CS 526

Advanced
Compiler
Construction

http://misailo.cs.Illinois.edu/courses/cs526

DEPENDENCE ANALYSIS

The slides adapted from Vikram Adve and David Padua

Kinds of Data Dependence

Direct Dependence

X =…
   = X + …

$$S_1 \longrightarrow S_2$$

Antidependence

… = X
X = …

$$S_1 \mathrel{+\!\!\!\!\longrightarrow} S_2$$

Output Dependence

X = …
X = …

$$S_1 \mathrel{\circ\!\!\!\!\longrightarrow} S_2$$

Dependence in Loop Nests

```
do i1 = 1 to n1
   do i2 = 1 to n2
      . . .
        do ik = 1 to nk
          statements
        enddo
      . . .
   enddo
enddo
```

Iteration space

The *iteration space* of the loop nest is a set of points in a k-dimensional integer space (i.e., a polyhedron):

$$
\begin{aligned}
L = \{[i_1, \ldots, i_n] :& \\
& 1 \leq i_1 \leq n_1 \wedge \ldots \wedge \\
& 1 \leq i_k \leq n_k\}
\end{aligned}
$$

Each element $[i_1, \ldots, i_n]$ is an **iteration vector**

Dependence in Loop Nests

Lexicographic Order: for iteration vectors $[i_1, \ldots, i_n]$ and $[j_1, \ldots, j_n]$:

$[i_1, \ldots, i_n] < [j_1, \ldots, j_n]$ iff there is a subscript k, such that $i_1 = j_1, \ldots i_{k-1} = j_{k-1}$ but $i_k < j_k$

If $\mathrm{I} = [i_1, \ldots, i_n] < [j_1, \ldots, j_n] = J$ we say that the iteration I precedes the iteration *J*

Dependence in Loop Nests

```
I=(i1, i2, …, ik)
```

```
do i1 = 1 to n1
   do i2 = 1 to n2
     . . .
      do ik = 1 to nk
         X(f1(I), …, fk(I)) = ...
         ... = X(g1(I), …, gk(I))
      enddo
     . . .
   enddo
enddo
```

Direct (Flow) Dependence in Loops

We say that S1→S2 iff there exist I, J ∈ L and I ≤ J where

1. There is a feasible path from instance I of statement S1 to instance J of statement S2,

```
X(f1(I), …, fk(I)) = ...
   ...
... = X(g1(J), …, gk(J))
```

2. **$f_s(I) = g_s(J)$**, $\forall\ 1 \le s \le k$

   The statement S1 in iteration I writes and S2 in iteration J reads from the same memory location M

Antidependence in Loops

We say that $S1 \nrightarrow S2$ iff there exist $I, J \in L$ and $I \leq J$ where

1. There is a feasible path from instance I of statement S1 to instance J of statement S2,

```
... = X(f1(I), …, fk(I))
        ...
X(g1(J), …, gk(J)) = ...
```

2. $\mathbf{f_s(I) = g_s(J)}$, $\forall\ 1 \leq s \leq k$

   The statement S1 in iteration I reads and S2 in iteration J writes to the same memory location M

Output Dependence in Loops

We say that S1 $\leftrightsquigarrow$ S2 iff there exist I, J ∈ L and I ≤ J where

1. There is a feasible path from instance I of statement S1 to instance J of statement S2,

```
X(f1(I), …, fk(I)) = ...
         ...
X(g1(J), …, gk(J)) = ...
```

2. **f_s(J) = g_s(I),** $\forall\ 1 \leq s \leq k$

   The statement S1 in iteration I and S2 in iteration J both write to the same memory location M

Dependence Distance

Dependence Distance: If there is a dependence from statement S1 on iteration $\vec{i}$ and statement S2 on iteration $\vec{j}$ then the corresponding dependence distance vector is

$$d_{\vec{\boldsymbol{i}},\vec{\boldsymbol{j}}} = [\overrightarrow{j_1} - \overrightarrow{i_1}, \ldots \overrightarrow{j_k} - \overrightarrow{i_k}]$$

Note: Computing distance vectors is harder than testing dependence

Dependence Distance

Direction Vector: For a distance vector of the form $d_{\vec{i},\vec{j}} = [\overrightarrow{j_1} - \overrightarrow{i_1}, \ldots \overrightarrow{j_k} - \overrightarrow{i_k}]$ the corresponding direction vector is $\delta_{\vec{i},\vec{j}} = [\delta_1, \ldots \delta_k]$, where

$$\delta_i = \begin{cases} -, & if\ \overrightarrow{j_1} - \overrightarrow{i_1} < 0 \\ +, & if\ \overrightarrow{j_1} - \overrightarrow{i_1} > 0 \\ =, & if\ \overrightarrow{j_1} - \overrightarrow{i_1} = 0 \\ *, & if\ \text{sign} <,>,= \end{cases}$$

Legal Direction Vectors

Note: Consider two iteration vectors **I** and **J** and the direction vector between them, δ (**I**, **J**):

- **I** < **J** iff the leftmost non-'=' entry in δ(**I**, **J**) is '+'.
- We use the property of lexicographical ordering

In the direction vector, for any dependence, the leftmost non-'=' entry must be '+' (if any non-'=' entry is present).

Equivalently: **the distance vector d ≥ 0**.

Loop Carried Dependence

Statement S2 has a loop carried dependence on statement S1 iff S1 references location M on iteration **I**, S2 references M on iteration **J** and **d(I,J)>0** .

```
do i = 1 to N
    A(i+1) = B(i)
    B(i+1) = A(i)
enddo
```

Level of loop-carried dependence is the leftmost non-"=" sign in the direction vector

Loop Independet Dependence

Statement S2 has a loop carried dependence on statement S1 iff S1 references location M on iteration **I**, S2 references M on iteration **J** and **d(I,J)=0** .

```
do i = 1 to N
    A(i+1) = B(i)
    B(i+1) = A(i+1)
enddo
```

Determines the order in which the code is executed within the nest of loops (compare to loop carried!)

The level of a loop-independent dependence is ∞.

Dependence in Loops

```
do i = 1 to N
    a(i+1) = a(i) + B
enddo
```

Transformations and Direction Vectors

Theorem: Consider a transformation T on a loop nest that does not reorder statements within a loop body.

Such a transformation is legal if, after applying the corresponding transformation to the direction vectors of each dependece, none of them have a leftmost non-'=' entry that is '-' (or, equivalently $d < 0$).

Equivalently: none of the dependences have had the order of their source and sink reversed.

Dependence Testing

Dependence testing requires finding a solution to $\{ f_s(\mathbf{I}) = g_s(\mathbf{J}), \forall 1 \leq s \leq n\}$ under the inequality constraints $\mathbf{I}, \mathbf{J} \in \mathrm{L}$

Complexity: undecidable in general

- Indirection arrays (e.g. X[Y[i]])
- Indirection arrays may only be known at runtime, without a specific application knowledge
- General alias analysis
- Non-linear subscript expressions

Dependence Testing

Assume linear subscript expressions, e.g.,

$$a_0 + a_1 i_1 + \ldots a_n i_n,$$

where $i_1 \ldots i_n$ are loop index variables.

Instance of integer programming

$\Rightarrow$ NP-complete in general

Simplifications

Two major simplifications in practice:

- Subscript expressions are usually simple:
  most often i_l or $a_l i_l + a_0$

- Be **conservative**:
  Check if a dependence **may exist**.

Simplifications

ZIV, SIV, MIV A subscript expression containing **z**ero, **s**ingle, or **m**ultiple index variable respectively:

E.g., A[n], A[2 * i1 + n], A[2 * i1 + 3 * i2 + 5]

Separable Subscripts : A subscript position is said to be **separable** if the index variables used in that subscript position are not used in any other subscript position.

E.g., A[i+1, j, k] and A[i, j, k]

Coupled Subscripts : Two subscript positions are said to be coupled if the same index variable is used in both positions.

E.g., A[i+1, i, k] and A[i, j+i, k]

GCD Test

Simplifications

1. ignore loop bounds!
2. only test if a solution is *possible* (GCD property)
3. test each subscript position separately

GCD Property for Single Variable

Let $f(i) = a_1 i + a_0$ and $g(i) = b_1 i + b_0$

$f(i_1) = g(i_2) \Rightarrow a_1 i_1 + a_0 = b_1 i_2 + b_0$.

GCD Property: If there is a solution to the previous equation, then $g = \gcd(a_1, b_1)$ divides $a_0 - b_0$.

Proof: Let $a_1 = n_1 g$, $b_1 = m_1 g$. Then $g \times (n_1 i_1 - m_1 i_2) = a_0 - b_0$, and the term in parenthesis must be an integer.

GCD Test for Multiple Indices

Let $f(\mathbf{I}) = a_k i_k + \ldots + a_0$ and
$g(\mathbf{I}) = b_k i_k + \ldots + b_0$.

GCD Property: If there is a solution to the equation

$$a_k i_{k1} + \ldots + a_0 = b_k i_{k2} + \ldots + b_0, \text{ then}$$

$$g = \gcd(a_1, \ldots, a_k, b_1, \ldots, b_k) \text{ divides } (a_0 - b_0).$$

More tests: E.g., Banerjee test, Lamport test, …

Solving Complicated Indices

E.g. `A[x+2y-1, 2y, z, w+z, v, 1].`

Simplify the problem by identifying common special cases:

1. Separate subscript positions into coupled groups
2. Label each subscript as ZIV, SIV, or MIV
3. For each separable subscript, apply appropriate test (ZIV, SIV, or MIV). Yields direction vectors.
4. For each coupled group, apply a coupled subscript test; e.g., GCD test or Delta test
5. *If no test yields independence, a dependence exists*:
6. Concatenate direction vectors from different groups

Exact Solutions for SIV

A pair of subscripts with index variable i_j are **Strong SIV** if the subscript expressions are the form $a\ i_j + b_1$ and $a\ i_j + b_2$

Dependence exists *iff* either of these hold:

1. $a = 0$ and $b_1 = b_2$, or
2. $|\ d_j\ | \leq n_j - 1$, where $d_j = (b1 - b2)/a$

Assumes: n_j , a, b_1, b_2 *are known*

Exact Solutions for SIV

The set of subscripts with index variable i_j are **Weak SIV** if the subscripts are of the form $a_1 i_j + b_1$ and $a_2 i_j + b_2$

Each such subscript position j gives an equation of the form:

$$a_1 y = a_2 x + b_2 - b_1$$

Approach for each index variable i_j :

1. Solve up to r simultaneous equations in 2 unknowns.
2. Check if solutions satisfy 2 inequalities

Exact Solutions for Weak SIV

Special case: one of a_1 or a_2 is zero: **Weak-Zero SIV**
(solution is similar to strong SIV)

General problem: Find if $a_1 i_1 + a_0 = b_1 i_2 + b_0$

(Lemma) An extended GCD property:
For any pair of values (x, y), the Euclidian GCD algorithm can also compute a triplet (g, n_x, n_y) such that

$$g = n_x x + n_y y = \gcd(x, y)$$

Exact Solutions for Weak SIV

Theorem. Let (g, n_a, n_b) be such a triplet for pair $(a_1, -b_1)$.
Let x_k and y_k be given by:

$$
\begin{aligned}
x_k &= n_a \left(\frac{b_0 - a_0}{g}\right) + k\frac{b_1}{g} \\
y_k &= n_b \left(\frac{b_0 - a_0}{g}\right) + k\frac{a_1}{g}
\end{aligned}
$$

Then (x_k, y_k) is a solution of $a_1 i_1 + a_0 = b_1 i_2 + b_0$ for an integral value of k.
Furthermore, for any solution (x,y) there is a k such that $x = x_k$ and $y = y_k$

Solution strategy:

1. Compute x_0, y_0 using the above equations
2. Then find all values of k for which $x_0 + k\, b_1/g$ falls within loop bounds, and similarly for y_k.
3. For dependence to exist, the solution (x_k, y_k) must be within the region bounded by loop bounds