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.

Linear transformation matrix: $T$

Dependence matrix of transformed program: $D'$

Transformation matrix: $J$ (a non-singular matrix)

Dependence matrix of transformed program: $L'$. Legality: $L' \cdot D' > 0$

Matrix in which each column is a distance/direction vector.

Small complication with code generation if scaling is included.
We have studied two transformations: permutation and tiling.

Permutation and tiling are both illegal.

Exploiting temporal locality in wavefront
Skewing of inner loop by an outer loop: always legal

Skewing of inner loop by an outer loop: 

\[ \Lambda = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \]  

\[ \Omega = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \]  

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

In this example, 

\[ DT \times \begin{bmatrix} 1 \\ -1 \end{bmatrix} = D \]  

New dependence vectors: compute \( T \times D \)

(k is some fixed integer)
How do we fix this?

Dependent iterations are closer together (good) but program is illegal (bad).

Incorrect:

\[
\begin{bmatrix}
I & 0 \\
0 & I
\end{bmatrix}^{-1} = D T D = \begin{bmatrix}
I & 0 \\
0 & I
\end{bmatrix}^{-1}
\]

In this example, \( D \) is illegal.

Outer loop skewing:

\[
\begin{bmatrix}
I & 0 \\
0 & I
\end{bmatrix} \begin{bmatrix}
I & 0 \\
0 & I
\end{bmatrix}
\]

Skewing outer loop by inner loop: not necessarily illegal.
Lo op Reversal: A linear loop transformation

\[ U = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \]

\[
\begin{align*}
& \text{DO } I = 1, N \\
& \text{X}(I) = I+2 \\
& \text{X}(-U) = -U +2
\end{align*}
\]

Another example: 2-D loop, reverse inner loop

Transformation matrix

\[
\begin{bmatrix} I & -1 \\ 0 & 1 \end{bmatrix} \]

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

Code generation: easy

Loop Reversal: Linear loop transformation

\[ [I][I^{-1}] = U \]

\[
\begin{align*}
& \text{DO } I = I, N \\
& X(I) = I+2 \\
& I^{-1} = I^{-1} \\
& \text{DO } I = 1, N
\end{align*}
\]
Need for composite transformations

How do we synthesize this composite transformation?

$$\begin{bmatrix}
1 & -0 \\
1 & 1
\end{bmatrix} = \begin{bmatrix}
1 & 0 \\
1 & 1
\end{bmatrix} \ast \begin{bmatrix}
1 & -0 \\
0 & 1
\end{bmatrix}$$

Composite transformation matrix is another linear transformation

Composition of linear transformations close together

In final program, dependent iterations are

Transformation: skewing followed by reversal

$$\begin{bmatrix}
\Lambda \\
\Omega
\end{bmatrix} = \begin{bmatrix}
1 & 0 \\
1 & 1
\end{bmatrix} \ast \begin{bmatrix}
1 & 0 \\
0 & 1
\end{bmatrix}$$
Some facts about permutation/reversal/skewing:

- Code generation algorithm:
  \[
  \begin{align*}
  &D \cdot (I \cdot L \cdot L \cdot L) = ((D \cdot I) \cdot L \cdot L) \cdot L \\
  \end{align*}
  \]

- Proof:
  Legality of composite transformation $L$; check that $L$. Any unimodular matrix can be decomposed into product of

- $q \geq n - 1$: compute from

- $T$ * $L = n$: transformation

- Original bounds: $q \geq n - 1$: compute from

- $T$ * $L$ = $n$: transformation

- $T$ * $L$ = $n$: transformation

- Legality of composite transformation $L$; check that $L$. Any unimodular matrix can be decomposed into product of

- $q \geq n - 1$: compute from

- $T$ * $L$ = $n$: transformation

- $T$ * $L$ = $n$: transformation
Synthesizing composite transformations using matrix-based approaches • Enabling abstraction: dependence matrix

Rather than reason about sequences of transformations, we can reason about the single matrix that represents the composite transformation.
When we can, how do we do it?

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

Tiling is legal if all loops are legal.

In general, tiling is not legal.
Theorem: If all dependence vectors are distance vectors, we can convert entire loop nest into a fully permutable loop nest.

General idea: skew inner loops by outer loops sufficiently to make dependence matrix of transformed program must have all positive entries.

Dependence matrix of transformed program is

\[
\begin{pmatrix}
-1 & \cdot \\
1 & I
\end{pmatrix}
\]

Example: Wavetrend

Dependence matrix is non-negative.
Transformation to make first row with negative entries into row with non-negative entries:

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
find the first positive number in the corresponding column
(b) skew the row with negative entries by appropriate multiples of
rows a, b, etc.

Transformation: I

\[
\begin{pmatrix}
1 & \text{ceiling}(n/p2) & 0 & 0 & \text{ceiling}(m/p1/k/p3) & 0 & \cdots \\
0 & 1 & \text{ceiling}(m/p1/k/p3) & 0 & 0 & \cdots \\
0 & 0 & \text{ceiling}(n/p2) & 1 & \text{ceiling}(m/p1/k/p3) & 0 & \cdots \\
\end{pmatrix}
\]
General algorithm for making loop nest fully permutable:

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

Otherwise, if all entries in dependence matrix are non-negative, done.
Tiling generates a 4-deep loop nest.

Result of tiling transformed wavefront.
What happens with direction vectors?

In general, we cannot make loop nest fully permutable.

Example: 

\[
\begin{pmatrix} 
+ \\ - \\ + 
\end{pmatrix} = \mathbf{A}
\]

We try to make outermost loops fully permutable, so we would

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

Loops only.

interchange the second and third loops, and then tile the first two

Idea: algorithm will find hands of fully permutable loop nests.