Scanning a Union of Z-Polyhedra

Gautam Gupta

COSI

The Scanning Problem

To visit each point of the domain of a variable in lexicographic order

We shall consider domains expressed as a union of Z-Polyhedra

Problem Context: Automated Code-Generation

Representation of a Z-Polyhedra

1. Intersection of a Polyhedra with a lattice

2. Affine image of a polyhedra with the lattice defining vectors

   Canonical Representation: This Transformation matrix is in Hermite Normal Form

Knowns

Fabiens Algorithm: Solves the scanning problem for a union of Polyhedra

Scanning a Single Z-Polyhedra

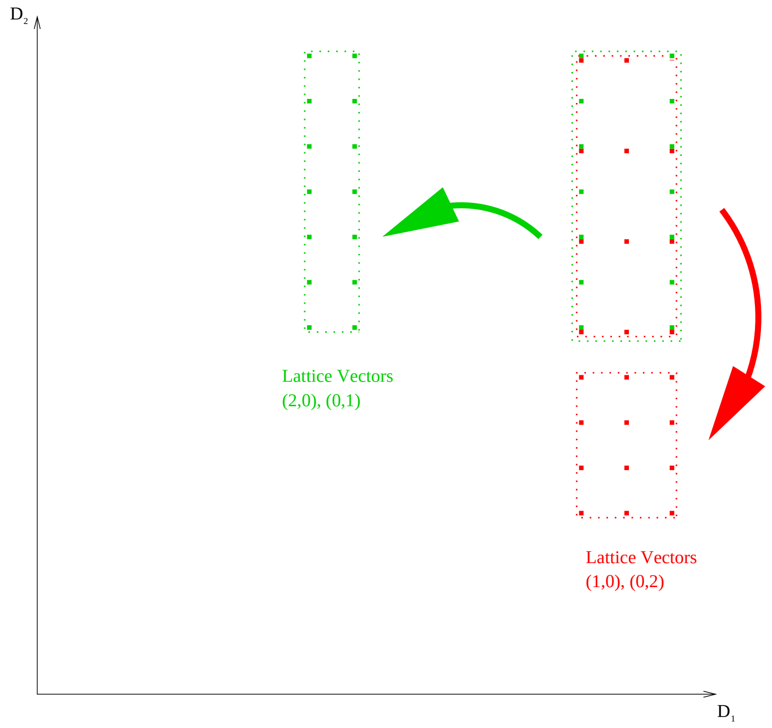
- The Z-Polyhedra is given in its canonical form
- Scan its affine preimage: This is a dense polyhedra
- Every point in the preimage corresponds to a point in the Z-Polyhedra. We can get our point by applying the Transformation

Extensions to a union of Z-Polyhedra

Problems with extending the previous method: An affine transformation will take 2 Z-Polyhedra with the same envelop to disjoint preimages.

Extensions to a union of Z-Polyhedra

Problems with extending the previous method: An affine transformation will take 2 Z-Polyhedra with the same envelop to disjoint preimages.



Our Method

Top Level View

Break up the canonical approach to applying one vector at a time

Instead of scanning the preimage, we scan the envelop of the Z-Polyhedra but apply a step to jump over the holes

Thus, we use the representation in which the Z-Polyhedra is the intersection of a Polyhedra with a Lattice

Our Method (Steps)

[1 Z-Polyhedra]

1. Use of Fabien's Algorithm to generate the loop nest that scans the union of Polyhedra (Lattice Neglected for this step)

2. Transform Lattice vectors to their canonical form

β_1 is along D_1 only

β_2 is along D_1 and D_2

...

β_n is along $D_1 ... D_n$

where,

D_1 *is the innermost dimension, ..., D_n is the outermost dimension*

3. Incorporation of holes in the loop nest

- involves changes in ...

- Loop Initialization
- Loop Increment
- Bound Testing (remains the same)

We will replace For-loops by While-loops

for (init$_2$, test$_2$, incr$_2$)
{
   for (init$_1$, test$_1$, incr$_1$)
   {
      CODE
   }
}

init'$_2$ MODIFICATION
while (test$_2$)
{
   init'$_1$
   while (test$_1$)
   {
      CODE
      incr'$_1$
   }
   incr'$_2$ MODIFICATION
}

We will proceed from outer loop to inner loops and change the initialization and increment functions at each step

We will follow the following convention:

- ID_1, ID_2, ID_3,..., ID_n are loop indices corresponding to dimensions D_1, D_2, D_3,..., D_n
- init' and incr' are modified functions

Loop Initialization

At loop K (from the inside), ID_k depends on $ID_{k+1}...ID_n$

- We calculate $ID_{K_{orig}}$ according to init
- Let us introduce another value $prev_k$
  - $prev_k$ is the distance we have already travelled along the k^{th} dimension, by moving along vectors $\beta_{k+1},...,\beta_n$
- $ID_k = \left\lceil \frac{ID_{k_{orig}} - prev_k}{c_k} \right\rceil \times c_k + prev_k$

  where c_k is the k^{th} component of β_k

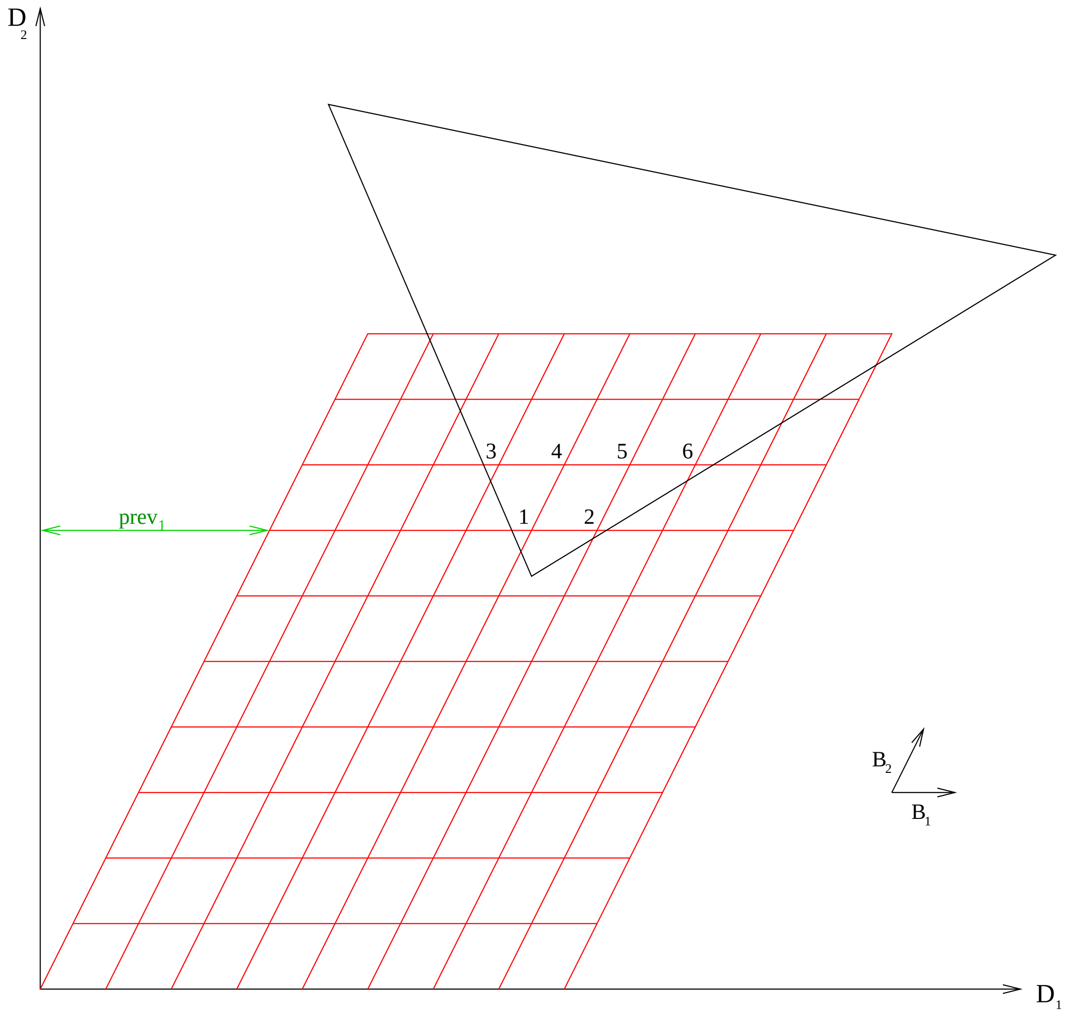
  All the above steps combined form init'

Loop Increment

We have to jump over the holes in this dimension

$$ID_k += c_k$$

Our method of scanning

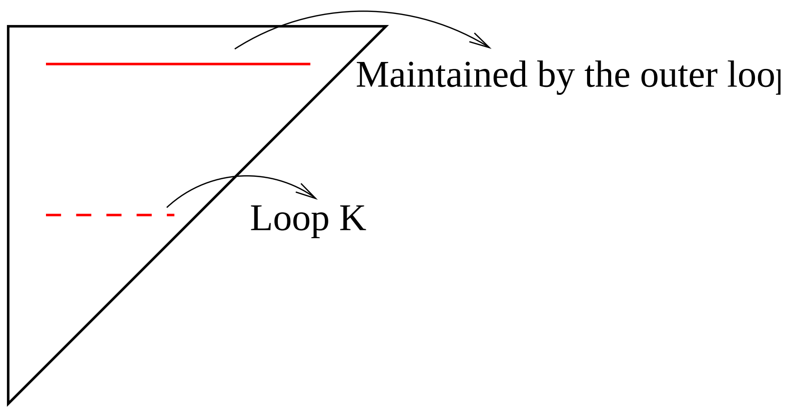


β_1=(2,0), β_2=(1,2)

What is left?

Calculation of $prev_k \rightarrow$ simply needs some book-keeping at each loop. We have to maintain a vector that gives the distance traversed along inner dimensions

(can be handled by a triangular 2-D array)



Scanning a Union of Z-Polyhedra

Simply needs some additional book-keeping!!

- All the previous steps are needed
- We need to maintain all the mentioned variables for each Z-Polyhedra in the union

Effects on the Loop nest

Loop initialization and increment remain the same for each set of $ID's$ (ID^i_1, ID^i_2, ID^i_3,...,ID^i_n)

where the superscript i identifies its Z-Polyhedra

A Union of Z-Polyhedra

However, initialization and increment will have to be carried out conditionally, for which we include another variable $flag$ that sets the condition for the above

Bound Testing

We have to see if every ID_k for the particular dimension (corresponding to all Z-Polyhedra) has reached its upper bound

$\rightarrow$ while (test$_k^1$ $\|$test$_k^2\|$ ... $\|$test$_k^p$)

where p identifies the Z-Polyhedra

Initialization, Increment and the $flag$ system

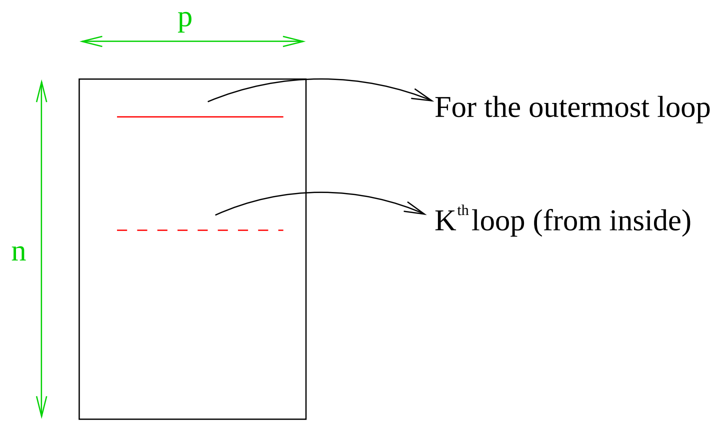
At loop k, we operate on a certain Z-Polyhedra if its flag $==$ true

else we set $ID_k^i =$ upper bound $+ 1$

Flag Management

- We find the minimum value of ID_k^p for all p
- The $flag$ for this/these Z-Polyhedra is marked true (may be more than 1)
- $flags$ for all other Z-Polyhedra are marked false
- Go to inner loops
- Upon return, only the true flagged ID$_k$'s are incremented

Flags can be implemented as a 2-D array ($n \times p$)



Where do we stand?

Perhaps all this has already been achieved (Paul Feutrier: *Scanning without Do loops*) by another but an equivalent method

However,

- We may have a performance benefit since the calculations at each loop level are for traversal along that dimension only

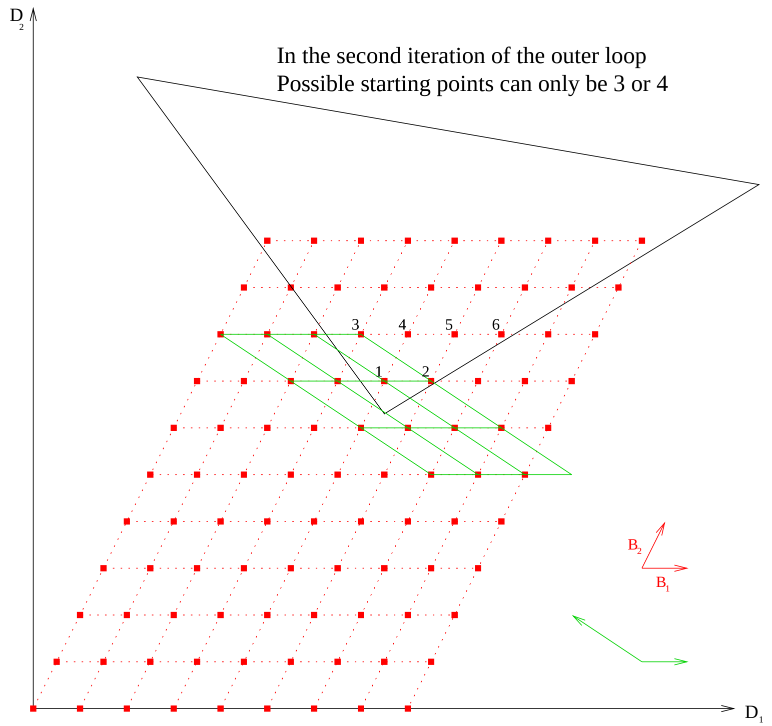
- We have an optimization over our above mentioned method that limits the initialization step to 'choosing between two values' only

Optimization (concept)

β_1= (2,0), β_2=(1,2)

From β_1, β_2 we obtain, γ_1, γ_2 such that

slope of γ_2 < slope of domain boundary $\leq$ slope of $(\gamma_1 + \gamma_2)$ {Slope is $\frac{\Delta D_1}{\Delta D_2}$}



Therefore,

γ_1= β_1= (2,0)

γ_2=β_2-2β_1= (-3,2)