

Loop Transformations for Parallelism & Locality

Previously

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

Today


- Formalizing the legality of loop transformations
- Loop interchange/permutation
- Loop transformations and transformation frameworks
 - Loop permutation
 - Loop reversal

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

<pre>do i = 1,n do j = 1,n x = A(2,j) enddo enddo</pre>		<pre>do j = 1,n do i = 1,n x = A(2,j) enddo enddo</pre>
<p>This access strides through a row of A</p>		<p>This code is invariant with respect to the inner loop, yielding better locality</p>

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)
  
```

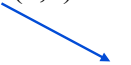


δ^a

After

```

(1,1) C(1,1) = C(2,0)
(2,1) C(2,1) = C(3,0)
...
(1,2) C(1,2) = C(2,1)
  
```



δ^f

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/Permutation, 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

- Could improve 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

```
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**

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)

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
- Code generation for transformed iteration space