

Loop Transformations for Parallelism & Locality

Last week

- Data dependences and loops
- Loop transformations
 - Parallelization
 - Loop interchange

Today

- Scalar expansion for removing false dependences
- Loop interchange
- Loop transformations and transformation frameworks
 - Loop permutation
 - Loop reversal
 - Loop skewing
 - Loop fusion

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 $D=(d_1, \dots, d_n)$ is **carried** at loop level i if d_i is the first nonzero element of D

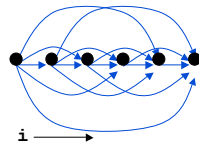
Scalar Expansion: Motivation

Problem

- Loop-carried dependences inhibit parallelism
- Scalar references result in loop-carried dependences

Example

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



Can this loop be parallelized? No.

What kind of dependences are these? Anti dependences.

Convention for these slides: Arrays start with upper case letters, scalars do not

Scalar Expansion

Idea

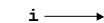
- Eliminate false dependences by introducing extra storage

Example

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



Can *this* loop be parallelized?



Disadvantages?

Scalar Expansion Details

Restrictions

- The loop must be a **countable** loop
i.e. The loop trip count must be independent of the body of the loop
- The expanded scalar must have no **upward exposed uses** in the loop

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

- Nested loops may require much more storage
- When the scalar is live after the loop, we must move the correct array value into the scalar

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)
 ...
 (2,1) C(2,1) = C(3,0) δ^a

After

(1,1) C(1,1) = C(2,0)
 (2,1) C(2,1) = C(3,0) δ^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
```

CS553 Lecture

Loop Transformations

14

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

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

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

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

CS553 Lecture

Loop Transformations

15

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
```

↓

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

Dependence: Flow
Distance Vector: (1)

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

CS553 Lecture

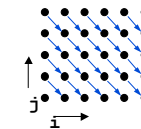
Loop Transformations

16

Loop Skewing

Original code

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

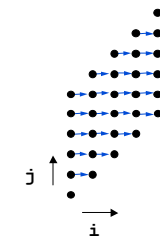


Distance vector: (1, -1)

Can we permute the original loop?

Skewing:

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



CS553 Lecture

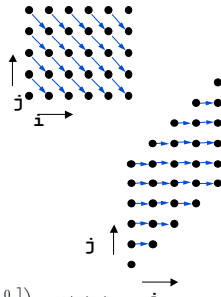
Loop Transformations

17

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:

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

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

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

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

CS553 Lecture

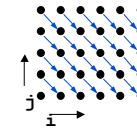
Loop Transformations

18

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:

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

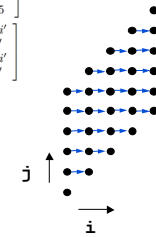
$$\begin{bmatrix} -1 & 0 \\ 1 & 0 \\ 0 & -1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} i' \\ j' \end{bmatrix} \leq \begin{bmatrix} -1 \\ 6 \\ -1 \\ 5 \end{bmatrix}$$

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

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

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
```



CS553 Lecture

Loop Transformations

19

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

CS553 Lecture

Loop Transformations

20

Legality of Loop Fusion

Basic Conditions

- Both loops must have same structure
 - Same loop depth
 - Same loop bounds
 - Same iteration directions
 - Dependences must be preserved
- e.g., Flow dependences must not become anti dependences
- Can we relax any of these restrictions?

```
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

CS553 Lecture

Loop Transformations

21

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
    
```



What are the dependences?

```

do i = 1, n
s1   A(i) = B(i) + 1
      |
      | s1δf s2
      v
s2   C(i) = A(i) / 2
      |
      | s3δa s2
      v
s3   D(i) = 1/C(i+1)
enddo
    
```

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

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

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



```

do i = n, 1, -1
s1   A(i) = B(i) + 1
      |
      | s1δf s2
      v
s2   C(i) = A(i) / 2
      |
      | s2δf s3
      v
s3   D(i) = 1/C(i+1)
enddo
    
```

After reversal and fusion all original dependences are preserved

Concepts

Using direction and distance vectors

Transformation legality (from previous)

- must respect data dependences
- scalar expansion as a technique to remove anti and output dependences

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

Next Time

Lecture

- More loop transformations
- An even cooler transformation framework