Compiling for Parallelism & Locality

Last time
– Predication and speculation

Today
– Data dependences and loops
– Parallelism and locality

Example 1: Loop Permutation for Improved Locality

What do we notice about the following code?
– Assume Fortran’s Column Major Order array layout

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

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

poor cache locality

good cache locality
Example 2: Parallelization

Can we parallelize the following loop?
– Can we execute the different iterations at the same time?

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

Yes

How about this loop?

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

No

Data Dependences

Recall
– A data dependence defines an ordering relationship between two statements
– In executing statements, data dependences must be respected to preserve correctness

Example

```
s1  a := 5;
s2  b := a + 1;
s3  a := 6;
```

```
s1  a := 5;
s3  a := 6;
s2  b := a + 1;
```
Data Dependences and Loops

How do we identify dependences in loops?

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

Simple view

- Imagine that all loops are fully unrolled
- Examine data dependences as before

<table>
<thead>
<tr>
<th>i</th>
<th>A(i) = A(i-1)+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A(0)+1</td>
</tr>
<tr>
<td>2</td>
<td>A(1)+1</td>
</tr>
<tr>
<td>3</td>
<td>A(2)+1</td>
</tr>
<tr>
<td>4</td>
<td>A(3)+1</td>
</tr>
<tr>
<td>5</td>
<td>A(4)+1</td>
</tr>
</tbody>
</table>

Problems?

- Impractical
- Lose loop structure

Dependence Analysis for Loops

Big picture

- To improve data locality and parallelism we often focus on loops
- To transform loops, we must understand data dependences in loops
- Since we can’t represent all iterations of a loop, we need some abstractions
- The basic question: does a transformation preserve all dependences?

Today

- Basic abstractions and machinery

Next class

- Its application
Recall Data Dependence Terminology

We say statement $s_2$ depends on $s_1$

- **True (flow) dependence**: $s_1$ writes memory that $s_2$ later reads
- **Anti-dependence**: $s_1$ reads memory that $s_2$ later writes
- **Output dependences**: $s_1$ writes memory that $s_2$ later writes
- **Input dependences**: $s_1$ reads memory that $s_2$ later reads

**Notation:** $s_1 \triangleright s_2$

- $s_1$ is called the **source** of the dependence
- $s_2$ is called the **sink or target**
- $s_1$ must be executed before $s_2$

Dependences and Loops

**Loop-independent dependences**

```latex
do \ i = 1,100
\begin{align*}
A(i) &= B(i)+1 \\
C(i) &= A(i)\cdot2
\end{align*}
enddo
```

**Dependences within the same loop iteration**

**Loop-carried dependences**

```latex
do \ i = 1,100
\begin{align*}
A(i) &= B(i)+1 \\
C(i) &= A(i-1)\cdot2
\end{align*}
enddo
```

**Dependences that cross loop iterations—these depend on the loop structure**
Iteration Spaces

**Idea**
- Explicitly represent the iterations of a loop nest

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

**Iteration Space**
- A set of tuples that represents the iterations of a loop
- Can use an iteration space to visualize the loop’s dependences

Distance Vectors

**Idea**
- Concisely describe dependence relationships among iterations of an iteration space
- For each dimension of an iteration space, the distance is the number of iterations between accesses to the same memory location

**Definition**
- $v = i^T - i^S$

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

**Distance Vector:** $(2,1)$
**Distance Vectors Example**

**Example**

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

**Iteration**

<table>
<thead>
<tr>
<th>i^T (i,j)</th>
<th>Write</th>
<th>Read</th>
</tr>
</thead>
<tbody>
<tr>
<td>i^S (i-2,j-1)</td>
<td>A(j-1,i-2)</td>
<td>A(j-1,i-2)</td>
</tr>
</tbody>
</table>

When does $A(j-1,i-2)$ get written?

Which iteration comes first?

$(i-2,j-1)$

$v = i^T : i^S = (i,j) - (i-2,j-1) = (2,1)$

---

**Distance Vectors and Loop Transformations**

**Idea**

- Any transformation we perform on the loop must respect the dependences

**Example**

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

Can we permute the $i$ and $j$ loops?
Distance Vectors and Loop Transformations (cont)

Idea
- Any transformation we perform on the loop must respect the dependences

Example

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

Can we permute the i and j loops?
- Yes

Example 2

Code
```
  do i = 1, 5
    do j = 2, 6
      A(j, i) = A(j-1, i+1) + 1
    enddo
  enddo
```

Iteration | Write | Read
--- | --- | ---
i× (i, j) | A(j, i) | A(j-1, i+1)
i× (i+1, j-1) | A(j-1, i+1) | . . . WAR dependence

When does A(j-1, i+1) get written?

Which iteration comes first?

\[(i, j) \quad v = i^T \cdot i^S = (i+1, j-1) - (i, j) = (1, -1)\]
Example 2 (cont)

Sample code

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

Kind of dependence: Anti

Distance vector: (1, -1)

Exercise: Consider Loop Permutation

Sample code

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

Kind of dependence: Flow

Distance vector: (1, -1)
Direction Vector

Definition
- A direction vector serves the same purpose as a distance vector when less precision is required or available.
- Element $i$ of a direction vector is $<$, $>$, or $=$ based on whether the source of the dependence precedes, follows, or is in the same iteration as the target in loop $i$.

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

Direction vector: $(<, <)$
Distance vector: $(1, 1)$

Distance Vectors: Legality

Definition
- A dependence vector, $v$, is lexicographically nonnegative when the leftmost non-zero entry in $v$ is positive or when all elements of $v$ are zero.
  Yes: $(0, 0, 0), (0, 1), (0, 2, -2)$
  No: $(-1), (0, -2), (0, -1, 1)$
- A dependence vector is legal when it is lexicographically nonnegative (assuming that indices increase as we iterate).

Why are lexicographically negative distance vectors illegal?

What are legal direction vectors?