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

```
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
**Legality of Loop Interchange**

**Case analysis of the direction vectors**

- 
  
  \((\leq,\leq)\)
  
  The dependence is loop independent, so it is unaffected by interchange.

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

- 
  
  \((<,\leq)\)
  
  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

\[
\begin{align*}
\text{Before} & \\
(1,1) & \quad C(1,1) = C(2,0) \\
(1,2) & \quad C(1,2) = C(2,1) \\
\vdots & \\
(2,1) & \quad C(2,1) = C(3,0)
\end{align*}
\]

\[
\begin{align*}
\text{After} & \\
(1,1) & \quad C(1,1) = C(2,0) \\
(2,1) & \quad C(2,1) = C(3,0)
\end{align*}
\]

Frameworks for Loop Transformations

Unimodular Loop Transformations [Banerjee 90],[Wolf & Lam 91]

\begin{itemize}
\item can represent loop permutation, loop reversal, and loop skewing
\item unimodular linear mapping (determinant of matrix is + or - 1)
\item \(T_i = i', T\) is a matrix, i and i’ are iteration vectors
\end{itemize}

\[
\begin{bmatrix}
1 & 0 \\
0 & 1
\end{bmatrix}
\begin{bmatrix}
i_1 \\
i_2
\end{bmatrix}
=
\begin{bmatrix}
i'_1 \\
i'_2
\end{bmatrix}
\]

\begin{itemize}
\item transformation is legal if the transformed dependence vector remain lexicographically positive
\item limitations
\item only perfectly nested loops
\item all statements are transformed the same
\end{itemize}
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
  \((i.e., \text{From low-to-high indices to high-to-low indices or vice versa})\)

Benefits
– Could improve cache performance
– Enables other transformations (coming soon)

Example

\[
\begin{array}{ll}
\text{do } i = 6,1,-1 & \text{do } i = 1,6 \\
A(i) = B(i) + C(i) & A(i) = B(i) + C(i) \\
\text{enddo} & \text{enddo}
\end{array}
\]
**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>Flow</th>
<th>Distance Vector:</th>
<th>Transformed Distance Vector:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></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?**

\[
do \ i = 1, 6 \\
\quad A(i) = A(i-1) \\
enddo
\]

<table>
<thead>
<tr>
<th>Dependence</th>
<th>Flow</th>
<th>Distance Vector:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>(1)</td>
</tr>
</tbody>
</table>

\[
do \ i = 6, 1, -1 \\
\quad A(i) = A(i-1) \\
enddo
\]

<table>
<thead>
<tr>
<th>Dependence</th>
<th>Anti Flow</th>
<th>Distance Vector:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>(1) (-1)</td>
</tr>
</tbody>
</table>
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