

The problem of finding loop bounds is related to the linear programming problem and shares its complexity. Fortunately, these problems tend to be relatively small (in terms of the dimension and number of constraints) due to the fact that loops are not deeply nested, and exact solutions for typical problems can be found in reasonable time.

3 Example

Given the domain defined as :

```
{i,j,k,N,M | i>=0; -i+M>=0; j>=0; -j+N>=0; k>=0; i+j-k>=0}:S
```

and the context domain `{i,j,k,N,M | N>0; M>0}` describing what is known to be true a priori, the following four different loop nests were generated by calling the proposed procedure four times, each time scanning the domain in a different order.

```
{i,j,k,N,M | 0<=i<=M} :
   {i,j,k,N,M | 0<=j<=N} :
      {i,j,k,N,M | 0<=k<=i+j} : S
```

a. The loop nest in {i, j, k} scan order.

```
{i,j,k,N,M | 0<=j<=N} :
   {i,j,k,N,M | 0<=k<=j+M} :
      {i,j,k,N,M | 0<=i<=M; i>=k-j} : S
```

b. The loop nest in {j, k, i} scan order.

```
{i,j,k,N,M | 0<=k<=N+M>=0} :
   {i,j,k,N,M | 0<=j<=N; j>=k-M} :
      {i,j,k,N,M | 0<=i<=M; i>=k-j} : S
```

c. The loop nest in {k, j, i} scan order.

```
{i,j,k,N,M | 0<=i<=M} :
   {i,j,k,N,M | 0<=k<=i+N} :
      {i,j,k,N,M | 0<=j<=N; j>=k-i} : S
```

d. The loop nest in {i, k, j} scan order.

4 Implementation

The implementation of the proposed method is based on the polyhedral library[10].

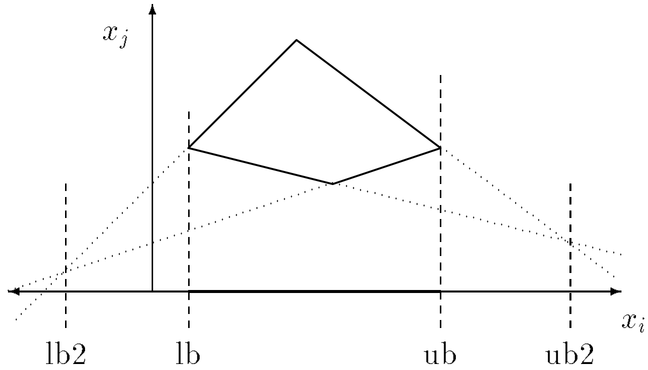

Figure 1: Projection using Fourier–Motzkin Elimination

4.1 Basis of method

This method resembles the Fourier–Motzkin (FM) method in that it projects the polyhedron in the canonical direction of inner loop variables in order to eliminate dependencies on them. Thus bounds on a loop are found independent of inner loop indices. However, the FM method considers all pairs of constraints when finding the bound on the projection. In figure 1, for example, lower bounds $lb2$ and lb and upper bounds $ub2$ and ub are all considered by the FM method. In general, given n constraints, as many as $\frac{n^2}{4}$ loop bounds could be generated in eliminating a single variable, and this number grows exponentially with the number of variables. Most of these bounds turn out to be redundant (non-tight) and must be eliminated. The elimination of these redundant loop bounds is a significant problem when using the FM method.

The proposed method is based on the double description by Motzkin[1] [9] which only considers pairs of constraints that are *adjacent*, and therefore never generates any redundant bounds (like $lb2$ and $ub2$ in the example). Thus the redundant bound elimination phase of the FM method is replaced with an adjacency test on pairs of constraints in the proposed method. This test is accomplished very efficiently assuming the rays and vertices of the polyhedron are known. The algorithm maintains both the constraint and the ray/vertex representation of a polyhedron allowing the employ of the adjacency test. Keeping the dual representation is only feasible for small dimensional domains since a d–polyhedron with n constraints might have as many as $d^{\lfloor \frac{n}{2} \rfloor}$ rays/vertices, in the worst case. We rely on the fact that computational domains tend to be relatively small polyhedra (in terms of the dimension and number of constraints) due to the fact that loops are not deeply nested.

4.2 Add Rays to a Domain

The library function *DomainAddRays* joins a set of lines, rays, or points to a domain and produces the resulting domain with all redundancies eliminated. It is used in this paper to

[1] sometime erroneously attributed to Chernikova

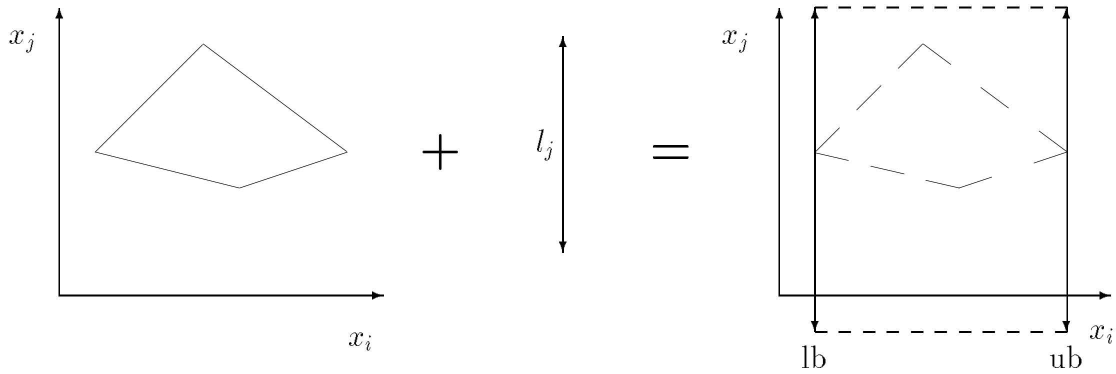

Figure 2: Adding a line to project out an index

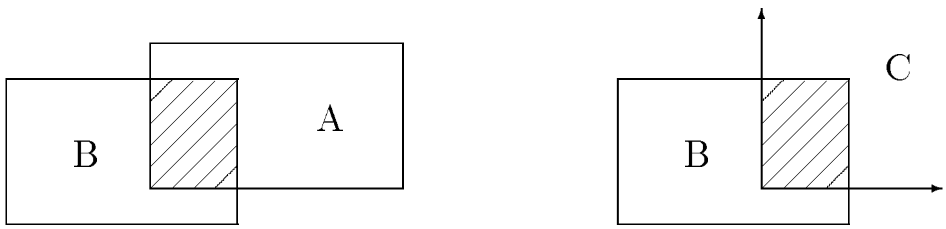

Figure 3: Domain Simplification

eliminate (or project out) the inner loop indices from the bound expressions of outer loops. This is illustrated in figure 2 where x_i is an outer loop variable and x_j is an inner loop variable. To compute the loop bounds for the x_i–loop as a function of parameters and outer loop variables, the inner loop variables $x_j, j > i$ must be removed from the domain. This is done by projecting the scan domain in the direction of the inner loop variables onto the x_i-axis, giving *lb* and *ub* as the lower and upper bounds of x_i, respectively. This can be accomplished using the polyhedral library by adding lines $\{l_{i+1}, l_{i+2}, \cdots\}$ in the direction of all of the inner loop variables $\{x_{i+1}, x_{i+2}, \cdots\}$. The resulting polyhedron is a cylinder open in the direction of inner loop variables (as shown in the figure) which has no constraints in terms of the inner loop variables.

4.3 Domain Simplify

The function *simplify in context* called *DomainSimplify* in the library is defined as follows:

> Given domains A and B, then Simplify(A, B) = C, when $C \cap B = A \cap B$, $C \supseteq A$ and there does not exist any other domain $C' \supset C$ such that $C' \cap B = A \cap B$.

The domain B is called the *context*. The simplify function therefore finds the largest domain set (or smallest list of constraints) that, when intersected with the context B is equal to $A \cap B$. The simplify operation is done by computing the intersection $A \cap B$ and while doing so, recording which constraints of A are “redundant” with result of the intersection. The result of the simplify operation is then the domain A with the "redundant" constraints removed. A simple of example is shown in figure 3. In the example, domain A is simplified (resulting in domain C) by eliminating the two constraints that are redundant with context domain B.

4.4 Loop Separation

Using the above two library functions, a function can be written which takes a specified domain $\mathcal{D}$ and separates (or factors) it into an intersection of the initial context domain $\mathcal{D}_0$ and a sequence of loop domains $\mathcal{D}_1, \mathcal{D}_2, \cdots (\mathcal{D} = \mathcal{D}_0 \cap \mathcal{D}_1 \cap \mathcal{D}_2 \cap \cdots)$ where each loop domain is not a function of inner loop variables. Accordingly, the loop domain $\mathcal{D}_i,\ i \geq 1$ can be recursively computed as:

$$\mathcal{D}_i = \mathit{DomainSimplify}(\ \mathit{DomainAddRays}(\ \mathcal{D}, \{l_{i+1}, l_{i+2}, \cdots\}\ ),\ \mathcal{D}_0 \cap \cdots \mathcal{D}_{i-1}\ );$$

5 Conclusion

We have implemented the procedure described in this paper, and tested it on a number of examples (kindly provided by Marc Le Fur). Its difficult to fairly compare the FM method with the method described here, because of the differences in implementation. The FM method programmed by Marc Le Fur [8] is based on CAML, an interpreted functional language, where as the method described here is programmed in C. Testing has shown about two orders of magnitude difference in run time, however, this is no doubt due to the implementation differences.

Some examples have been found to cause problems in the polyhedral library. Two different problems have been encountered. The first is an numeric overflow problem. The polyhedral library performs exact rational computation, and numbers are stored using 32 bit integer numerators and denominators. If two rational numbers are multiplied, and there is no cancelation, then the storage requirement for the result is the sum of the storage for the two operands (measured in number of bits). The solution to this problem is either to use a *multi-precision arithmetic package* in which storage grows to meet demand, or to integerize the vertices by adding additional constraints which "chop off" non integral vertices without excluding any of the integral points inside the polyhedron.

The second problem is a memory overflow problem. Given a d dimensional polyhedron with n constraints, as many as $n^{\lfloor \frac{d}{2} \rfloor}$ vertices could be required in the dual representation. This effectively limits computation to small dimensional polyhedra. This problem is aggravated by the fact that the current implementation allocates a fixed amount of work space to perform a computation. A dynamic work space would be better in light of this problem.

On the sunnier side, this method has several advantages. First of all, this method produces well minimized results in a convenient form. The implementation is very straight forward, using procedures from the polyhedral library. Since no redundant bounds are generated, as in the Fourier–Motzkin method, we expect this method to be more efficient.

Future experimentation is needed to empirically compare this method to the other two known methods.

Acknowledgments

We thank Marc Le Fur for his collaboration and the examples he gave us.

References

[1] C. Ancourt and F. Irigoin. Scanning polyhedra with DO loops. In *Third Symposium on Principles and Practice of Parallel Programming (PPoPP)*, pages 39–50, ACM SIGPLAN, ACM Press, 1991.

[2] Zbigniew Chamski. *Fast and Efficient Generation of Loop Bounds.* Technical Report Internal Publication 771, IRISA, Rennes, France, Oct 1993.

[3] J.-F. Collard, P. Feautrier, and T. Risset. *Construction of DO loops from systems of affine constraints.* Technical Report 93–15, Ecole Normale Supérieure de Lyon, May 1993.

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

[5] P. Feautrier. Parametric integer programming. *RAIRO Recherche Opérationelle*, 22(3):243–268, Sep 1988.

[6] F. Irigoin. *Code generation for the hyperplane method and for loop interchange.* Technical Report ENSMP-CAI-88-E102/CAI/I, Ecole Nationale Superieure des Mines de Paris, Oct 1988.

[7] D. J. Kuck. *The Structure of Computers and Computations.* J. Wiley and Sons, NY, 1978.

[8] M. Le Fur, J.-L. Pazat, and F. André. *Static Domain Analysis for Compiling Commutative Loop Nests.* Technical Report 757, IRISA, September 1993.

[9] T. S. Motzkin, H. Raiffa, G. L. Thompson, and R. M. Thrall. The double description method. *Theodore S. Motzkin: Selected Papers*, 1953.

[10] D. Wilde. *A Library for Doing Polyhedral Operations.* Technical Report Internal Publication 785, IRISA, Rennes, France, Dec 1993.

[11] M.E. Wolf and M. Lam. Loop transformation theory and an algorithm to maximize parallelism. *IEEE Transactions on Parallel and Distributed Systems*, 2(4):452–471, Oct 1991.