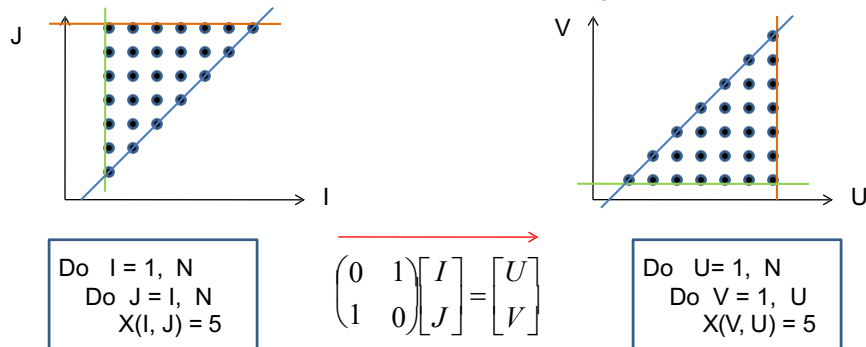


Loop permutation can be modeled as a linear transformation on iteration space:



Permutation of loops in n-loop nest:  $n \times n$  permutation matrix  $P$

$P \mathbf{I} = \mathbf{U}$ , where  $\mathbf{I}$  and  $\mathbf{U}$  are original and transformed iteration vectors, respectively

Questions:

- (1) How do we generate new loop bounds?
- (2) How do we modify the loop body?
- (3) How do we know when loop interchange is legal?

## Code Generation for Transformed Loop Nest

Two problems: (1) Loop bounds (2) Change of variables in body

### (1) New bounds:

Original bounds:  $A * I \leq b$  where  $A$  is in echelon form

Transformation:  $U = T * I$

Note: for loop permutation,  $T$  is a permutation matrix  $\Rightarrow$  inverse is integer matrix

So, bounds on  $U$  can be written as  $A * T^{-1}U \leq b$

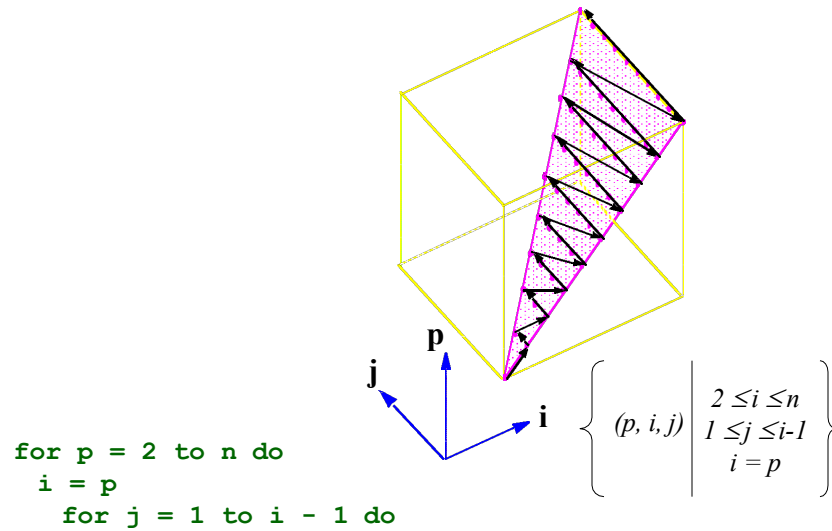
Perform Fourier-Motzkin elimination on this system of inequalities to obtain bounds on  $U$ .

### (2) Change of variables (fixing loop body):

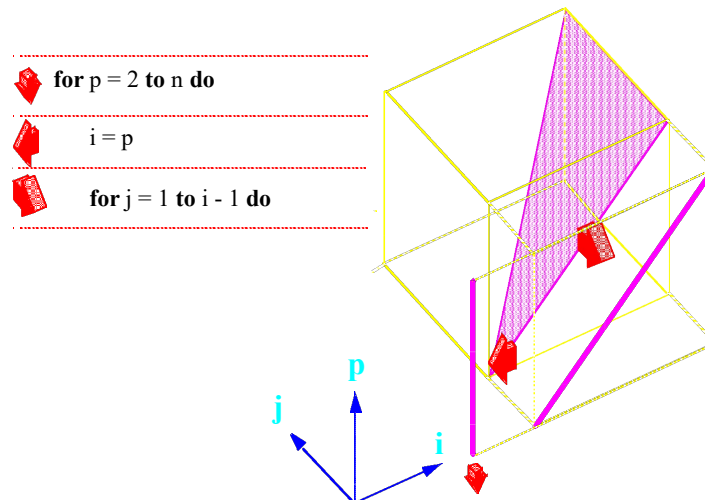
$$I = T^{-1}U$$

Replace old variables by new using this formula

## Space of Iterations



## Projections



## Fourier Motzkin Elimination

$2 \leq i \leq n$   
 $1 \leq j \leq i-1$   
 $i = p$

- Project  $i \rightarrow j \rightarrow p$
- Find the bounds of  $i$ 

$2 \leq i$   
 $j+1 \leq i$   
 $p \leq i$   
 $i \leq n$   
 $i \leq p$

$i: \max(2, j+1, p) \text{ to } \min(n, p)$   
 $i: p$

- Eliminate  $i$

$2 \leq n$   
 $j+1 \leq n$   
 $p \leq n$   


---

 $2 \leq p$   
 $j+1 \leq p$   
 $p \leq p$   


---

 $1 \leq j$

- Eliminate redundant

$p \leq n$   
 $2 \leq p$   
 $j+1 \leq p$   
 $1 \leq j$

- Continue onto finding bounds of  $j$

Further read: [https://en.wikipedia.org/wiki/Fourier-Motzkin\\_elimination](https://en.wikipedia.org/wiki/Fourier-Motzkin_elimination)

## Fourier Motzkin Elimination

$p \leq n$   
 $2 \leq p$   
 $j+1 \leq p$   
 $1 \leq j$

- Find the bounds of  $j$

$1 \leq j$   
 $j \leq p-1$

$j: 1 \text{ to } p-1$

- Eliminate  $j$

$1 \leq p-1$   


---

 $p \leq n$   
 $2 \leq p$

- Eliminate redundant

$2 \leq p$   
 $p \leq n$

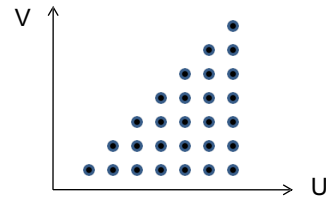
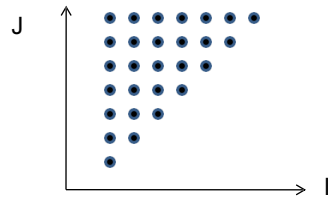
- Find the bounds of  $p$

$2 \leq p$   
 $p \leq n$

$p: 2 \text{ to } n$

$p = \text{my\_pid}()$   
 if  $p \geq 2$  and  $p \leq n$  then  
   for  $j = 1$  to  $p-1$  do  
      $i = p$

Example:



Do  $I = 1, N$   
Do  $J = 1, N$   
 $X(I, J) = 5$

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{bmatrix} I \\ J \end{bmatrix} = \begin{bmatrix} U \\ V \end{bmatrix}$$

Do  $U = 1, N$   
Do  $V = 1, \min(U, N)$   
 $X(V, U) = 5$

$$\begin{bmatrix} -1 & 0 \\ 1 & 0 \\ 1 & -1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} I \\ J \end{bmatrix} \leq \begin{bmatrix} -1 \\ N \\ 0 \\ N \end{bmatrix}$$

$$\begin{bmatrix} -1 & 0 \\ 1 & 0 \\ 1 & -1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} U \\ V \end{bmatrix} \leq \begin{bmatrix} -1 \\ N \\ 0 \\ N \end{bmatrix}$$

Fourier-  
Motzkin  
Elimination

$$\begin{bmatrix} -1 & 0 \\ 1 & 0 \\ 1 & -1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} U \\ V \end{bmatrix} \leq \begin{bmatrix} -1 \\ N \\ 0 \\ N \end{bmatrix}$$

$$\begin{bmatrix} 0 & -1 \\ 0 & 1 \\ -1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} U \\ V \end{bmatrix} \leq \begin{bmatrix} -1 \\ N \\ 0 \\ N \end{bmatrix}$$

Projecting out  $V$  from system gives

$$1 \leq U \leq N$$

Bounds for  $V$  are

$$1 \leq V \leq \min(U, N)$$

These are loop bounds given by FM elimination.

With a little extra work, we can simplify the upper bound of  $V$  to  $U$ .

## Key points:

- Loop bounds determination in transformed code is mechanical.
- Polyhedral algebra technology can handle very general bounds with max's in lower bounds and min's in upper bounds.

### When is permutation legal?

Position so far: if there is a dependence between iterations, then permutation is illegal.

Do  $I = 1, 100$

Do  $J = 1, 100$

$X(2I, J) = \dots\dots X(2I-1, J-1) \dots\dots$

Is there a flow dependence between different iterations?  $1 \leq I_w, I_r, J_w, J_r \leq 100$

$(I_w, J_w) \prec (I_r, J_r)$

$2I_w = 2I_r - 1$

$J_w = J_r - 1$

ILP decision problem: is there an integer in union of two convex polyhedra?

No  $\Rightarrow$  permutation is legal.

Permutation is legal only if dependence does not exist: too simplistic.

Example:

Do  $I = 1, 100$

Do  $J = 1, 100$

$X(I, J) = \dots\dots X(I-1, J-1) \dots\dots$

Only dependence is flow

dependence:

$$1 \leq I_w, I_r, J_w, J_r \leq 100$$

$$(I_w, J_w) \prec (I_r, J_r)$$

$$I_w = I_r - 1$$

$$J_w = J_r - 1$$

ILP problem has solution: for example,  $(I_w=1, J_w=1, I_r=2, J_r=2)$   
 Dependence exists but loop interchange is legal!

**Point:** Existence of dependence is a very “coarse” criterion to determine if interchange is legal.

Additional information about dependence may let us decide whether a given transformation is legal.

To get a handle on all this, let us first define “dependence” precisely.

Consider single loop case first:

Do  $I = 1, 100$

$X(2I + 1) = \dots\dots X(I)\dots\dots$

Flow dependences between iterations:

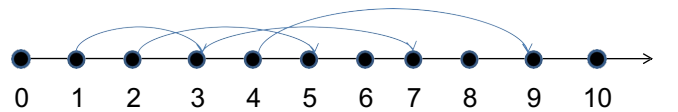
Iteration 1 writes to  $X(3)$  which is read by iteration 3.

Iteration 2 writes to  $X(5)$  which is read by iteration 5.

.....

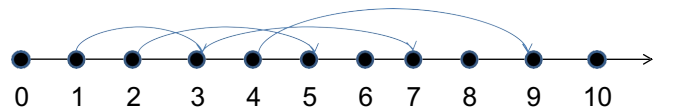
Iteration 49 writes to  $X(99)$  which is read by iteration 99.

If we ignore the array locations and just think about dependence between iterations, we can draw this geometrically as follows:



Dependence arrows always go forward in iteration space. (e.g., there cannot be a dependence from iteration 5 to iteration 2)

Intuitively, dependence arrows tell us constraints on transformations.



Suppose a transformed program does iteration 2 before iteration 1. **OK!**

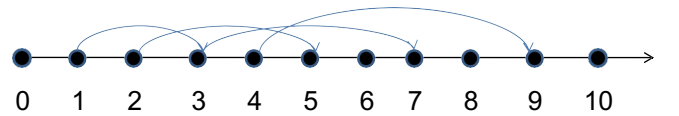
Transformed program does iteration 3 before iteration 1. **Illegal!**

Formal view of a dependence: relation between points in the iteration space.

Do  $I = 1, 100$

$$X(2I + 1) = \dots\dots X(I) \dots\dots$$

$$\text{Flow dependence} = \{(I_w, 2I_w + 1) \mid 1 \leq I_w \leq 49\}$$



In the spirit of dependence, we will often write this as follows:

$$\text{Flow dependence} = \{(I_w \rightarrow 2I_w + 1) \mid 1 \leq I_w \leq 49\}$$

## 2D loop nest

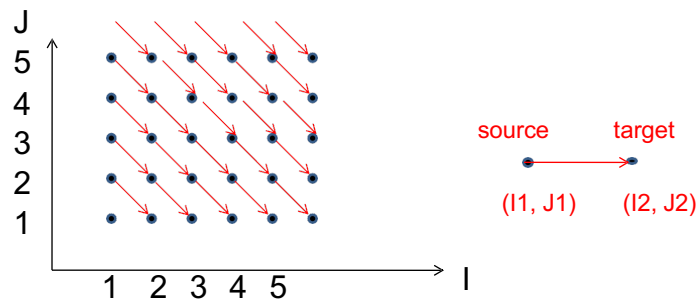
Do 10  $I = 1, 100$

Do 10  $J = 1, 100$

$$10 \quad X(I, J) = X(I-1, J+1) + 1$$

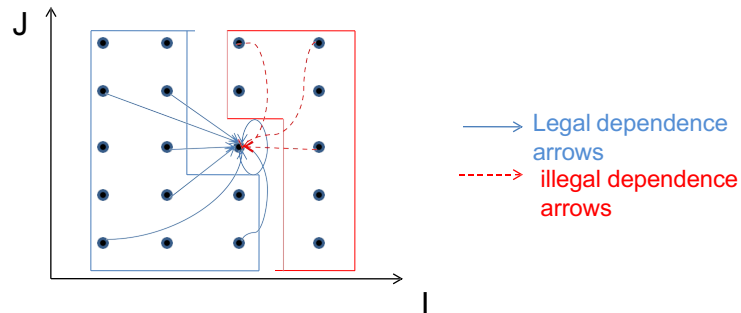
Dependence: relation of the form  $(I_1, J_1) \rightarrow (I_2, J_2)$ .

Picture in iteration space:





Legal and illegal dependence arrows:



If  $(A \rightarrow B)$  is a dependence arrow, then  $A$  must be lexicographically less than or equal to  $B$ .

Dependence relation can be computed using ILP calculator

Do 10  $I = 1, 100$

Do 10  $J = 1, 100$

10  $X(I, J) = X(I-1, J+1) + 1$

$(I_w, J_w) \rightarrow (I_r, J_r)$

Flow dependence constraints:  $1 \leq I_w, I_r, J_w, J_r \leq 100$

$(I_w, J_w) \prec (I_r, J_r)$

$I_w = I_r - 1$

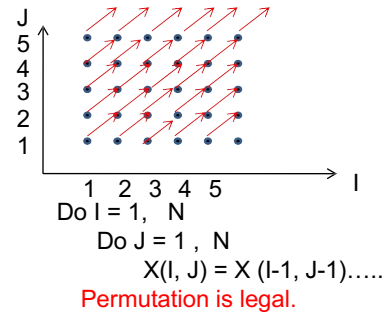
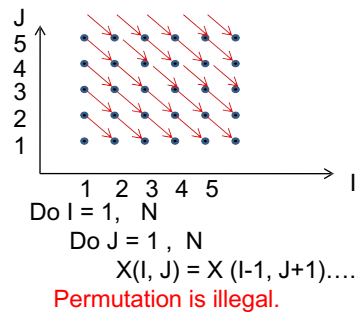
$J_w = J_r + 1$

Use ILP calculator to determine the following relation:

$$D = \{(I_w, J_w) \rightarrow (I_w + 1, J_w - 1) \mid (1 \leq I_w \leq 99) \wedge (2 \leq J_w \leq 100)\}$$

If we have the full dependence relation, can we determine when permutation is legal?

Let us look at geometric picture to understand when permutation is legal.



Intuitively, if an iteration is dependent on an iteration in its “upper left hand corner”, permutation is illegal. How do we express this formally?

Legality of permutation can be framed as an ILP problem too.

Do 10  $I = 1, 100$

Do 10  $J = 1, 100$

10  $X(I, J) = X(I-1, J+1) + 1$

Permutation is illegal if there exist iterations in source program such that  $(I_1, J_1), (I_2, J_2)$

$((I_1, J_1) \rightarrow (I_2, J_2)) \in D$  (dependent iterations)

$(J_2, I_2) \prec (J_1, I_1)$  (iterations done in wrong order in transformed program)

This can obviously be phrased as an ILP problem and solved.

One solution:  $(I_1, J_1) = (1, 2), (I_2, J_2) = (2, 1)$

Thus, interchange is illegal.

### General picture:

Permutation is co-ordinate transformation:  $U = P * I$   
 where P is a permutation matrix.

Conditions for legality of transformation:

For each dependence D in loop nest, check that there do not exist iterations  $I_1$  and  $I_2$  such that

$$(I_1 \rightarrow I_2) \in D$$

$$P(I_2) \prec P(I_1)$$

First condition: dependent iterations

Second condition: iterations are done in wrong order in transformed program.

Legality of permutation can be determined by solving a bunch of ILP problems.

### Problems with using full dependence sets:

- Expensive (time/space) to compute full relations
- Need to solve ILP problems again to determine legality of permutation
- Symbolic loop bounds ('N') require parameterized sets ('N' is unbound variable in definition of dependence set)

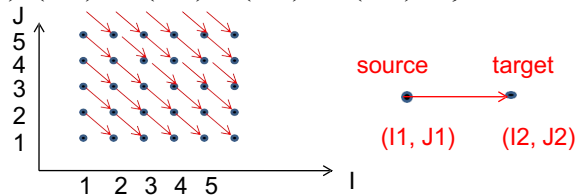
Dependence abstractions: summary of dependence set D

- Less information than full set of tuples in D
- More information than non-emptiness of D
- Intuitively, “as much as is needed for transformations of interest”

### Distance/direction: Summarize dependence relation

Look at dependence relation from earlier slides:

$$\{(1,2) \rightarrow (2,1), (1,3) \rightarrow (2,2), \dots, (2,2) \rightarrow (3,1), \dots\}$$



Difference between dependent iterations =  $(1, -1)$ . That is,  $(I_w, J_w) \rightarrow (I_r, J_r) \in \text{dependence relation}$ , implies :

$$I_r - I_w = 1$$

$$J_r - J_w = -1$$

We will say that the **distance vector** is  $(1, -1)$ .

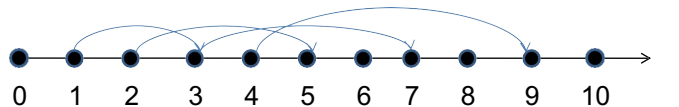
**Note:** From distance vector, we can easily recover the full relation. In this case, distance vector is an **exact** summary of relation.

Set of dependent iterations usually is represented by many distance vectors.

Do  $I = 1, 100$

$$X(2I + 1) = \dots X(I) \dots$$

$$\text{Flow dependence} = \{(I_w \rightarrow 2I_w + 1) \mid 1 \leq I_w \leq 49\}$$



Distance vectors:  $\{(2), (3), (4), \dots, (50)\}$

**Distance vectors can obviously never be negative** (if  $(-1)$  was a distance vector for some dependence, there is an iteration  $I_1$  that depends on iteration  $I_1+1$  which is impossible).

Distance vectors are an approximation of a dependence:

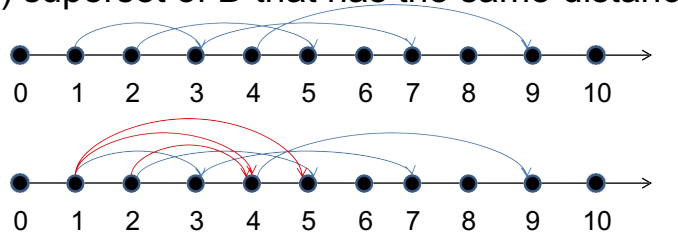
(intuitively, we know the arrows but we do not know their sources.)  $D = \{(I_w, 2I_w + 1) | 1 \leq I_w \leq 49\}$

Example:

Distance vectors:  $\{(2), (3), (4), \dots, (50)\}$

$$D_1 = \{(I_1, I_2) | (1 \leq I_1 \leq 49) \wedge (50 + I_1) \geq I_2 \geq (2I_1 + 1)\}$$

(convex) superset of D that has the same distance vectors.



Both dependences have same set of distance vectors

### Computing distance vectors for a dependence

Do  $I = 1, 100$

$$X(2I + 1) = \dots X(I) \dots$$

Flow dependence:  $1 \leq I_w \leq I_r \leq 100$

$$2I_w + 1 = I_r$$

Flow dependence =  $\{(I_w, 2I_w + 1) | 1 \leq I_w \leq 49\}$

Computing distance vectors without computing dependence set:

Introduce a new variable  $\Delta = I_r - I_w$  and project onto  $\Delta$

$$1 \leq I_w < I_r \leq 100$$

$$2I_w + 1 = I_r$$

$$\Delta = I_r - I_w$$

Solution:  $\Delta = \{d | 2 \leq d \leq 50\}$

### Example: 2D loop nest

```

Do 10 I = 1, 100
  Do 10 J = 1, 100
    10 X(I, J) = X(I-1, J+1) + 1
  
```

Flow dependence constraints:  $(I_w, J_w) \rightarrow (I_r, J_r)$

Distance vector:  $(\Delta_1, \Delta_2) = (I_r - I_w, J_r - J_w)$

$$1 \leq I_w, I_r, J_w, J_r \leq 100$$

$$(I_w, J_w) \prec (I_r, J_r)$$

$$I_w = I_r - 1$$

$$J_w = J_r + 1$$

$$(\Delta_1, \Delta_2) = (I_r - I_w, J_r - J_w)$$

Solution:  $(\Delta_1, \Delta_2) = (1, -1)$

### General approach to computing distance vectors:

Set of distance vectors generated from a dependence is itself a polyhedral set.

Computing distance vectors without computing dependence set:

To the linear system representing the existence of the dependence, add new variables corresponding to the entries in the distance vector and project onto these variables.