Linear Loop Transformations for Locality Enhancement

Story so far

- Cache performance can be improved by tiling and permutation
- Permutation of perfectly nested loop can be modeled as a *linear transformation* on the iteration space of the loop nest.
- Legality of permutation can be determined from the dependence matrix of the loop nest.
- Transformed code can be generated using ILP calculator.

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

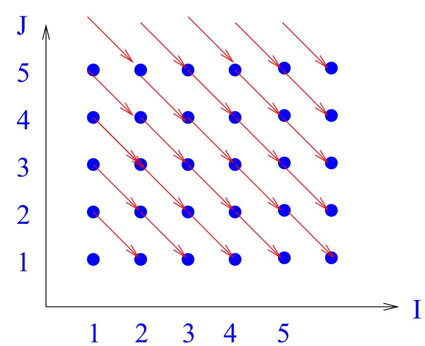
Matrix in which each column is a distance/direction vector

Legality: $T.D \succ 0$

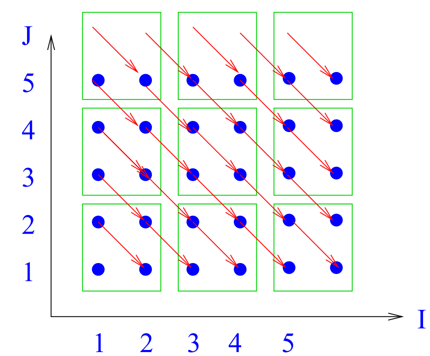
Dependence matrix of transformed program: $T.D$

Small complication with code generation if scaling is included.

Exploiting temporal locality in wavefront



Permutation is illegal!

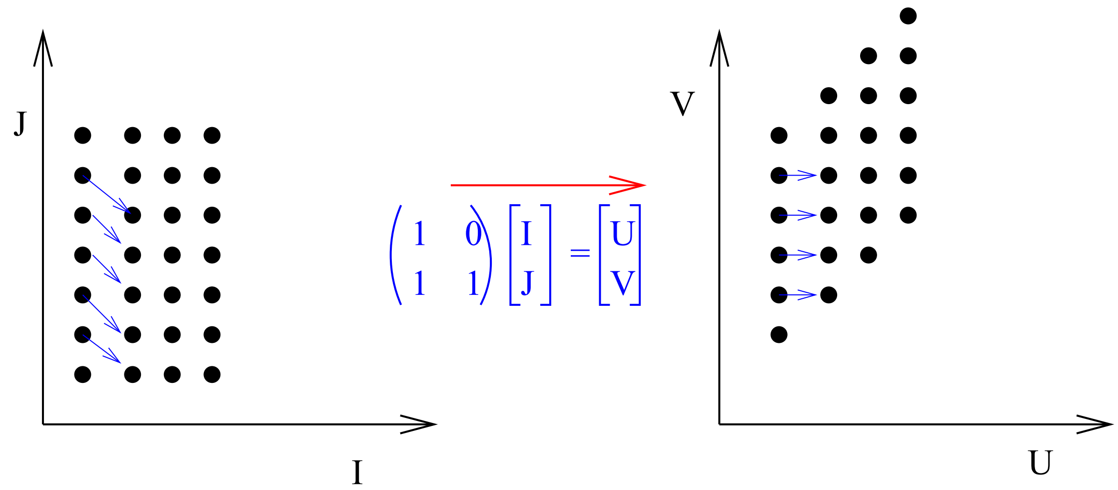


Tiling is illegal!

We have studied two transformations: permutation and tiling.

Permutation and tiling are both illegal.

Loop Skewing: a linear loop transformation



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

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

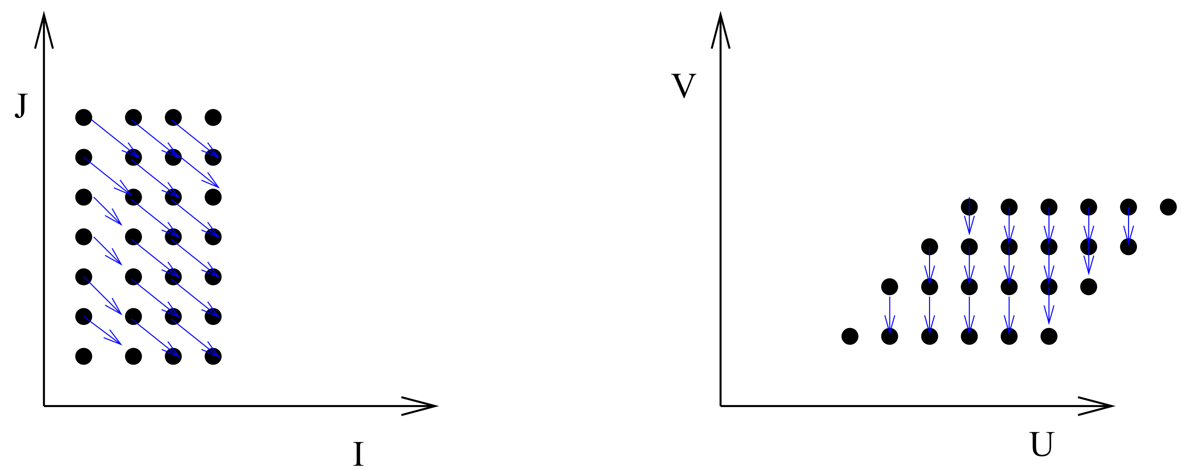
Skewing of inner loop by an outer loop: always legal

New dependence vectors: compute T*D

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

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

Skewing outer loop by inner loop



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

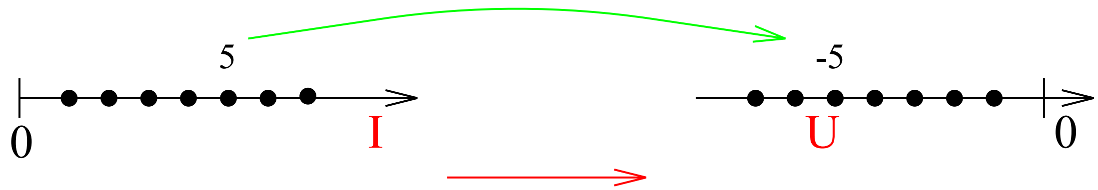
Outer loop skewing: $\begin{pmatrix} 1 & k \\ 0 & 1 \end{pmatrix}$

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



U = [-1][I]

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

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

Transformation matrix = [-1]

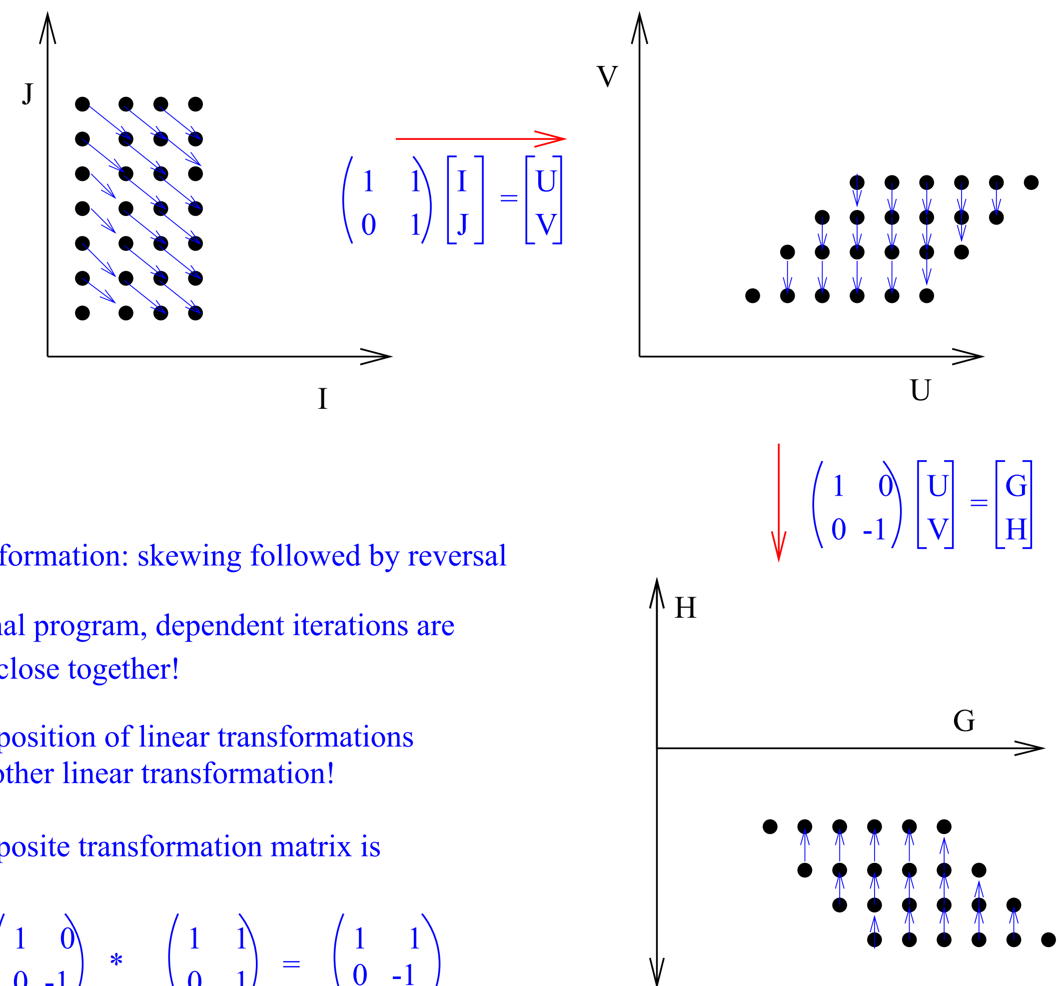
Another example: 2-D loop, reverse inner loop

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

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

Code generation: easy

Need for composite transformations



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

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

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??

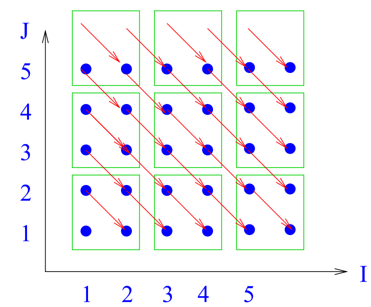
Some facts about permutation/reversal/skewing

- 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

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?

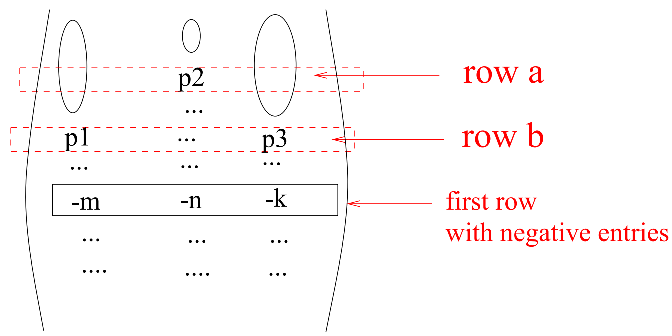
Theorem: If all dependence vectors are distance vectors, we can convert entire loop nest into a fully permutable loop nest.

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:

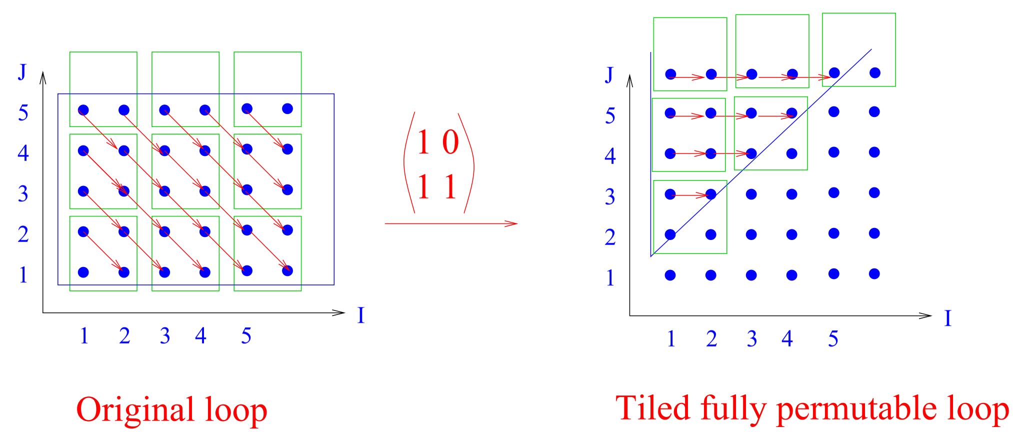
$$\begin{pmatrix} I \\ 0\ 0\ ..0\ \text{ceiling}(n/p2)\ 0\ 0\ \text{ceiling}(\max(m/p1,k/p3))0...0 \\ I \end{pmatrix}$$

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



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).

What happens with direction vectors?

In general, we cannot make loop nest fully permutable.

Example: $D = \begin{pmatrix} + \\ - \\ + \end{pmatrix}$

Best we can do is to make some of the loops fully permutable.

We try to make outermost loops fully permutable, so we would interchange the second and third loops, and then tile the first two loops only.

Idea: algorithm will find *bands* of fully permutable loop nests.