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**

```plaintext
do i = 1,n  
do j = 1,n  
x = A(2,j)  
enddo
enddo
This access strides through a row of A
```

**Example (cont)**

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

**Loop Interchange (cont)**

**Example**

```plaintext
do i = 1,n  
do j = 1,n  
x = A(i,j)  
enddo
enddo
This array has stride n access
```

```plaintext
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

\[
\begin{align*}
d & = (><) \\
\delta^a & = (><) \\
d & = (<,>)
\end{align*}
\]

Before:

\[
\begin{align*}
(1,1) & : C(1,1) = C(2,0) \\
(1,2) & : C(1,2) = C(2,1) \\
\ldots & \\
(2,1) & : C(2,1) = C(3,0)
\end{align*}
\]

After:

\[
\begin{align*}
(1,1) & : C(1,1) = C(2,0) \\
(2,1) & : C(2,1) = C(3,0) \\
\ldots & \\
(1,2) & : C(1,2) = C(2,1)
\end{align*}
\]

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

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

\((=,\prec)\)

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

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

\((\prec,\succ)\)

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

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

Loop Reversal

Idea

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

Benefits

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

Example

\[
\begin{align*}
do & \ i = 6,1,-1 \\
& \quad A(i) = B(i) + C(i) \\
\end{align*}
\]

\[
\begin{align*}
do & \ i = 1,6 \\
& \quad A(i) = B(i) + C(i) \\
\end{align*}
\]
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

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

<table>
<thead>
<tr>
<th>Dependence</th>
<th>Distance Vector</th>
<th>Transformed Distance Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flow</td>
<td>(1,1)</td>
<td>(1,-1) legal</td>
</tr>
</tbody>
</table>

Loop Reversal Example

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

Is the following legal?

\[
\begin{align*}
do i &= 1, 6 \\
A(i) &= A(i-1) \\
\end{align*}
\]

Dependence: Flow
Distance Vector: (1)

\[
\begin{align*}
do i &= 6, 1, -1 \\
A(i) &= A(i-1) \\
\end{align*}
\]

Dependence: Anti Flow
Distance Vector: (1, -1)
### Loop Skewing

Original code

```plaintext
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

```plaintext
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:

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

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

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

\[
A\left(\begin{bmatrix}
1 & 0 \\
0 & 0
\end{bmatrix}\cdot\begin{bmatrix}
i \\
0
\end{bmatrix} + \begin{bmatrix}
0 & 0 \\
1 & 1
\end{bmatrix}\cdot\begin{bmatrix}
1 \\
0
\end{bmatrix} + \begin{bmatrix}
0 & 0 \\
0 & 0
\end{bmatrix}\right) = A(i-1,j-1)
\]
**Transforming the Loop Bounds**

**Original code**

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

**Bounds:**

\[
\begin{pmatrix}
1 & 0 \\
0 & -1 \\
1 & 0 \\
0 & -1
\end{pmatrix}
\begin{pmatrix}
i \\
j
\end{pmatrix}
\leq
\begin{pmatrix}
-1 & 0 & 0 & 1 \\
0 & 1 & 1 & 0 \\
1 & 0 & -1 & 0 \\
0 & 1 & 1 & 0
\end{pmatrix}
\begin{pmatrix}
\ell' \\
\ell
\end{pmatrix}
\leq
\begin{pmatrix}
-1 & 0 & 0 & 1 \\
0 & 1 & 1 & 0 \\
1 & 0 & -1 & 0 \\
0 & 1 & 1 & 0
\end{pmatrix}
\begin{pmatrix}
i \\
j
\end{pmatrix}
\leq
\begin{pmatrix}
6 & 5 & 5 & 5 \\
-1 & -1 & -1 & -1 \\
6 & 1 & 1 & 1 \\
-1 & -1 & -1 & -1
\end{pmatrix}
\]

**Transformed code**

```plaintext
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**

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

- Dependences must be preserved
  *e.g.*, Flow dependences must not become anti dependences

Can we relax any of these restrictions?

<table>
<thead>
<tr>
<th>do i = 1,n</th>
<th>body1</th>
<th>All cross-loop dependences flow from body1 to body2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>body2</td>
<td></td>
</tr>
</tbody>
</table>

Ensure that fusion does not introduce dependences from body2 to body1

<table>
<thead>
<tr>
<th>do i = 1,n</th>
<th>body1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>body2</td>
<td>Ensure that fusion does not introduce dependences from body2 to body1</td>
</tr>
</tbody>
</table>

**Loop Fusion Example**

What are the dependences?

1. \( s_1 \): \( A(i) = B(i) + 1 \)
2. \( s_2 \): \( C(i) = A(i)/2 \)
3. \( s_3 \): \( D(i) = 1/C(i+1) \)

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: $s_3\delta^f s_2$ will become $s_2\delta^f s_3$

```latex
\begin{align*}
\text{do } i \text{ = } n,1 & \\
& \quad \text{A}(i) = \text{B}(i) + 1 \\
& \quad \text{enddo} \\
\text{do } i \text{ = } n,1 & \\
& \quad \text{C}(i) = \text{A}(i)/2 \\
& \quad \text{enddo} \\
\text{do } i \text{ = } n,1 & \\
& \quad \text{D}(i) = 1/\text{C}(i+1) \\
& \quad \text{enddo}
\end{align*}
```

```latex
\begin{align*}
\text{do } i \text{ = } n,1 & \\
& \quad \text{A}(i) = \text{B}(i) + 1 \\
& \quad \text{enddo} \\
\text{do } i \text{ = } n,1 & \\
& \quad \text{C}(i) = \text{A}(i)/2 \\
& \quad \text{enddo} \\
\text{do } i \text{ = } n,1 & \\
& \quad \text{D}(i) = 1/\text{C}(i+1) \\
& \quad \text{enddo}
\end{align*}
```

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