Advanced Restructuring
Compilers

Advanced Topics Spring 2008
Prof. Robert van Engelen
Overview

- Data dependence classification
- Data dependences in loops
- Dependence direction and distance
  - Loop-independent dependences
  - K-level loop-carried dependences
- Testing dependence
  - Dependence equations
  - Dependence solvers
- Two example loop transformations
  - Fission
  - Interchange
Data Dependence

Definition

There is a *data dependence* from statement $S_1$ to statement $S_2$ iff

1) both statements $S_1$ and $S_2$ access the same memory location and at least one of them stores into it, and

2) there is a feasible run-time execution path from statement $S_1$ to $S_2$
Data Dependence Classification

Definition

A dependence relation $\delta$ is

- A true dependence (or flow dependence), denoted $S_1 \delta S_2$
- An anti dependence, denoted $S_1 \delta^{-1} S_2$
- An output dependence, denoted $S_1 \delta^o S_2$
Compiling for Parallelism

- **Theorem**

Any instruction statement reordering transformation that preserves every dependence in a program preserves the meaning of that program.

- Bernstein’s conditions for loop parallelism:
  - Iteration $I_1$ does not write into a location that is read by iteration $I_2$
  - Iteration $I_2$ does not write into a location that is read by iteration $I_1$
  - Iteration $I_1$ does not write into a location that is written into by iteration $I_2$
Dependence in Loops

The *statement instances* $S_1[i]$ for iterations $i = 1, \ldots, N$ represent the loop execution.

We have the following flow dependences:

- $S_1[1] \delta S_1[2]
- S_1[2] \delta S_1[3]
- \ldots
- S_1[N-1] \delta S_1[N]

**Statement instances with flow dependences**

```plaintext
DO I = 1, N
S_1 A(I+1) = A(I) + B(I)
ENDDO
```
Representing Dependences with Data Dependence Graphs

- It is generally infeasible to represent all data dependences that arise in a program.
- Usually only static data dependences are recorded.
  - $S_1 \delta(=) S_2$ means $S_1[i] \delta S_2[i]$ for all $i = 1, \ldots, 10000$
  - $S_2 \delta(<) S_2$ means $S_1[i] \delta S_2[j]$ for all $i,j = 1, \ldots, 10000$ with $i < j$
- A data dependence graph compactly represent data dependences in a loop nest.

```plaintext
DO I = 1, 10000
  S_1 A(I) = B(I) * 5
  S_2 C(I+1) = C(I) + A(I)
ENDDO
```

Static data dependences for accesses to $A$ and $C$:
- $S_1 \delta(=) S_2$ and $S_2 \delta(<) S_2$
Iteration Vector

Definition

Given a nest of $n$ loops, the iteration vector $i$ of a particular iteration of the innermost loop is a vector of integers

$$i = (i_1, i_2, \ldots, i_n)$$

where $i_k$ represents the iteration number for the loop at nesting level $k$

The set of all possible iteration vectors spans an iteration space over loop statement instances $S_j[i]$
Iteration Vector Example

The iteration space of the statement at $S_1$ is the set of iteration vectors \{(1,1), (2,1), (2,2), (3,1), (3,2), (3,3)\}

Example: at iteration $i = (2,1)$ statement instance $S_1[i]$ assigns the value of $A(1,2)$ to $A(2,1)$
Iteration Vector Ordering

- The iteration vectors are naturally ordered according to a lexicographical order
  - For example, iteration (1,2) precedes (2,1) and (2,2)

- Definition

Iteration \( i \) precedes iteration \( j \), denoted \( i < j \), iff

1) \( i[1:n-1] < j[1:n-1] \), or

2) \( i[1:n-1] = j[1:n-1] \) and \( i_n < j_n \)
Cross-Iteration Dependence

Definition

There exist a dependence from $S_1$ to $S_2$ in a loop nest iff there exist two iteration vectors $i$ and $j$ such that

1) $i < j$ and there is a path from $S_1$ to $S_2$

2) $S_1$ accesses memory location $M$ on iteration $i$ and $S_2$ accesses memory location $M$ on iteration $j$

3) one of these accesses is a write
Dependence Example

- Show that the loop has no cross-iteration dependence

- Answer: there are no iteration vectors $i$ and $j$ in \{(1,1), (2,1), (2,2), (3,1), (3,2), (3,3)\} such that $i < j$ and
  - $S_1$ in $i$ writes to the same element of $A$ that is read at $S_1$ in iteration $j$
  - or $S_1$ in iteration $i$ reads an element $A$ that is written at $S_1$ in iteration $j$
Dependence Distance and Direction Vectors

Definition

Given a dependence from $S_1$ on iteration $i$ to $S_2$ on iteration $j$, the dependence distance vector $d(i, j)$ is defined as $d(i, j) = j - i$

Given a dependence from $S_1$ on iteration $i$ and $S_2$ on iteration $j$, the dependence direction vector $D(i, j)$ is defined for the $k^{th}$ component as

$$D(i, j)_k = \begin{cases} 
"<" & \text{if } d(i, j)_k > 0 \\
=" & \text{if } d(i, j)_k = 0 \\
">" & \text{if } d(i, j)_k < 0 
\end{cases}$$
Example 1

DO I = 1, 3
  DO J = 1, I
    A(I+1,J) = A(I,J)
  ENDDO
ENDDO

Flow dependence between $S_1$ and itself on:
i = (1,1) and $j = (2,1)$: $d(i, j) = (1, 0)$, $D(i, j) = (<, =)$
i = (2,1) and $j = (3,1)$: $d(i, j) = (1, 0)$, $D(i, j) = (<, =)$
i = (2,2) and $j = (3,2)$: $d(i, j) = (1, 0)$, $D(i, j) = (<, =)$
Example 2

Distance vector is (1,1)

\[ S_1 \delta_{(<, <)} S_1 \]

\[ \text{DO } I = 1, 4 \]
\[ \text{DO } J = 1, 4 \]
\[ A(I,J+1) = A(I-1,J) \]
\[ \text{ENDDO} \]
\[ \text{ENDDO} \]
Example 2

Distance vector is \((1, -1)\)

\[ S_1 \delta_{(<,>)} S_1 \]

\[
\begin{array}{c}
\text{DO } I = 1, 4 \\
\quad \text{DO } J = 1, 5-I \\
\quad A(I+1,J+I-1) = A(I,J+I-1) \\
\quad \text{ENDDO} \\
\text{ENDDO}
\end{array}
\]
Example 3

Distance vectors are $(0,1)$ and $(1,0)$

\[ S_1 \delta_{(=,<)} S_1 \]
\[ S_2 \delta_{(<,=)} S_1 \]
\[ S_2 \delta_{(<,=)} S_2 \]
Loop-Independent Dependences

- Definition

Statement $S_1$ has a loop-independent dependence on $S_2$ iff there exist two iteration vectors $i$ and $j$ such that

1) $S_1$ refers to memory location $M$ on iteration $i$, $S_2$ refers to $M$ on iteration $j$, and $i = j$

2) there is a control flow path from $S_1$ to $S_2$ within the iteration
K-Level Loop-Carried Dependences

Definition
Statement $S_1$ has a loop-carried dependence on $S_2$ iff
1) there exist two iteration vectors $i$ and $j$ such that $S_1$ refers to memory location $M$ on iteration $i$ and $S_2$ refers to $M$ on iteration $j$
2) $d(i,j) > 0$, that is, $D(i,j)$ contains a “<” as its leftmost non-“=” component

A loop-carried dependence from $S_1$ to $S_2$ is
1) lexically forward if $S_2$ appears after $S_1$ in the loop body
2) lexically backward if $S_2$ appears before $S_1$ (or if $S_1=S_2$)

The level of a loop-carried dependence is the index of the leftmost non-“=” of $D(i,j)$ for the dependence
Example

```
DO I = 1, 10
  DO J = 1, 10
    DO K = 1, 10
      A(I,J,K+1) = A(I,J,K)
    ENDDO
  ENDDO
ENDDO
```

- All loop-carried dependences are of level 3, \( D(i,j) = (=, =, <) \)

- Level-\( k \) dependences are sometimes denoted by \( S_x \delta_k S_y \)

\[ S_1 \delta_{(=,=,<)} S_1 \quad \text{Alternative notation for a level-3 dependence} \]
Combining Direction Vectors

- A loop nest can have multiple different directions at the same loop level \( k \)

- We abbreviate a level-\( k \) direction vector component with "\( * \)" to denote any direction

\[
\begin{align*}
&\text{DO } I = 1, 9 \\
&S_1 \quad A(I) = \ldots \\
&S_2 \quad \ldots = A(10-I) \\
&\text{ENDDO}
\end{align*}
\]

\[
\begin{align*}
&\text{DO } I = 1, 10 \\
&S_1 \quad S = S + A(I) \\
&\text{ENDDO}
\end{align*}
\]

\[
\begin{align*}
&\text{DO } I = 1, 10 \\
&\quad \text{DO } J = 1, 10 \\
&S_1 \quad A(J) = A(J)+1 \\
&\text{ENDDO} \\
&\text{ENDDO}
\end{align*}
\]
How to Determine Dependencies?

- A system of *Diophantine dependence equations* is set up and solved to test direction of dependence (flow/anti)
  - Proving there is (no) solution to the equations means there is (no) dependence
  - Most dependence solvers require affine array index expressions of the form
    \[ a_1 i_1 + a_2 i_2 + \ldots + a_n i_n + a_0 \]
    where \( i_j \) are loop index variables and \( a_k \) are integer

- Non-linear subscripts are problematic
  - Symbolic terms
  - Function values
  - Indirect indexing
  - etc
Dependence Equation: Testing Flow Dependence

To determine flow dependence:
prove the dependence equation
\[ f(\alpha) = g(\beta) \]
has a solution for iteration vectors \( \alpha \) and \( \beta \), such that \( \alpha < \beta \)

\[
\begin{align*}
\text{DO } & I = 1, N \\
S_1 & \quad A(f(I)) = A(g(I)) \\
\end{align*}
\]

\[ \text{write } \rightarrow \text{read} \]

\[
\begin{align*}
\text{DO } & I = 1, N \\
S_1 & \quad A(I+1) = A(I) \\
\end{align*}
\]

\[ \alpha + 1 = \beta \text{ has solution } \alpha = \beta - 1 \]

\[
\begin{align*}
\text{DO } & I = 1, N \\
S_1 & \quad A(2*I+1) = A(2*I) \\
\end{align*}
\]

\[ 2\alpha + 1 = 2\beta \text{ has no solution} \]
Dependence Equation: Testing Anti Dependence

To determine anti dependence: prove the dependence equation
\[ f(\alpha) = g(\beta) \]
has a solution for iteration vectors \( \alpha \) and \( \beta \), such that \( \beta < \alpha \)

\[
\begin{align*}
\text{DO I = 1, N} \\
S_1 & \quad A(\mathbf{f}(I)) = A(\mathbf{g}(I)) \\
\text{ENDDO}
\end{align*}
\]

\[ \alpha + 1 = \beta \text{ has no solution} \]

\[
\begin{align*}
\text{DO I = 1, N} \\
S_1 & \quad A(I+1) = A(I) \\
\text{ENDDO}
\end{align*}
\]

\[ 2\alpha = 2\beta + 2 \text{ has solution } \alpha = \beta + 1 \]
Dependence Equation

Examples

To determine flow dependence:
prove there are iteration vectors \( \alpha < \beta \) such that \( f(\alpha) = g(\beta) \)

\[
\begin{align*}
\text{DO I = 1, N} \\
\quad \text{DO J = 1, N} \\
\quad S_1 \quad A(I-1) = A(J) \\
\quad \text{ENDDO} \\
\text{ENDDO}
\end{align*}
\]

\( \alpha = (\alpha_I, \alpha_J) \) and \( \beta = (\beta_I, \beta_J) \)
\( \alpha_I - 1 = \beta_J \) has solution

\[
\begin{align*}
\text{DO I = 1, N} \\
\quad S_1 \quad A(5) = A(I) \\
\quad \text{ENDDO}
\end{align*}
\]

\( 5 = \beta \) is solution if \( 5 < N \)

\[
\begin{align*}
\text{DO I = 1, N} \\
\quad S_1 \quad A(5) = A(6) \\
\quad \text{ENDDO}
\end{align*}
\]

\( 5 = 6 \) has no solution
Dependence System: Loop Normalization

Consider *normalized loop iteration spaces*
- Set lower bound to zero by adjusting limits and occurrence of loop variable

A *dependence system* consists of a dependence equation along with a set of constraints:
- Solution must lie within loop bounds
- Solution must be integer
- Need dependence distance or direction vector (flow/anti)

```plaintext
DO I = 2, 100
  DO J = 1, I-1
  S1 A(I,J) = A(J,I)
  ENDDO
ENDDO

DO I = 0, 98
  DO J = 0, I+2-2
  S1 A(I+2,J+1) = A(J+1,I+2)
  ENDDO
ENDDO
```
Dependence System: Formulating Flow Dependence

\[ \alpha < \beta \text{ such that } f(\alpha) = g(\beta) \]

where \( \alpha = (\alpha_I, \alpha_J) \) and \( \beta = (\beta_I, \beta_J) \)

- Consider normalized loop iteration spaces
  - Set lower bound to zero by adjusting limits and occurrence of loop variable

A dependence system consists of a dependence equation along with a set of constraints:

- Solution must lie within loop bounds
- Solution must be integer
- Need dependence distance or direction vector (flow/anti)

\[
\begin{align*}
\alpha_I + 2 &= \beta_J + 1 \\
\alpha_J + 1 &= \beta_I + 2 \\
0 &\leq \alpha_I, \beta_I \leq 98 \\
0 &\leq \alpha_J, \beta_J \leq \alpha_I \\
0 &\leq \alpha_J, \beta_J \leq \beta_I \\
\alpha_I &< \beta_I \\
\end{align*}
\]

\( S_I \; A(I+2, J+1) = A(J+1, I+2) \)

ENDDO

ENDDO

\[ \alpha_I < \beta_I \]

Constraint for \((<, \ast)\) dep. direction
Dependence System: Matrix Notation

Consider normalized loop iteration spaces

- Set lower bound to zero by adjusting limits and occurrence of loop variable

A dependence system consists of a dependence equation along with a set of constraints:

- Solution must lie within loop bounds
- Solution must be integer
- Need dependence distance or direction vector (flow/anti)
Fourier-Motzkin Projection

\[
\begin{pmatrix}
0 & 1 \\
1 & 1 \\
1 & -1 \\
-2 & -1 \\
\end{pmatrix}
\begin{pmatrix}
x_1 \\
x_2 \\
\end{pmatrix}
\leq
\begin{pmatrix}
6 \\
9 \\
5 \\
-7 \\
\end{pmatrix}
\]

System of linear inequalities
\[Ax \leq b\]

Projections on \(x_1\) and \(x_2\)

\[x_2 \leq 6\]
\[2x_1 - x_2 \leq -7\]
\[x_1 + x_2 \leq 9\]
\[-2x_1 - x_2 \leq -7\]
Fourier-Motzkin Variable Elimination (FMVE)

\[
\begin{pmatrix}
0 & 1 \\
1 & 1 \\
1 & -1 \\
-2 & -1
\end{pmatrix}
\begin{pmatrix}
x_1 \\
x_2
\end{pmatrix}
\leq
\begin{pmatrix}
6 \\
9 \\
5 \\
-7
\end{pmatrix}
\]

Select \( x_2 \): \( L = \{3, 4\}, \ U = \{1, 2\} \)

new system:

\[
\begin{pmatrix}
1 & 0 \\
2 & 0 \\
-2 & 0 \\
-1 & 0
\end{pmatrix}
\begin{pmatrix}
x_1 \\
x_2
\end{pmatrix}
\leq
\begin{pmatrix}
11 \\
14 \\
-1 \\
2
\end{pmatrix}
\]

\[
\text{max}(1/2, -2) \leq x_1 \leq \text{min}(11, 7)
\]

- FMVE procedure:
  1. Select an unknown \( x_j \)
  2. \( L = \{i \mid a_{ij} < 0\} \)
  3. \( U = \{i \mid a_{ij} > 0\} \)
  4. if \( L = \emptyset \) or \( U = \emptyset \) then \( x_j \) is unconstrained (delete it)
  5. for \( i \in L \cup U \)
     \[ A_{[i]} := \frac{A_{[i]}}{|a_{ij}|} \]
     \[ b_i := \frac{b_i}{|a_{ij}|} \]
  6. for \( i \in L \)
     for \( k \in U \)
       add new inequality
       \[ A_{[i]} + A_{[k]} \leq b_i + b_k \]
  7. Delete old rows \( L \) and \( U \)
GCD Test

- Requires that $f(\alpha)$ and $g(\beta)$ are affine:
  
  $f(\alpha) = a_0 + a_1\alpha + \ldots + a_n\alpha_n$
  
  $g(\beta) = b_0 + b_1\beta + \ldots + b_n\beta_n$

- Reordering gives linear Diophantine equation:
  
  $a_1\alpha_1 - b_1\beta_1 + \ldots + a_n\alpha_n - b_n\beta_n = b_0 - a_0$
  
  which has a solution if $\gcd(a_1, \ldots, a_n, b_1, \ldots, b_n)$ divides $b_0 - a_0$

Which of these loops have dependences?
**Loop Fission**

- Compute the *acyclic condensation* of the dependence graph to find a legal order of the loops.

### Code Example

```
S_1 DO I = 1, 10
S_2 A(I) = A(I) + B(I-1)
S_3 B(I) = C(I-1)*X + Z
S_4 C(I) = 1/B(I)
S_5 D(I) = sqrt(C(I))
S_6 ENDDO
```

### Dependence Graph

**Acyclic condensation**

```
S_1 DO I = 1, 10
S_3 B(I) = C(I-1)*X + Z
S_4 C(I) = 1/B(I)
S_x ENDDO
S_y DO I = 1, 10
S_2 A(I) = A(I) + B(I-1)
S_z ENDDO
S_u DO I = 1, 10
S_v D(I) = sqrt(C(I))
S_v ENDDO
```
Loop Interchange

- Compute the direction matrix and find which columns can be permuted without violating dependence relations in original loop nest.
Further Reading

- [High] Chapters 5, 7-9 (parts)