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

\[ X = \ldots = X + \ldots \]

Antidependence

\[ \ldots = X \]

\[ X = \ldots \]

Output Dependence

\[ X = \ldots \]

\[ X = \ldots \]
Dependence in Loop Nests

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

$$L = \{[i_1, \ldots, i_n] : 1 \leq i_1 \leq n_1 \land \ldots \land 1 \leq i_k \leq n_k\}$$

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

do i1 = 1 to n1
  do i2 = 1 to n2
    . . .
    do ik = 1 to nk
      statements
    enddo
  enddo
.enddo
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 \(I = [i_1, \ldots, i_n] < [j_1, \ldots, j_n] = J\) we say that the iteration \(I\) preceds the iteration \(J\)
 Dependence in Loop Nests

\[
\begin{align*}
\text{do } i_1 &= 1 \text{ to } n_1 \\
\text{do } i_2 &= 1 \text{ to } n_2 \\
\cdots \cdots \\
\text{do } i_k &= 1 \text{ to } n_k \\
X(f_1(I), \ldots, f_k(I)) &= \ldots \\
\cdots &= X(g_1(I), \ldots, g_k(I)) \\
\text{enddo} \nonumber \\
\cdots \cdots \\
\text{enddo} \nonumber \\
\text{enddo} \nonumber
\end{align*}
\]
Direct (Flow) Dependence in Loops

We say that $S_1 \rightarrow S_2$ iff there exist $I, J \in L$ and $I \leq J$ where

1. There is a feasible path from instance $I$ of statement $S_1$ to instance $J$ of statement $S_2$,

   $$X(f_1(I), \ldots, f_k(I)) = \ldots$$
   $$\ldots = X(g_1(J), \ldots, g_k(J))$$

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

   The statement $S_1$ in iteration $I$ writes and $S_2$ in iteration $J$ reads from the same memory location $M$
Antidependence in Loops

We say that $S_1 \not \leftrightarrow S_2$ iff there exist $I, J \in L$ and $I \leq J$ where

1. There is a feasible path from instance $I$ of statement $S_1$ to instance $J$ of statement $S_2$, 

$$
\ldots = X(f_1(I), \ldots, f_k(I)) \\
\ldots = X(g_1(J), \ldots, g_k(J)) = \ldots
$$

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

The statement $S_1$ in iteration $I$ reads and $S_2$ in iteration $J$ writes to the same memory location $M$
Output Dependence in Loops

We say that $S_1 \leftrightarrow S_2$ iff there exist $I, J \in L$ and $I \leq J$ where

1. There is a feasible path from instance $I$ of statement $S_1$ to instance $J$ of statement $S_2$,
   
   $$X(f_1(I), \ldots, f_k(I)) = \ldots$$
   $$\ldots$$
   $$X(g_1(J), \ldots, g_k(J)) = \ldots$$

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

   The statement $S_1$ in iteration $I$ and $S_2$ 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

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

*Note: Computing distance vectors is harder than testing dependence*
**Dependence Distance**

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

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

Note: Consider two iteration vectors $I$ and $J$ and the direction vector between them, $\delta (I, J)$:

- $I < J$ iff the leftmost non-’=’ entry in $\delta(I, J)$ 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 $d \geq 0$. 
Loop Carried Dependence

Statement $S_2$ has a loop carried dependence on statement $S_1$ iff $S_1$ references location $M$ on iteration $I$, $S_2$ references $M$ on iteration $J$ and $d(I,J) > 0$.

$$
do \ i = 1 \ to \ N
\begin{align*}
A(i+1) &= B(i) \\
B(i+1) &= A(i)
\end{align*}
enddo$$

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 $d(I,J)=0$.

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

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 $\infty$. 
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 dependence, 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 \( \{ f_s(l) = g_s(j), \forall 1 \leq s \leq n \} \) under the inequality constraints \( l, j \in L \).

**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
\[ \Rightarrow \text{NP-complete in general} \]
Simplifications

Two major simplifications in practice:

• Subscript expressions are usually simple:
  most often $i_1$ or $a_1 i_1 + a_0$

• Be **conservative**: Check if a dependence **may exist**.
Simplifications

**ZIV, SIV, MIV** A subscript expression containing zero, single, or multiple index variable respectively:
E.g., $A[n], A[2 \times i_1 + n], A[2 \times i_1 + 3 \times i_2 + 5]$

**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 $f(i) = a_1i + a_0$ and $g(i) = b_1i + b_0$
$f(i_1) = g(i_2) \Rightarrow a_1i_1 + a_0 = b_1i_2 + b_0$.

**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 $a_1 = n_1g$, $b_1 = m_1g$. Then $g \times (n_1i_1 - m_1i_2) = a_0 - b_0$, and the term in parenthesis must be an integer.
GCD Test for Multiple Indices

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

**GCD Property**: If there is a solution to the equation $a_k i_k + \ldots + a_0 = b_k i_k + \ldots + b_0$, then

$$g = \gcd(a_1, \ldots, a_k, b_1, \ldots, b_k)$$

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 $d_j = (b_1 - b_2)/a$

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 $a_1i_1 + a_0 = b_1i_2 + b_0$

**(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

\[ g = n_x x + n_y y = \text{gcd}(x, y) \]
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:

\[
x_k = n_a \left( \frac{b_0 - a_0}{g} \right) + k \frac{b_1}{g}
\]

\[
y_k = n_b \left( \frac{b_0 - a_0}{g} \right) + k \frac{a_1}{g}
\]

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 \(x_0 + k \frac{b_1}{g}\) 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.