## Loop Transformations for Parallelism & Locality

**Last week**
- Data dependences and loops
- Loop transformations
  - (Parallelization)
  - Scalar expansion
- Value data dependences

**Today and Monday**
- Loop transformations and transformation frameworks
  - Loop reversal
  - Loop fusion
  - Loop fission
  - Loop interchange
  - Unroll and Jam



## Review

**Distance vectors**
- Concisely represent dependences in loops (*i.e.*, in iteration spaces)
- Dictate what transformations are legal
  - *e.g.,* Permutation and parallelization

**Legality**
- A dependence vector is **legal** when it is lexicographically nonnegative

**Loop-carried dependence**
- A dependence $D=(d_1,\ldots d_n)$ is **carried** at loop level $i$ if $d_i$ is the first nonzero element of D

## Loop Permutation

**Idea**

- Swap the order of two loops to increase parallelism, to improve spatial locality, or to enable other transformations
- Also known as **loop interchange**

**Example**

```
do i = 1,n
    do j = 1,n
       x = A(2,j)
    enddo
enddo
```

This access strides through a row of A

→

```
do j = 1,n
    do i = 1,n
       x = A(2,j)
    enddo
enddo
```

This code is invariant with respect to the inner loop, yielding better locality



## Loop Interchange (cont)

**Example**

```
do i = 1,n
    do j = 1,n
       x = A(i,j)
    enddo
enddo
```

This array has stride n access

→

```
do j = 1,n
    do i = 1,n
       x = A(i,j)
    enddo
enddo
```

This array now has stride 1 access

(Assuming column-major order for Fortran)

## Legality of Loop Interchange

**Case analysis of the direction vectors**

**(=,=)**

The dependence is loop independent, so it is unaffected by interchange

**(=,<)**

The dependence is carried by the j loop.
After interchange the dependence will be (<,=), so the dependence will still be carried by the j loop, so the dependence relations do not change.

**(<,=)**

The dependence is carried by the i loop.
After interchange the dependence will be (=,<), so the dependence will still be carried by the i loop, so the dependence relations do not change.



## Legality of Loop Interchange (cont)

**Case analysis of the direction vectors (cont.)**

**(<,<)**

The dependence distance is positive in both dimensions.
After interchange it will still be positive in both dimensions, so the dependence relations do not change.

**(<,>)**

The dependence is carried by the outer loop.
After interchange the dependence will be (>,<), which changes the dependences and results in an illegal direction vector, so interchange is illegal.

**(>,*) (=,>)**

Such direction vectors are not possible for the original loop.

## Loop Interchange Example

**Consider the (<,>) case**

```
do i = 1,n
   do j = 1,n
      C(i,j) = C(i+1,j-1)
   enddo
enddo
```

⇨

```
do j = 1,n
   do i = 1,n
      C(i,j) = C(i+1,j-1)
   enddo
enddo
```

**Before**

(1,1) C(1,1) = C(2,0)
(1,2) C(1,2) = C(2,1)
. . . d = (<,>) $\delta^a$
(2,1) C(2,1) = C(3,0)

**After**

(1,1) C(1,1) = C(2,0)
(2,1) C(2,1) = C(3,0)
. . . d = (>,<) $\delta^f$
(1,2) C(1,2) = C(2,1)



## 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

## Legality of Loop Interchange, Reprise

**Reduced case analysis of the direction vectors** $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} j \\ i \end{bmatrix}$

**(=,=)**

The dependence is loop independent, so it is unaffected by interchange

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

**(=,<)**

The dependence is carried by the j loop.

After interchange the dependence will be (<,=), so the dependence will still be carried by the j loop, so the dependence relations do not change.

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

**(<,>)**

The dependence is carried by the outer loop.

After interchange the dependence will be (>,<), which changes the dependences and results in an illegal direction vector, so interchange is illegal.

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} < \\ > \end{bmatrix} = \begin{bmatrix} > \\ < \end{bmatrix}$$



## Loop Reversal

**Idea**

- Change the direction of loop iteration
  (*i.e.,* From low-to-high indices to high-to-low indices or vice versa)

**Benefits**

- Improved cache performance
- Enables other transformations (coming soon)

**Example**

```
do i = 6,1,-1
    A(i) = B(i) + C(i)
enddo
```

→

```
do i = 1,6
    A(i) = B(i) + C(i)
enddo
```

## Loop Reversal and Distance Vectors

**Impact**

– Reversal of loop *i* negates the $i^{th}$ entry of all distance vectors associated with the loop

– What about direction vectors?

**When is reversal legal?**

– When the loop being reversed does not carry a dependence
(*i.e.*, When the transformed distance vectors remain legal)

**Example**

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

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

Dependence: Flow
Distance Vector: (1,1)
Transformed
Distance Vector: (1,-1) legal



## Loop Reversal Example

**Legality**

– Loop reversal will change the direction of the dependence relation

**Is the following legal?**

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

Dependence: Flow
Distance Vector: (1)

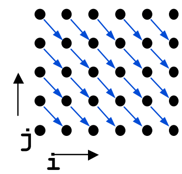
```
do i = 6,1,-1
    A(i) = A(i-1)
enddo
```

Dependence: Anti Flow
Distance Vector: (1) (−1)

## Loop Skewing

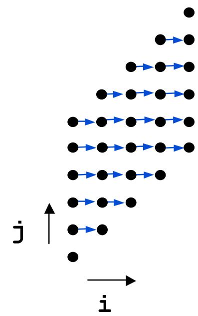
**Original code**

```
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:**

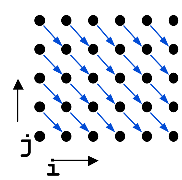
$$\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} i \\ i+j \end{bmatrix}$$



## Transforming the Dependences and Array Accesses

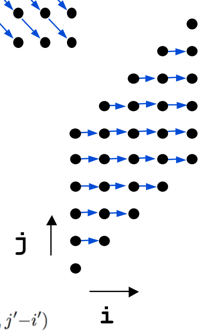
**Original code**

```
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:**

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

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

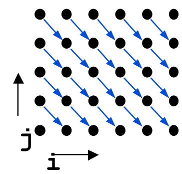
$$A\left(\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} i \\ j \end{bmatrix} + \begin{bmatrix} -1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} i \\ j \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix}\right) = A(i-1, j+1)$$

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

## 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
```



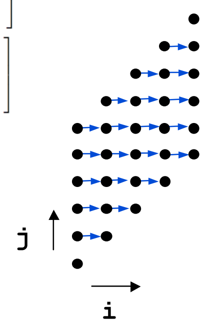

**Bounds:**

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

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

**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
```






## Loop Fusion

**Idea**

- Combine multiple loop nests into one

**Example**

```
do i = 1,n
    A(i) = A(i-1)
enddo
do j = 1,n
    B(j) = A(j)/2
enddo
```

→

```
do i = 1,n
    A(i) = A(i-1)
    B(i) = A(i)/2
enddo
```

**Pros**

- May improve data locality
- Reduces loop overhead
- Enables **array contraction** (opposite of scalar expansion)
- May enable better instruction scheduling

**Cons**

- May hurt data locality
- May hurt icache performance

## Legality of Loop Fusion

**Basic Conditions**

- Both loops must have same structure
  - Same loop depth
  - Same loop bounds
  - Same iteration directions

  Can we relax any of these restrictions?
- Dependences must be preserved
  *e.g.,* Flow dependences must not become anti dependences

```
do i = 1,n
    body1
enddo
do i = 1,n
    body2
enddo
```

All cross-loop dependences flow from body1 to body2

```
do i = 1,n
    body1
    body2
enddo
```

Ensure that fusion does not introduce dependences from body2 to body1



## Loop Fusion Example

**What are the dependences?**

```
    do i = 1,n
s1      A(i) = B(i) + 1
    enddo

    do i = 1,n
s2      C(i) = A(i)/2
    enddo

    do i = 1,n
s3      D(i) = 1/C(i+1)
    enddo
```

$s_1\delta^f s_2$

$s_2\delta^f s_3$

**What are the dependences?**

```
    do i = 1,n
s1      A(i) = B(i) + 1
s2      C(i) = A(i)/2
s3      D(i) = 1/C(i+1)
    enddo
```

$s_1\delta^f s_2$

$s_3\delta^a s_2$

Fusion changes the dependence between $s_2$ and $s_3$, so fusion is illegal

**Is there some transformation that will enable fusion of these loops?**

## Loop Fusion Example (cont)

**Loop reversal is legal for the original loops**

- Does not change the direction of any dep in the original code
- Will reverse the direction in the fused loop: $s_3\delta^a\ s_2$ will become $s_2\delta^f\ s_3$

```
    do i = n,1
s1      A(i) = B(i) + 1
    enddo
                        s1 δf s2
    do i = n,1
s2      C(i) = A(i)/2
    enddo
                        s2 δf s3
    do i = n,1
s3      D(i) = 1/C(i+1)
    enddo
```

⇒

```
    do i = n,1,-1
s1      A(i) = B(i) + 1
                  s1 δf s2
s2      C(i) = A(i)/2
                  s2 δf s3
s3      D(i) = 1/C(i+1)
    enddo
```

After reversal and fusion all original dependences are preserved



## Concepts

**Using direction and distance vectors**

**Transformations:**

- What is the benefit?
- What do they enable?
- When are they legal?

**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

*** The example did not require Fourier Motzkin elimination.**

## Next Time

**Lecture**

– More loop transformations
– An even cooler transformation framework