Transformations and Dependences

Recall:

- Polyhedral algebra tools for
  - determining emptiness of convex polyhedra
  - enumerating integers in such a polyhedron.
- Central ideas:
  - reduction of matrices to echelon form by unimodular column operations,
  - Fourier-Motzkin elimination

Let us use these tools to determine (i) legality of permutation and (ii) generation of transformed code.

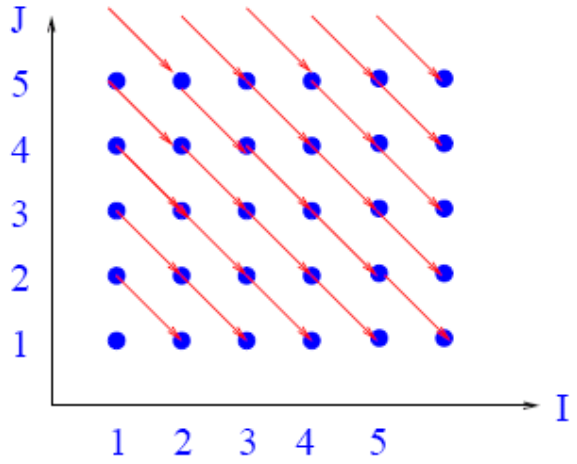
Dependence relation

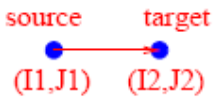
2D loop nest

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

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

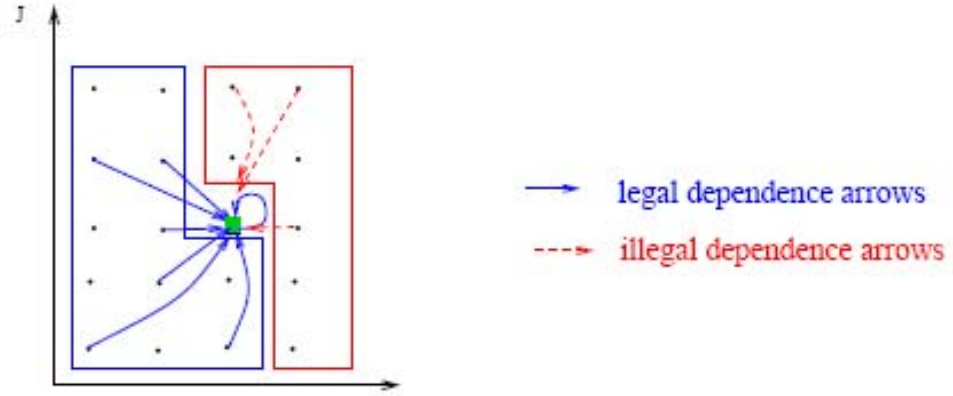
Picture in iteration space:





Legal and impossible dependences

Legal and illegal dependence arrows:



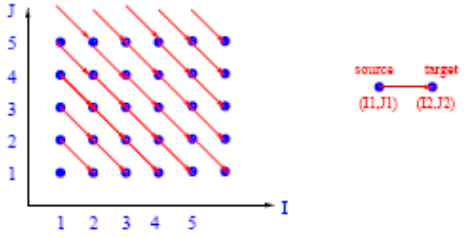
If $(A \to B)$ is a dependence arrow, then A must be lexicographically less than or equal to B.

Distance vectors

Distance/direction: Summarize dependence relation

Look at dependence relation from earlier slides:

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



Difference between dependent iterations $= (1, -1)$. That is,

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

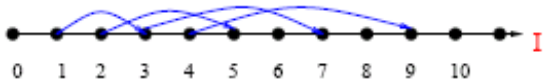
In this case, distance vector is an *exact* summary of relation.

Distance vectors

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

```
DO I = 1, 100
  X(2I+1) = ....X(I)...
```

Flow dependence $= \{(Iw \rightarrow 2Iw + 1) | 1 \leq Iw \leq 49\}$



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

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.)

Computing distance vectors

Computing distance vectors for a dependence

```
DO I = 1, 100
  X(2I+1) = ....X(I)...
```

Flow dependence:

$$\begin{aligned} 1 &\leq Iw < Ir \leq 100 \\ 2Iw+1 &= Ir \end{aligned}$$

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

Computing distance vectors without computing dependence set:
Introduce a new variable $\Delta = Ir - Iw$ and project onto Δ

$$\begin{aligned} 1 &\leq Iw < Ir \leq 100 \\ 2Iw+1 &= Ir \\ \Delta &= Ir - Iw \end{aligned}$$

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

Distance vectors: 2D loop nest

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

- $1 \leq Iw, Ir, Jw, Jr \leq 100$
- $(I_w, J_w) \prec (I_r, J_r)$
- $I_w = I_r - 1$
- $J_w = J_r + 1$
- $(\Delta_1, \Delta_2) = (I_r - I_w, J_r - J_w)$

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

Direction vectors

Direction vectors **Example:**

```
DO 10 I = 1,100
 10 X(2I+1) = X(I) + 1
```

Flow dependence equation: $2I_w + 1 = I_r$.
Dependence relation: $\{(1 \rightarrow 3), (2 \rightarrow 5), (3 \rightarrow 7), ...\}$ (1).

No fixed distance between dependent iterations!
But all distances are +ve, so use *direction vector* instead.
Here, direction = (+).
Intuition: (+) direction = some distances in range $[1, \infty)$

In general, direction = (+) or (0) or (-).
Also written by some authors as (<), (=), or (>).

Direction vectors are not exact.
(eg):if we try to recover dependence relation from direction (+), we get bigger relation than (1):
$\{(1 \rightarrow 2), (1 \rightarrow 3), ..., (1 \rightarrow 100), (2 \rightarrow 3), (2 \rightarrow 4), ...\}$

Direction vectors for nested loops

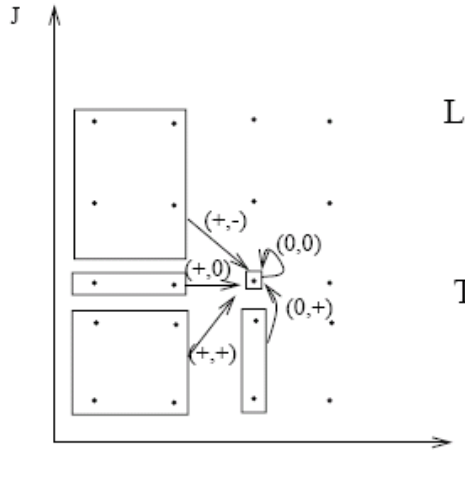
Directions for Nested Loops

Assume loop nest is (I,J).

If $(I_1, J_1) \rightarrow (I_2, J_2) \in$ dependence relation, then

Distance $= (I_2 - I_1, J_2 - J_1)$

Direction $= (sign(I_2 - I_1), sign(J_2 - J_1))$



Legal direction vectors:

(+,+) (0,+)
(+,-) (0,0)
(+,0)

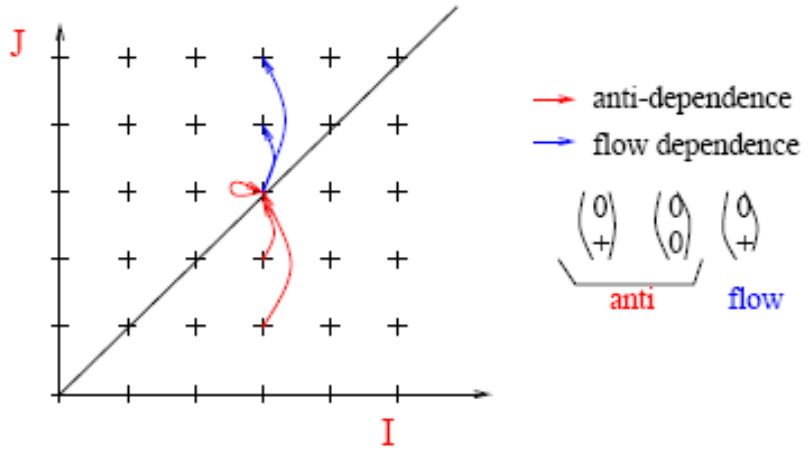
The following direction vectors cannot exist:

(0,-) (-,+)
(-,0)
(-,-)

Valid dependence vectors are lexicographically positive.

Example

```
DO I = 1,N
  DO J = 1,N
    X(I,J) = ...X(I,I)...
```



Example (contd.)

Linear system for anti-dependence:

$$I_w = I_r$$
$$J_w = I_r$$
$$1 \leq I_w, I_r, J_w, J_r \leq N$$
$$(I_r, J_r) \preceq (I_w, J_w)$$
$$\Delta 1 = (I_w - I_r)$$
$$\Delta 2 = (J_w - J_r)$$

Projecting onto $\Delta 1$ and $\Delta 2$, we get

$$\Delta 1 = 0$$
$$0 \leq \Delta 2 \leq (N - 1)$$

So directions for anti-dependence are

```
0    and   0
0          +
```

Example (contd.)

Similarly, you can compute direction for flow dependence

0

+

and also show that no output dependence exists.

Dependence matrix

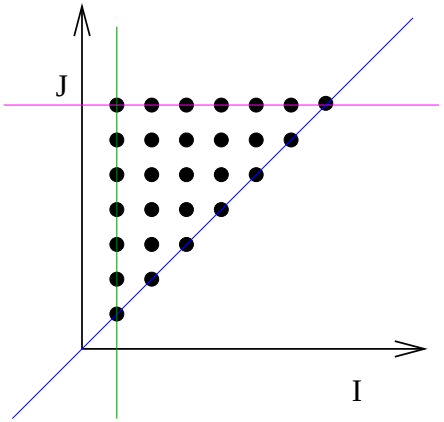
Dependence matrix for a loop nest

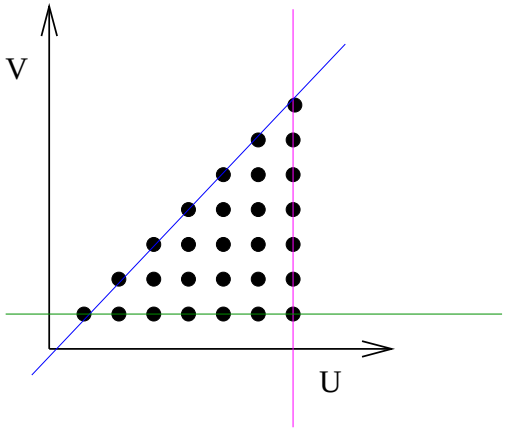
Matrix containing all dependence distance/direction vectors for all dependences of loop nest.

In our example, the dependence matrix is

$$\begin{pmatrix} 0 & 0 \\ 0 & + \end{pmatrix}$$

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





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

```
DO  I =  1, N
  DO J = I,N
    X(I,J) = 5
```

```
DO U = 1, N
  DO V = 1,U
    X(V,U) = 5
```

Permutation of loops in n-loop nest: nxn permutation matrix P

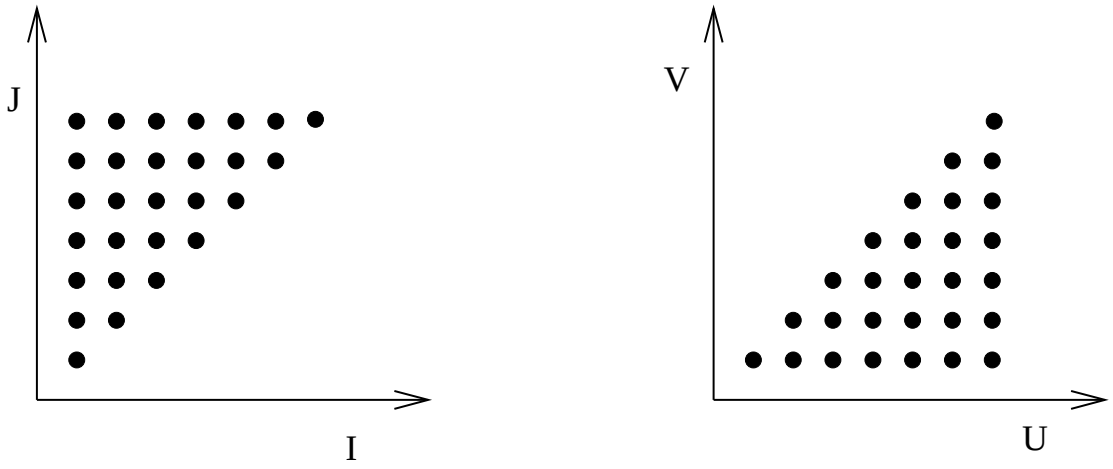
$$P\,I = U$$

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?




```
DO  I =  1, N
  DO J = I,N
  .........
```

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

```
DO U = 1, N
  DO V = 1,U
  .........
```

$$\begin{bmatrix} I1 \\ J1 \end{bmatrix} \rightarrow \begin{bmatrix} I2 \\ J2 \end{bmatrix}$$

$$T \begin{bmatrix} I1 \\ J1 \end{bmatrix} \qquad T \begin{bmatrix} I2 \\ J2 \end{bmatrix}$$

Dependence distance $= \begin{bmatrix} I2 - I1 \\ J2 - J1 \end{bmatrix}$

Distance between iterations =

$$T \begin{bmatrix} I2 \\ J2 \end{bmatrix} - T \begin{bmatrix} I1 \\ J1 \end{bmatrix} = T \begin{bmatrix} I2 - I1 \\ J2 - J1 \end{bmatrix} = \begin{bmatrix} J2 - J1 \\ I2 - I1 \end{bmatrix}$$

Check for legality: interchange positions in distance/direction vector & check for lex +ve

If transformation P is legal and original dependence matrix is D, new dependence matrix is T*D.

Correctness of general permutation

Transformation matrix: T

Dependence matrix: D

Matrix in which each column is a distance/direction vector

Legality: $T.D \succ 0$

Dependence matrix of transformed program: $T.D$

Examples:

```
DO I = 1,N
  DO J = 1,N
   X(I,J) = X(I-1,J-1)....
```

Distance vector = (1,1) => permutation is legal
Dependence vector of transformed program = (1,1)

```
DO I = 1,N
  DO J = 1,N
   X(I,J) = X(I-1,J+1)....
```

Distance vector = (1,-1) => permutation is not legal

Code Generation for Transformed Loop Nest

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

(1) New bounds:

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

Transformation: $\underline{U} = T * \underline{I}$

Note: for loop permutation, T is a permutation matrix
=> inverse is integer matrix

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

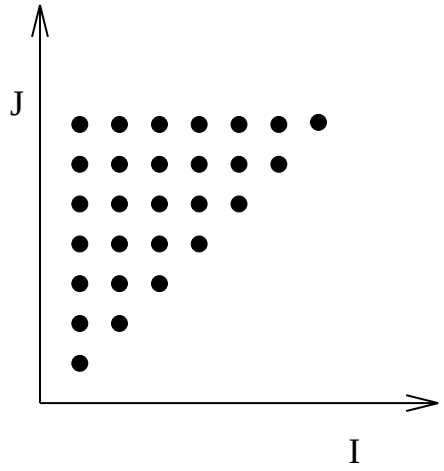
Perform Fourier-Motzkin elimination on this system of inequalities to obtain bounds on $\underline{U}$.

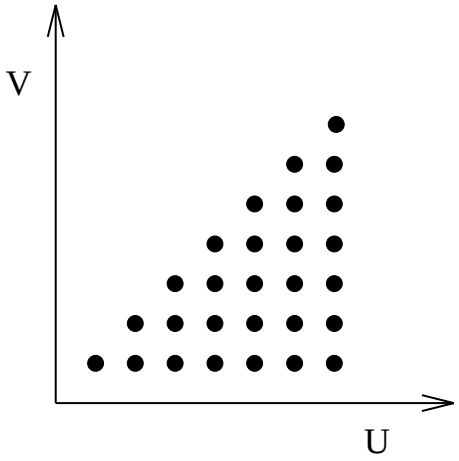
(2) Change of variables:

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

Replace old variables by new using this formula

Example:





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

```
DO  I =  1, N
  DO J = I,N
    X(I,J) = 5
```

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

```
DO U = 1, N
  DO V = 1,U
    X(V,U) = 5
```

Fourier-Motzkin elimination

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

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

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

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.
- No need for pattern matching etc for triangular bounds and the like.

Theory for permutations applies to other loop transformations that can be modeled as linear transformations: skewing, reversal,scaling.

Transformation matrix: T (a non-singular matrix)

Dependence matrix: D

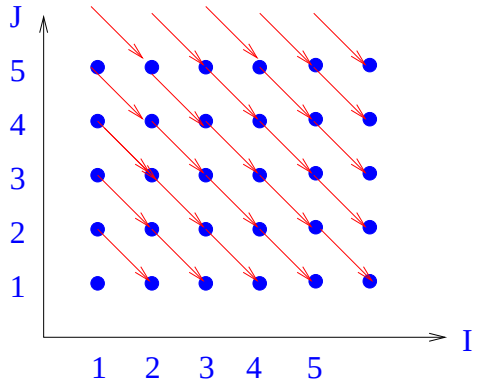
Matrix in which each column is a distance/direction vector

Legality: $T.D \succ 0$

Dependence matrix of transformed program: $T.D$

Small complication with code generation if scaling is included.

Motivating example: Wavefront code




```
DO I = 1,N
  DO J = 1,N
    X(I,J) = X(I-1,J+1).....
```

Dependence matrix = $\begin{bmatrix} 1 \\ -1 \end{bmatrix}$

Dependence between two iterations

=> iterations touch the same location

=> potential for exploiting data reuse!

`N` iteration points between executions of dependent iterations.

Can we schedule dependent iterations closer together?

For now, focus only on reuse opportunities between dependent iterations.

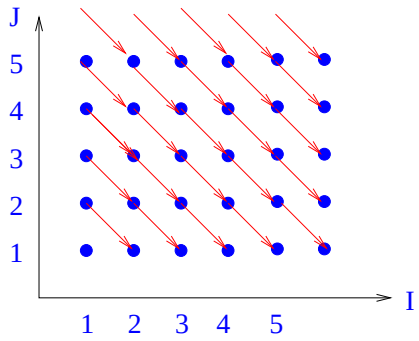
This exploits only one kind of temporal locality.

There are other kinds of locality that are important:

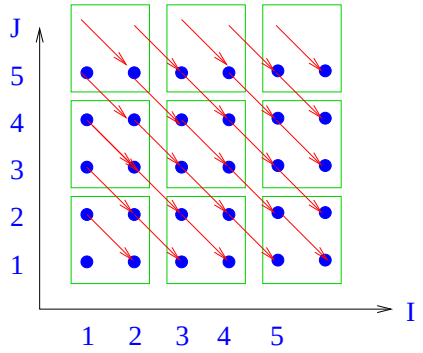
- iterations that *read* from the same location: input dependences
- spatial locality

Both are easy to add to basic model as we will see later.

Exploiting temporal locality in wavefront



Permutation is illegal!



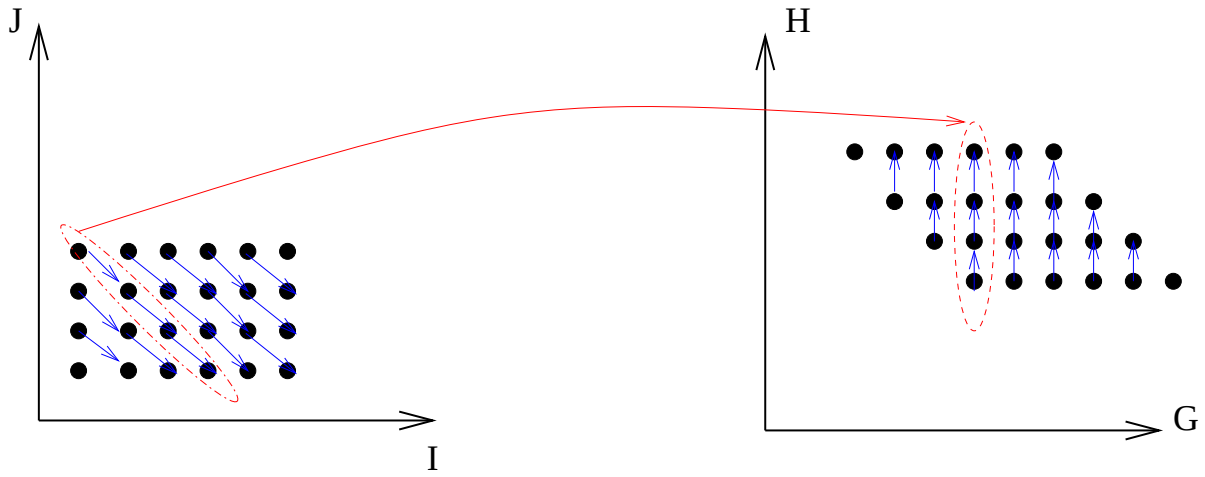
Tiling is illegal!

We have studied two transformations: permutation and tiling.

Permutation and tiling are both illegal.

Height Reduction

One solution: schedule iterations along 45 degree planes !



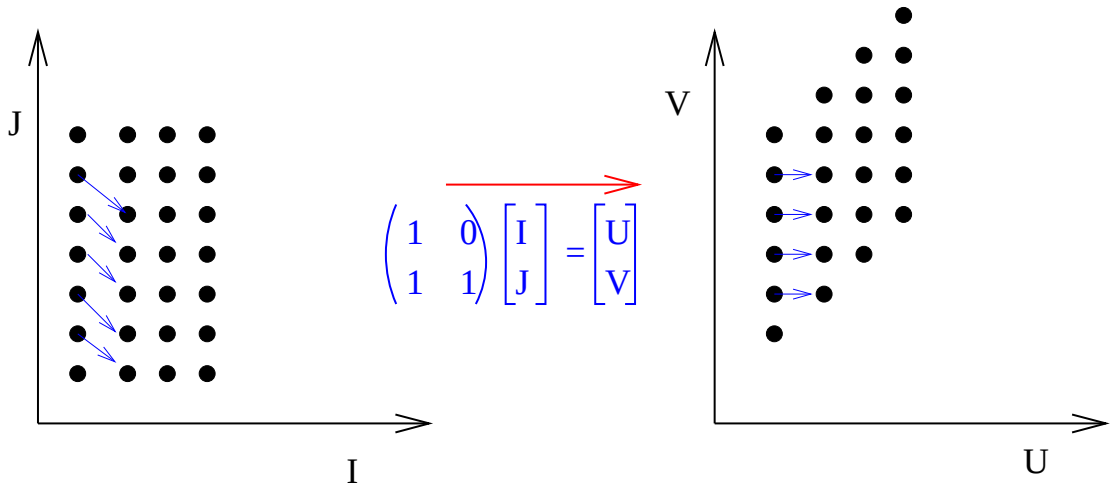
Note:

- Transformation is legal.
- Dependent iterations are scheduled close together, so good for locality.

Can we view this in terms of loop transformation?

Loop skewing followed by loop reversal.

Loop Skewing: a linear loop transformation



Skewing of inner loop by outer loop: $\begin{pmatrix} 1 & 0 \\ k & 1 \end{pmatrix}$ (k is some fixed integer)

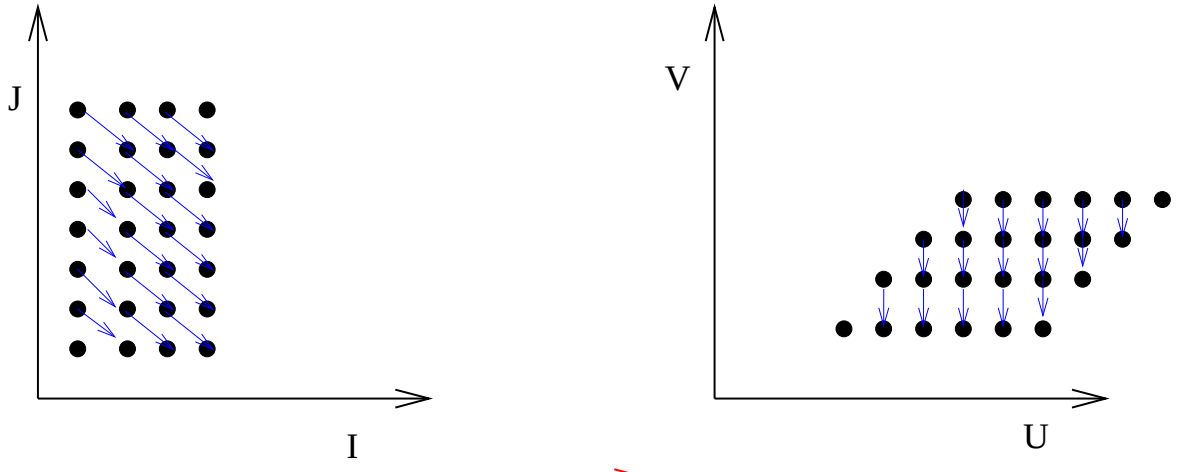
Skewing of inner loop by an outer loop: always legal

New dependence vectors: compute T*D

In this example, D = $\begin{bmatrix} 1 \\ -1 \end{bmatrix}$ T*D = $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$

This skewing has changed dependence vector but it has not brought dependent iterations closer together....

Skewing outer loop by inner loop



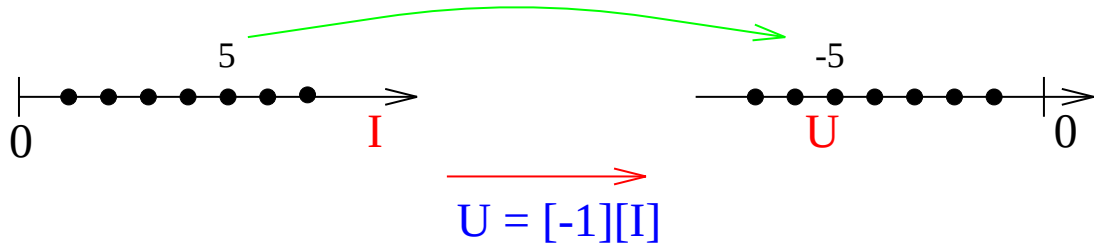
Outer loop skewing: $\begin{pmatrix} 1 & k \\ 0 & 1 \end{pmatrix}$

Skewing of outer loop by inner loop: not necessarily legal

In this example, D = $\begin{bmatrix} 1 \\ -1 \end{bmatrix}$ T*D = $\begin{bmatrix} 0 \\ -1 \end{bmatrix}$ incorrect

Dependent iterations are closer together (good) but program is illegal (bad).
How do we fix this??

Loop Reversal:a linear loop transformation




```
DO  I =  1, N
 X(I) = I+2
```

```
DO U =  -N,-1
 X(-U) = -U +2
```

Transformation matrix = [-1]

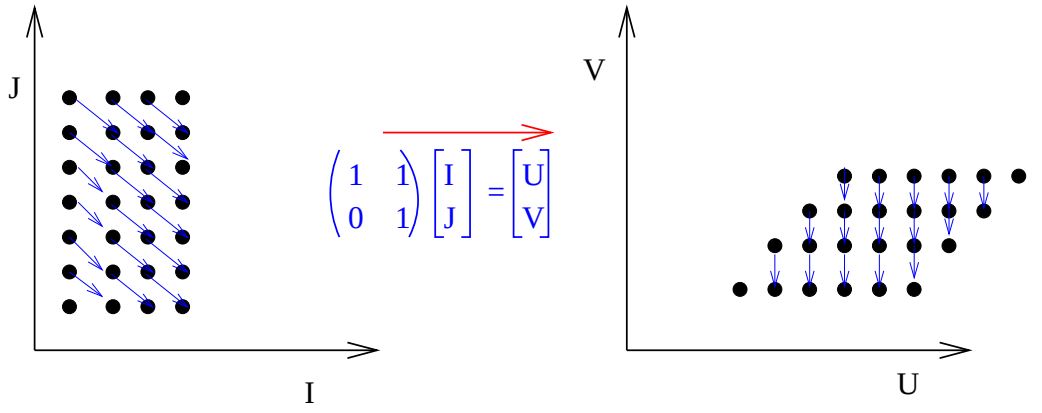
Another example: 2-D loop, reverse inner loop

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

Legality of loop reversal: Apply transformation matrix to all dependences & verify lex +ve

Code generation: easy

Need for composite transformations



$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{bmatrix} U \\ V \end{bmatrix} = \begin{bmatrix} G \\ H \end{bmatrix}$

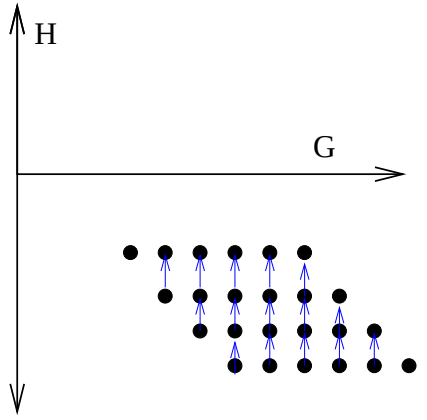
Transformation: skewing followed by reversal

In final program, dependent iterations are close together!

Composition of linear transformations = another linear transformation!

Composite transformation matrix is

$$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} * \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 0 & -1 \end{pmatrix}$$



How do we synthesize this composite transformation??

Some facts about permutation/reversal/skewing

- Transformation matrices for permutation/reversal/skewing are unimodular.
- Any composition of these transformations can be represented by a unimodular matrix.
- Any unimodular matrix can be decomposed into product of permutation/reversal/skewing matrices.
- Legality of composite transformation T: check that $T.D \succ 0$. (Proof: $T_3 * (T_2 * (T_1 * D)) = (T_3 * T_2 * T_1) * D$.)
- Code generation algorithm:
  - Original bounds: $A * \underline{I} \leq b$
  - Transformation: $\underline{U} = T * \underline{I}$
  - New bounds: compute from $A * T^{-1}\underline{U} \leq b$

Synthesizing composite transformations using matrix-based approaches

- Rather than reason about sequences of transformations, we can reason about the single matrix that represents the composite transformation.
- Enabling abstraction: dependence matrix

Height reduction: move reuse into inner loops

Dependence vector is $\begin{pmatrix} 1 \\ -1 \end{pmatrix}$.

Prefer not to have dependent iterations in different outer loop iterations.

So dependence vector in transformed program should look like $\begin{pmatrix} 0 \\ ?? \end{pmatrix}$.

So $T * \begin{pmatrix} 1 \\ -1 \end{pmatrix} = \begin{pmatrix} 0 \\ ?? \end{pmatrix}$

This says first row of T is orthogonal to $\begin{pmatrix} 1 \\ -1 \end{pmatrix}$.

So first row of T can be chosen to be (1 1).

What about second row?

Second row of T (call it s) should satisfy the following properties:

- s should be linearly independent of first row of T (non-singular matrix).
- $T * D$ should be a lexicographically positive vector.
- Determinant of matrix should be +1 or -1 (unimodularity).

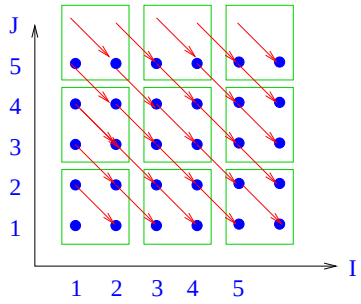
One choice: $(0 \quad -1)$, giving $T = \begin{pmatrix} 1 & 1 \\ 0 & -1 \end{pmatrix}$.

General questions

1. How do we choose the first few rows of the transformation matrix to push reuses down?
2. How do we fill in the rest of the matrix to get unimodular matrix?

Linear loop transformations to enable tiling

In general, tiling is not legal.



D = $\begin{pmatrix} 1 \\ -1 \end{pmatrix}$

Tiling is illegal!

Tiling is legal if loops are fully permutable (all permutations of loops are legal).

Tiling is legal if all entries in dependence matrix are non-negative.

- Can we always convert a perfectly nested loop into a fully permutable loop nest?
- When we can, how do we do it?

Theorem: If all dependence vectors are distance vectors, we can convert entire loop nest into a fully permutable loop nest.

Example: wavefront

Dependence matrix is $\begin{pmatrix} 1 \\ -1 \end{pmatrix}$.

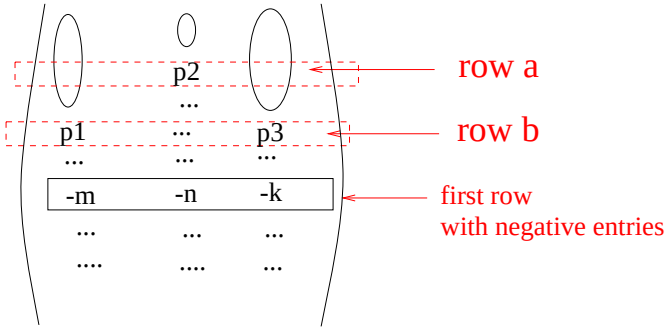
Dependence matrix of transformed program must have all positive entries.

So first row of transformation can be (1 0).

Second row of transformation (m 1) (for any m $>$ 0).

General idea: skew inner loops by outer loops sufficiently to make all negative entries non-negative.

Transformation to make first row with negative entries into row with non-negative entries



(a) for each negative entry in the first row with negative entries,
find the first positive number in the corresponding column
assume the rows for these positive entries are a,b etc as shown above

(b) skew the row with negative entries by appropriate multiples of
rows a,b....
For our example, multiple of row a = ceiling(n/p2)
multiple of row b = ceiling(max(m/p1,k/p3))

Transformation:

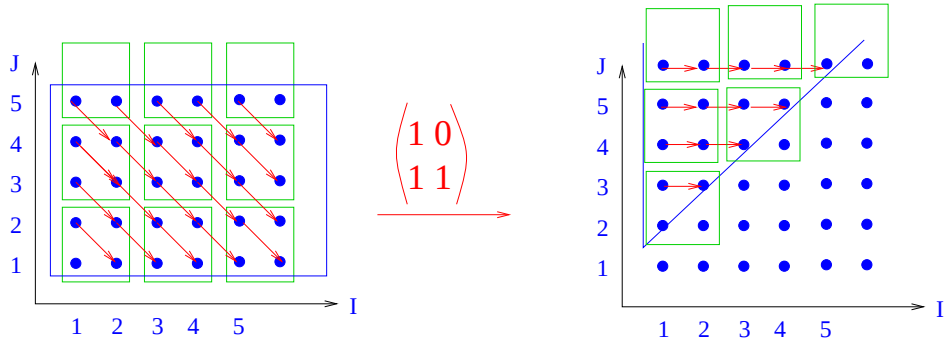
$$\begin{pmatrix} I & & \\ 0\ 0\ ..0\ \text{ceiling}(n/p2)\ 0\ 0\ \text{ceiling}(\max(m/p1,k/p3))0...0 \\ & I & \end{pmatrix}$$

General algorithm for making loop nest fully permutable:

If all entries in dependence matrix are non-negative, done.
Otherwise,

1. Apply algorithm on previous slide to first row with non-negative entries.
2. Generate new dependence matrix.
3. If no negative entries, done.
4. Otherwise, go step (1).

Result of tiling transformed wavefront



Original loop

Tiled fully permutable loop

Tiling generates a 4-deep loop nest.

Not as nice as height reduction solution, but it will work fine for locality enhancement except at tile boundaries (but boundary points small compared to number of interior points).