### Loop-Carried Dependences

**Definition**
- A dependence \( D=(d_1,...,d_n) \) is **carried** at loop level \( i \) if \( d_i \) is the first non-zero element of \( D \)

**Example**
```plaintext
do i = 1,5
  do j = 2,6
    A(j,i) = B(j-1,i) + 1
    B(j,i) = A(j,i-1) * 2
  enddo
enddo
```

**Distance vectors:**
- \((1,0)\) for accesses to \( A \)
- \((0,1)\) for accesses to \( B \)

**Loop-carried dependences**
- The \( i \) loop carries dependence due to \( A \)
- The \( j \) loop carries dependence due to \( B \)

---

### Parallelization

**Idea**
- The iterations of a loop can be executed in parallel if the loop carries no dependences

**Example**
```plaintext
do i = 1,5
  do j = 2,6
    A(j,i) = B(j-1,i-1) + 1
    B(j,i) = A(j,i-1) * 2
  enddo
enddo
```

**Can we parallelize the \( i \) loop?**
- \((1,0)\) for \( A \) (flow)
- \((1,1)\) for \( B \) (flow)
Parallelization (cont)

Idea
- The iterations of a loop can be executed in parallel if the loop carries no dependences

Example
\[
\begin{align*}
d & \text{do } i = 1,5 \\
& \quad \text{do } j = 2,6 \\
& \quad \quad A(j,i) = B(j-1,i-1)+1 \\
& \quad \quad B(j,i) = A(j,i-1)*2 \\
& \quad \text{enddo} \\
& \text{enddo}
\end{align*}
\]

Can we instead parallelize the \( j \) loop?

Scalar Expansion: Motivation

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

Example
\[
\begin{align*}
d & \text{do } i = 1,6 \\
& \quad t = A(i) + B(i) \\
& \quad C(i) = t + 1/t \\
& \text{enddo}
\end{align*}
\]

Can this loop be parallelized? No.
What kind of dependences are these? Anti dependences.
Scalar Expansion

Idea
– Eliminate false dependences by introducing extra storage

Example

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

Can this loop be parallelized? Yes.

Disadvantages?

Scalar Expansion Details

Restrictions
– The loop must be a countable loop
  \textit{i.e.} The loop trip count must be independent of the body of the loop
– There can not be loop-carried flow dependences due to the scalar
– The expanded scalar must have no upward exposed uses in the loop
  \[
  \begin{align*}
  &\text{do } i = 1, 6 \\
  &\quad \text{print}(t) \\
  &\quad t = A(i) + B(i) \\
  &\quad C(i) = t + 1/t
  \end{align*}
  \]
  – When the scalar is live after the loop, we must move the correct array value into the scalar
  – Nested loops may require much more storage
Example 2: Parallelization (reprise)

**Why can’t this loop be parallelized?**

```plaintext
do i = 1, 100
    A(i) = A(i-1) + 1
enddo
```

**Distance Vector:** (1)

**Why can this loop be parallelized?**

```plaintext
do i = 1, 100
    A(i) = A(i) + 1
enddo
```

**Distance Vector:** (0)

---

Example 1: Loop Permutation (reprise)

**Sample code**

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

**Why is this legal?**

- There are no loop-carried dependences, so we can arbitrarily change the order of iteration

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

Consider the following code

\[
\text{do } i = 1, 5 \\
\quad A(3i+2) = A(2i+1) + 1 \\
\text{enddo}
\]

Question

- How do we determine whether one array reference depends on another across iterations of an iteration space?

Dependence Testing in General

General code

\[
\text{do } i_1 = l_1, h_1 \\
\quad \ldots \\
\quad \text{do } i_n = l_n, h_n \\
\quad \quad A(f(i_1, \ldots, i_n)) = \ldots A(g(i_1, \ldots, i_n)) \\
\quad \text{enddo} \\
\quad \ldots \\
\text{enddo}
\]

There exists a dependence between iterations I=(i_1, ..., i_n) and J=(j_1, ..., j_n) when

- \(f(I) = g(J)\)
- \((l_1, \ldots, l_n) < I, J < (h_1, \ldots, h_n)\)
Dependence Testing: Simple Case

Sample code

```plaintext
do i = l, h
    A(a*i+c₁) = ... A(a*i+c₂)
enddo
```

Dependence?

\[a*i₁+c₁ = a*i₂+c₂, \text{ or}\]
\[a*i₁ - a*i₂ = c₂ - c₁\]

A solution exists if \(a\) divides \(c₂ - c₁\) evenly.

Exercise

Code

```plaintext
do i = l, h
    A(2*i+2) = A(2*i-2)+1
enddo
```

Dependence?

\[2*i₁ - 2*i₂ = -2 - 2 = -4\]
\[i₁ - i₂ = -2\] (yes, 2 divides -4)

Kind of dependence?

- Anti? \(i₂ + d = i₁\)  \(\Rightarrow\) \(d = -2\)

- Flow? \(i₁ + d = i₂\)  \(\Rightarrow\) \(d = 2\)
GCD Test

Idea
- Generalize test to linear functions of iterators

Code
```
do i = l_i, h_i
  do j = l_j, h_j
    A(a_1*i + a_2*j + a_0) = ... A(b_1*i + b_2*j + b_0) ...
  enddo
enddo
```

Again
- a_1*i_1 - b_1*i_2 + a_2*j_1 - b_2*j_2 = b_0 - a_0
- Solution exists if gcd(a_1, a_2, b_1, b_2) divides b_0 - a_0

Example

Code
```
do i = l_i, h_i
  do j = l_j, h_j
    A(4*i + 2*j + 1) = ... A(6*i + 2*j + 4) ...
  enddo
enddo
```

\( \text{gcd}(4, -6, 2, -4) = 2 \)

Does 2 divide 4-1?
Next Time

Lecture
- Loop transformations

Loop Transformations for Parallelism & Locality

Last time
- Data dependences and loops
- Loop transformations
  - Parallelization
  - Scalar expansion

Today
- Loop transformations
  - 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

**Direction vectors**
- Compare $i^S$ and $i^T$: $<$, $>$, $=$

**Legality**
- A dependence vector is **legal** when it is lexicographically nonnegative

**Loop-carried dependence**
- A dependence $D=(d_1,...,d_n)$ is **carried** at loop level $i$ if $d_i$ is the first nonzero element of $D$

---

**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**

```plaintext
do i = 6,1,-1
   A(i) = B(i) + C(i)
endo
```

```plaintext
do i = 1,6
   A(i) = B(i) + C(i)
endo
```

Loop Reversal and Distance Vectors

**Impact**
- Reversal of loop $i$ negates the $i^{th}$ entry of all distance vectors associated with the loop
- What does it do to 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**

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

*Dependence:* Flow
*Distance Vector:* (1,1)

**Can either loop be reversed?**

Loop Reversal Example

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

**Is the following legal?**

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

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

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

*Dependence:* Anti
*Distance Vector:* (1)
Loop Fusion

Idea
– Combine multiple loop nests into one

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

Pros?
– May improve data locality
– Reduces loop overhead
– May enable better instruction scheduling
– Enables array contraction (opposite of scalar expansion)

Cons?
– May hurt i-cache performance
– May hurt data locality

Loop Fusion Can Hurt Locality

Example
\[
\begin{align*}
d & \text{do } j = 1,n \\
& \quad \text{do } i = 1,m \\
& \quad B(i,j) = A(i,j) + A(i,j-1) + A(i,j-2) \\
& \text{enddo} \\
& \text{enddo} \\
& \text{do } j = 1,n \\
& \quad \text{do } i = 1,m \\
& \quad C(i,j) = B(i,j) + D(i,j) \\
& \text{enddo} \\
& \text{enddo}
\end{align*}
\]
Loop Fusion Can Hurt Locality (cont)

After fusion

\[
\begin{align*}
\text{do } j &= 1,n \\
\text{do } i &= 1,m \\
B(i,j) &= A(i,j) + A(i,j-1) + A(i,j-2) \\
C(i,j) &= B(i,j) + D(i,j)
\end{align*}
\]

Lost reuse

Saved loads

Legality of Loop Fusion

Basic Requirements

- 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?

\[
\begin{align*}
\text{do } i &= 1,n \\
\text{body}_1 \\
\text{endo} \\
\text{do } i &= 1,n \\
\text{body}_2 \\
\text{endo}
\end{align*}
\]

All cross-loop dependences flow from body1 to body2

\[
\begin{align*}
\text{do } i &= 1,n \\
\text{body}_1 \\
\text{endo} \\
\text{do } i &= 1,n \\
\text{body}_2 \\
\text{endo}
\end{align*}
\]

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

What are the dependences?

```plaintext
do i = 1,n
A(i) = B(i) + 1
enddo

s1

s2
C(i) = A(i)/2
enddo

s2

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

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 dependence in the original code
- Will reverse the direction in the fused loop: $s_3 \delta f s_2$ will become $s_2 \delta f s_3$

```plaintext
do i = n,1
A(i) = B(i) + 1
enddo

s1

s2
C(i) = A(i)/2
enddo

s2

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

After reversal and fusion all original dependences are preserved
Loop Fission (Loop Distribution)

**Idea**
- Split a loop nest into multiple loop nests (the inverse of fusion)

**Example**
```plaintext
do i = 1,n  do i = 1,n
   A(i) = B(i) + 1  A(i) = B(i) + 1
   C(i) = A(i)/2  C(i) = A(i)/2
endo
endo```

**Motivation?**
- Produces multiple (potentially) less constrained loops
- May improve locality
- Enable other transformations, such as interchange

**Legality?**
April 29, 2015 Loop Transformations

Loop Fission (cont)

**Legality**
- Fission is legal when the loop body contains no cycles in the dependence graph

```plaintext
do i = 1,n  do i = 1,n
   body1  body1
   enddo
   enddo
   Cycles cannot be preserved because after fission all cross-loop dependences flow from body1 to body2```

April 29, 2015 Loop Transformations