

Loop Transformations for Parallelism & Locality

Previously

- Loop transformations, unimodular transformation framework
 - Loop interchange/permutation
 - Loop reversal
 - Checking transformation legality

Today

- Loop transformations and transformation frameworks
 - Loop skewing
- Using Fourier-Motzkin Elimination for code generation

Why Transformation Frameworks?

Currently

- Frameworks used *in compiler* to ...
 - abstract loops, memory accesses, and data dependences in loop
 - specify the effect of a sequence of loop transformations on the loop, its memory accesses, and its data dependences
 - generate code from the transformed loop
- Loop transformations affect the *schedule* of the loop

Future

- How can framework technology be exposed in the programming model?

Frameworks

- Unimodular
- Polyhedral
- Presburger
- Sparse Polyhedral

Frameworks for Loop Transformations

Unimodular Loop Transformations [Banerjee 90],[Wolf & Lam 91]

- can represent loop permutation, loop reversal, and loop skewing
- unimodular linear mapping (determinant of matrix is + or - 1)
 - $T i = i'$, T is a matrix, i and i' are iteration vectors

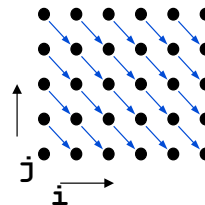
$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} i_1 \\ i_2 \end{bmatrix} = \begin{bmatrix} i'_1 \\ i'_2 \end{bmatrix}$$

- transformation is legal if the transformed dependence vector remain lexicographically positive
- limitations
 - only perfectly nested loops
 - all statements are transformed the same

Loop Skewing

Original code

```
do i = 1,6
  do j = 1,5
    A(i,j) = A(i-1,j+1)+1
  enddo
enddo
```

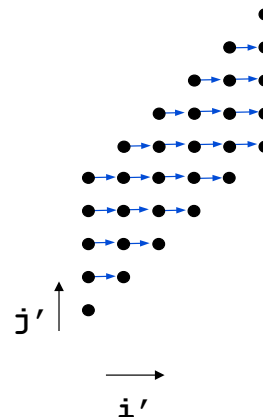


Distance vector: (1, -1)

Can we permute the original loop?

Skewing:

$$\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} i \\ i + j \end{bmatrix}$$



Transforming the Dependences and Array Accesses

Original code

```
do i = 1,6
  do j = 1,5
    A(i,j) = A(i-1,j+1)+1
  enddo
enddo
```

Dependence vector:

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

New Array Accesses:

$$A\left(\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}\right) = A(i, j)$$

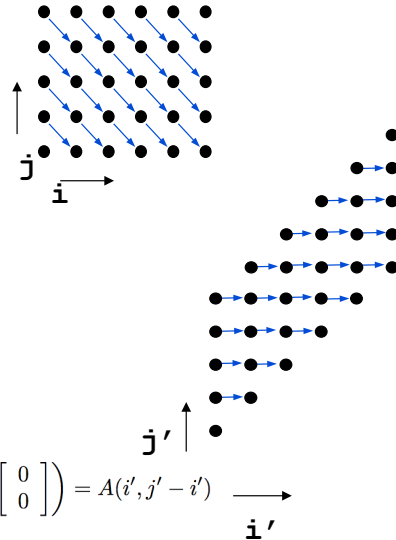
$$A\left(\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} i' \\ j' \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} i' \\ j' \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}\right) = A(i', j' - i')$$

$$A\left(\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} + \begin{bmatrix} -1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix}\right) = A(i-1, j+1)$$

$$A\left(\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} i' \\ j' \end{bmatrix} + \begin{bmatrix} -1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} i' \\ j' \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix}\right) = A(i'-1, j'-i'+1)$$

CS553 Lecture

5



Transforming the Loop Bounds

Original code

```
do i = 1,6
  do j = 1,5
    A(i,j) = A(i-1,j+1)+1
  enddo
enddo
```

Bounds:

$$\begin{bmatrix} -1 & 0 \\ 1 & 0 \\ 0 & -1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} \leq \begin{bmatrix} -1 \\ 6 \\ -1 \\ 5 \end{bmatrix}$$

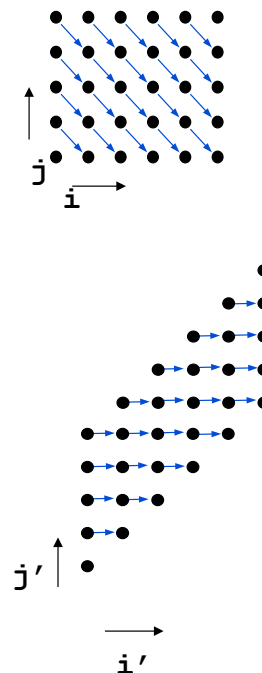
$$\begin{bmatrix} -1 & 0 \\ 1 & 0 \\ 0 & -1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} i' \\ j' \end{bmatrix} \leq \begin{bmatrix} -1 \\ 6 \\ -1 \\ 5 \end{bmatrix}$$

Transformed code

```
do i' = 1,6
  do j' = 1+i', 5+i'
    A(i', j'-i') = A(i'-1, j'-i'+1)+1
  enddo
enddo
```

CS553 Lecture

6



Code Generation

Goals

- express outermost loop bounds in terms of symbolic constants and constants
- express inner loop bounds in terms of any enclosing loop variables, symbolic constants, and constants

Approach

- Project out inner loop iteration variables to determine loop bounds for outer loops
- Fourier Motzkin elimination is the algorithm that projects a variable out of a polyhedron

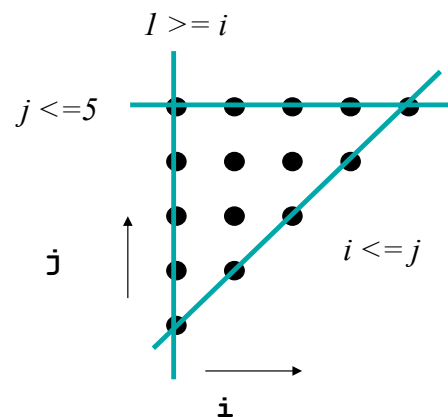
Fourier-Motzkin Elimination: The Idea

Polyhedron

- convex intersection of a set of inequalities
- model for iteration spaces

Problem

- given a polyhedron how do we generate loop bounds that scan all of its points?
- example: two possible loop orders
 - (i, j)
 - (j, i)



Fourier-Motzkin Elimination: The Algorithm

$\text{FM}(P, i_k) \Rightarrow P'$

Input: $P = \{(i_1, i_2, \dots, i_d) \mid Q\vec{i} \geq (\vec{q} + B\vec{p})\}$
 i_k such that $1 \leq k \leq d$

Output:

$$P' = \{(i_1, \dots, i_{k-1}, i_{k+1}, \dots, i_d) \mid Q'\vec{i}' \geq (\vec{q}' + B'\vec{p}')\}$$

Algorithm:

for each lower bound of $i_k, (L \leq c_1 i_k)$

$$P = P - \{L \leq c_1 i_k\}$$

for each upper bound of $i_k, (c_2 i_k \leq U)$

$$P = P - \{c_2 i_k \leq U\}$$

$$P' = P' \cup \{c_2 L \leq c_1 U\}$$

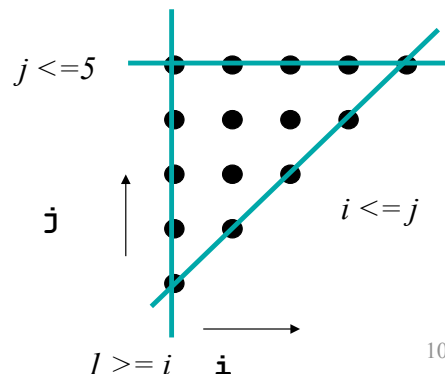
Distinguishing Upper and Lower Bounds

Simple Algorithm

– given that the polyhedron is represented as follows:

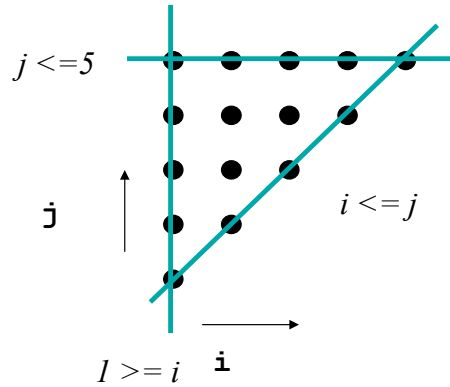
$$P = \{(i_1, i_2, \dots, i_d) \mid Q\vec{i} \geq (\vec{q} + B\vec{p})\}$$

- any constraint with a positive coefficient for i_k is a lower bound
- any constraint with a negative coefficient for i_k is an upper bound



Triangular Iteration Space Example

(i, j) for target iteration space



(j, i) for target iteration space

General Algorithm for Generating Loop Bounds

Input: $P = \{(i_1, i_2, \dots, i_d) \mid Q\vec{i} \geq (\vec{q} + B\vec{p})\}$
 where the i vector is the desired loop order

Output: $L_{i_1}, L_{i_2}, \dots, L_{i_d}$ such that $L_{i_k} = f(i_1, \dots, i_{k-1})$
 $U_{i_1}, U_{i_2}, \dots, U_{i_d}$ such that $U_{i_k} = g(i_1, \dots, i_{k-1})$

Algorithm:

$$P_n = P$$

for $k = d$ to 1 by -1

$$L_{i_k} = \text{all lower bounds for } i_k \text{ in } P_k$$

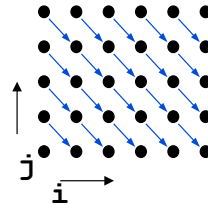
$$U_{i_k} = \text{all upper bounds for } i_k \text{ in } P_k$$

$$P_{k-1} = FM(P_k, i_k)$$

Loop Skewing and Permutation

Original code

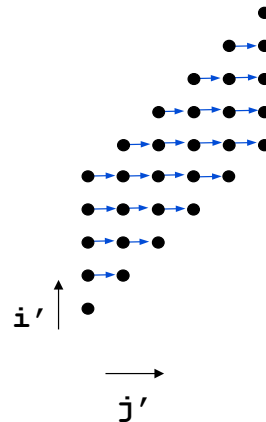
```
do i = 1,6
  do j = 1,5
    A(i,j) = A(i-1,j+1)+1
  enddo
enddo
```



Distance vector: (1, -1)

Skewing followed by Permutation:

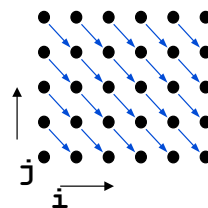
$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} i' \\ j' \end{bmatrix}$$



Transforming the Dependences and Array Accesses

Original code

```
do i = 1,6
  do j = 1,5
    A(i,j) = A(i-1,j+1)+1
  enddo
enddo
```



Dependence vector:

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

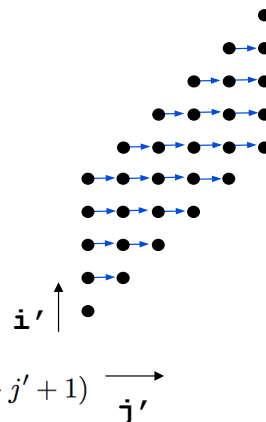
New Array Accesses:

$$A\left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}\right) = A(i, j)$$

$$A\left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} i' \\ j' \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}\right) = A(j', i' - j')$$

$$A\left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} + \begin{bmatrix} -1 \\ 1 \end{bmatrix}\right) = A(i-1, j+1)$$

$$A\left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} i' \\ j' \end{bmatrix} + \begin{bmatrix} -1 \\ 1 \end{bmatrix}\right) = A(j'-1, i'-j'+1)$$



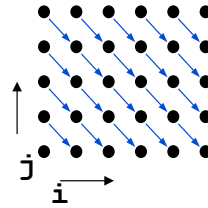
Transforming the Loop Bounds

Original code

```
do i = 1, 6
  do j = 1, 5
    A(i, j) = A(i-1, j+1) + 1
  enddo
enddo
```

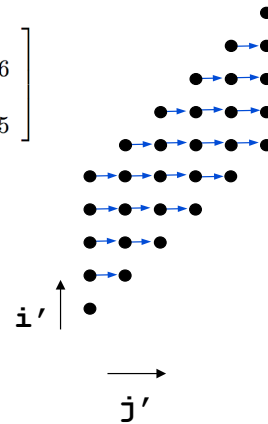
Bounds:

$$\begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} \geq \begin{bmatrix} 1 \\ -6 \\ 1 \\ -5 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} i' \\ j' \end{bmatrix} \geq \begin{bmatrix} 1 \\ -6 \\ 1 \\ -5 \end{bmatrix}$$



Transformed code (use general loop bound alg)

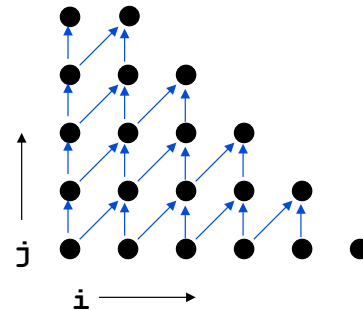
```
do i' = 2, 11
  do j' = max(i'-5, 1), min(6, i'-1)
    A(j', i'-j') = A(j'-1, i'-j'+1) + 1
  enddo
enddo
```



Wavefront Parallelism Example

Example

```
do i = 1, 6
  do j = 1, min(5, 7-i)
    A(i, j) = A(i-1, j-1)
              + A(i, j-1)
  enddo
enddo
```



Iteration Space

Goal

- Determine a unimodular transformation that enables indicating that the inner loop is fully parallel. (with an OpenMP directive for example)

$$T = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

```
do i' = 1, 5
  do j' = 1, 7-i' (parallel)
    A(j', i') = A(j'-1, i'-1)
                + A(j', i'-1)
  enddo
enddo
```

Concepts

Unimodular transformation framework

- represents loop permutation, loop reversal, and loop skewing
- provides mathematical framework for ...
 - testing transformation legality,
 - transforming array accesses and loop bounds,
 - and combining transformations

Fourier-Motzkin Elimination

- algorithm
- using for code generation

Loop bounds

- how to determine upper and lower bounds for a variable when bounds are in matrix format

Examples

- triangular matrix, skew and permute example, and wavefront example

Next Time

Lecture

- More loop transformations
- Another transformation framework