Loop Transformations for Parallelism & Locality

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

Today
- Loop interchange
- Loop transformations and transformation frameworks
  - Loop permutation
  - Loop reversal
  - Loop skewing
  - Loop fusion

Loop Interchange (cont)

Example

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

(Assuming column-major order for Fortran)

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 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

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.
### Legal 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.

\(>,>\) \(\sim ,>\)

Such direction vectors are not possible for the original loop.

### Loop Interchange Example

Consider the \( <,> \) case

\[
\begin{align*}
&\quad \quad \text{do } i = 1,n \\
&\quad \quad \text{do } j = 1,n \\
&\quad \quad \quad \quad C(1,1) = C(2,1) \\
&\quad \quad \quad \quad \vdots \\
&\quad \quad \quad \quad C(2,1) = C(3,0) \\
&\quad \quad \text{enddo} \\
&\quad \text{enddo}
\end{align*}
\]

Before

\[
(1,1) \rightarrow (2,0) \\
(1,2) \rightarrow (2,1) \\
\ldots \\
(2,1) \rightarrow (3,0)
\]

After

\[
(1,1) \rightarrow (3,0) \\
(1,2) \rightarrow (2,1) \\
\ldots \\
(2,1) \rightarrow (3,0)
\]

### 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**

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

\( =,= \)

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

\[
\begin{bmatrix}
0 & 1 \\
1 & 0
\end{bmatrix}
\begin{bmatrix}
\delta f \\
\delta a
\end{bmatrix}
= 
\begin{bmatrix}
0 \\
0
\end{bmatrix}
\]

\( =,< \)

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.

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

\( <,> \)

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

Dependence: \( \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \)
Flow Distance Vector: \( (1) \)
Transformed Distance Vector: \( (1,-1) \)
Legal

Loop Reversal Example

Legality
– Loop reversal will change the direction of the dependence relation

Is the following legal?
```fortran
  do i = 1,6
    A(i) = A(i-1)
  enddo
  do i = 6,1,-1
    A(i) = A(i-1)
  enddo
```

Dependence: \( (1) \)
Flow Distance Vector: \( (1) \)

Loop Skewing

Original code
```fortran
  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:
```plaintext
\[
\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} i+j \\ j \end{bmatrix}
\]"
Transforming the Dependencies and Array Accesses

Original code
\[
\begin{align*}
d & = 1,6 \\
d & = 1,5 \\
A(i,j) &= A(i-1,j+1) + 1 \\
\end{align*}
\]

Dependence vector:
\[
\begin{bmatrix}
1 & 0 \\
1 & 1 \\
1 & -1
\end{bmatrix} = 
\begin{bmatrix}
1 \\
0
\end{bmatrix}
\]

New Array Accesses:
\[
\begin{align*}
A(i,j) &= A(i-1,j+1) + 1 \\
A(i-1,j+1) &= A(i,j) \\
A(i-1,j+1) &= A(i,j-1,j+1)
\end{align*}
\]

Transforming the Loop Bounds

Original code
\[
\begin{align*}
d & = 1,6 \\
d & = 1,5 \\
A(i,j) &= A(i-1,j+1) + 1 \\
\end{align*}
\]

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

Transformed code
\[
\begin{align*}
d & = 1,6 \\
d & = 1,5 \\
A(i',j') &= A(i,j) \\
A(i,j) &= A(i-1,j+1) + 1 \\
\end{align*}
\]

Loop Fusion

Idea
– Combine multiple loop nests into one

Example
\[
\begin{align*}
do & = 1,n \\
A(i) &= A(i-1) \\
enddo \\
do & = 1,n \\
B(j) &= A(j)/2 \\
enddo
\end{align*}
\]

Pros
– May improve data locality
– Reduces loop overhead
– Enables array contraction (opposite of scalar expansion)

Cons
– May hurt data locality
– May hurt cache performance
– May enable better instruction scheduling

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
  \textit{e.g.}, Flow dependences must not become anti dependences

Can we relax any of these restrictions?

Ensure that fusion does not introduce dependences from body2 to body1
**Loop Fusion Example**

What are the dependences?

```c
for i = 1 to n
    A(i) = B(i) + 1
    C(i) = A(i) / 2
    D(i) = 1 / C(i + 1)
end for
```

What are the dependences?

```c
for i = 1 to n
    A(i) = B(i) + 1
    C(i) = A(i) / 2
    D(i) = 1 / C(i + 1)
end for
```

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?

---

**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
- Another transformation framework