**The General Problem**

DO \( i_1 = L_1, U_1 \)
DO \( i_2 = L_2, U_2 \)
\[ \ldots \]
DO \( i_n = L_n, U_n \)

S\(_1\) \[ A(f_1(i_1, \ldots, i_n), \ldots, f_m(i_1, \ldots, i_n)) = \ldots \]
S\(_2\) \[ \ldots = A(g_1(i_1, \ldots, i_n), \ldots, g_m(i_1, \ldots, i_n)) \]

ENDDO
\[ \ldots \]
ENDDO
ENDDO

A dependence exists from S\(_1\) to S\(_2\) if:

- There exist \( \alpha \) and \( \beta \) such that:
  - \( \alpha < \beta \) (control flow requirement)
  - \( f_i(\alpha) = g_i(\beta) \) for all \( i, 1 \leq i \leq m \) (common access req)

**ZIV Test**

DO \( j = 1, 100 \)
S \[ A(e_1) = A(e_2) + B(j) \]
ENDDO

e\(_1\), e\(_2\) are constants or loop invariant symbols
If \( (e_1 - e_2) \neq 0 \) No Dependence exists

**Strong SIV Test**

DO \( i_1 = L_1, U_1 \)
DO \( i_2 = L_2, U_2 \)
\[ \ldots \]
DO \( i_n = L_n, U_n \)

S\(_1\) \[ A(f_1(i_1, \ldots, i_n), \ldots, f_m(i_1, \ldots, i_n)) = \ldots \]
S\(_2\) \[ \ldots = A(g_1(i_1, \ldots, i_n), \ldots, g_m(i_1, \ldots, i_n)) \]

ENDDO
\[ \ldots \]
ENDDO
ENDDO

- Strong SIV test when
  - \( f(\ldots) = a_i c_1 + c_2 \) and \( g(\ldots) = a_i c_2 \)
  - Plug in \( \alpha, \beta \) and solve for dependence:
    - \( \beta - \alpha = (c_1 - c_2)/a \)
  - A dependence exists from S\(_1\) to S\(_2\) if:
    - \( \beta - \alpha \) is an integer
    - \( |\beta - \alpha| \leq U_k - L_k \)
Can extend to symbolic constants

- Determine d symbolically
- If d is a constant, use previous procedure
- Otherwise, calculate U-L symbolically
- Compare U-L and d symbolically (& hope)

E.g.,

\[ \text{for } i=1 \text{ to } N \]

Weak-zero SIV Test

\[
\begin{align*}
\text{DO } i_1 &= L_1, U_1 \\
\text{DO } i_2 &= L_2, U_2 \\
\vdots \\
\text{DO } i_n &= L_n, U_n \\
S_1 &= A(f_1(i_1, \ldots, i_n), \ldots, f_n(i_1, \ldots, i_n)) = \ldots \\
S_2 &= \ldots = A(g_1(i_1, \ldots, i_n), \ldots, g_n(i_1, \ldots, i_n)) \\
\text{ENDDO} \\
\vdots \\
\text{ENDDO} \\
\text{ENDDO}
\end{align*}
\]

- Weak-Zero SIV test when
  \[ f(...) = a_i k + c_1 \text{ and } g(...) = c_2 \]
  - Plug in \( \alpha, \beta \) and solve for dependence:
    \[ \alpha = \frac{(c_2 - c_1)}{a} \]
  - A dependence exists from S1 to S2 if:
    \[ 2 \alpha \text{ is an integer} \]
    \[ L_k \leq \alpha \leq U_k \]

Weak-crossing SIV Test

\[
\begin{align*}
\text{DO } i_1 &= L_1, U_1 \\
\text{DO } i_2 &= L_2, U_2 \\
\vdots \\
\text{DO } i_n &= L_n, U_n \\
S_1 &= A(f_1(i_1, \ldots, i_n), \ldots, f_n(i_1, \ldots, i_n)) = \ldots \\
S_2 &= \ldots = A(g_1(i_1, \ldots, i_n), \ldots, g_n(i_1, \ldots, i_n)) \\
\text{ENDDO} \\
\vdots \\
\text{ENDDO} \\
\text{ENDDO}
\end{align*}
\]

- Weak-Zero SIV test when
  \[ f(...) = a_i k + c_1 \text{ and } g(...) = -a_i k c_2 \]
  - To find crossing point, set \( \alpha = \beta \) and solve:
    \[ \alpha = \frac{(c_2 - c_1)}{2a} \]
  - A dependence exists from S1 to S2 if:
    \[ 2 \alpha \text{ is an integer} \]
    \[ L_k \leq \alpha \leq U_k \]

Non-rectangular spaces

- Triangular iteration space when only one loop bound depends on an outer loop index
- Trapezoidal space when both loop bounds depend on an outer loop index

- Example:
  \[
  \begin{align*}
  \text{for } i=1 \text{ to } N \\
  \text{for } j=L_0+L_1*I \text{ to } U_0+U_1*I \\
  A[j+D] &= \ldots \\
  \ldots &= A[j] \\
  - \text{Is } d \text{ in loop bounds?}
  \end{align*}
  \]
Complex Iteration Spaces

- For example consider this special case of a strong SIV subscript
  
  ```
  DO I = 1, N
      DO J = L_0 + L_1*I, U_0 + U_1*I
         S1 A(J + d) = 
         S2 = A(J) + B
      ENDDO
  ENDDO
  ```

- Strong SIV test gives dependence if

  \[
  |d| \leq U_0 - L_0 + (U_1 - L_1)I
  \]

  \[
  I \geq \frac{|d| - (U_0 - L_0)}{U_1 - L_1}
  \]

- Unless this inequality is violated for all values of I in its iteration range, we must assume a dependence in the loop

Breaking Conditions

- Consider the following example
  
  ```
  DO I = 1, L
      S1 A(I + N) = A(I) + B
  ENDDO
  ```

- If L<=N, then there is no dependence from S_1 to itself

- L<=N is called the Breaking Condition

Using Breaking Conditions

- Using breaking conditions the vectorizer can generate alternative code

  ```
  IF (L<=N) THEN
      A(N+1:N+L) = A(1:L) + B
  ELSE
      DO I = 1, L
         S1 A(I + N) = A(I) + B
      ENDDO
  ENDIF
  ```
Index Set Splitting

DO I = 1, 100
  DO J = 1, I
    S1 A(J+20) = A(J) + B
  ENDDO
ENDDO

For values of 
$$I \leq \frac{U_0 - L_0}{U_i - L_i} = \frac{20 - (-1)}{1} = 21$$

there is no dependence

This condition can be used to partially vectorize S1 by Index set splitting as shown

DO I = 1, 20
  DO J = 1, I
    S1a A(J+20) = A(J) + B
  ENDDO
ENDDO

DO I = 21, 100
  DO J = 1, Ix
    S1b A(J+20) = A(J) + B
  ENDDO
ENDDO

Now the inner loop for the first nest can be vectorized.

How are we doing so far?

- Empirical study from Goff, Kennedy, & Tseng
  - Look at how often independence and exact dependence information is found in 4 suites of fortran programs
  - Compare ZIV, SIV (strong, weak-0, weak-crossing, exact), MIV, Delta
  - Check usefulness of symbolic analysis
- ZIV used 44% of time and proves 85% of indep
- Strong-SIV used 33% of time and proves 5% (success per application 97%)
- S-SIV, 0-SIV, x-SIV used 41%
- MIV used only 5% of time
- Delta used 8% of time, proves 5% of indep
- Coupled subscripts rare (20% overall, but concentrated)

Basics: Coupled Subscript Groups

- Why are they important?
  
  Coupling can cause imprecision in dependence testing

  DO I = 1, 100
  S1 A(I+1,I) = B(I) + C
  S2 D(I) = A(I,I) * E
  ENDDO
Dealing w/ Coupled Groups

- subscript-by-subscript testing too imprecise
  However, we could intersect deps
  
  \[
  \begin{align*}
  \text{DO } & I = 1, 100 \\
  S1 & A(I+1,I) = B(I) + C \\
  S2 & D(I) = A(I,I) \times E
  \end{align*}
  \]

  first yields \(d=+1\), second \(d=0\). That’s impossible. Therefore, no dependence

- Delta test uses this intuition when the subscripts are SIV to apply information between indices

Constraints

- An assertion about an index that must hold for a dependence to exist.
- So, when intersection of constraints is empty, must be independent
- In Delta test we generate constraints from SIV tests, so distance (or direction vector) is sufficient

Type of Constraints

- Dependence distance: \(<d>\)
  For \(I\)
  
  \[
  \]
  first subscript \(\rightarrow d = 1\)
  For \(I\)
  
  \[
  \]
  second subscript \(\rightarrow <i-i'=-1>\)

- Dependence line: \(<ax+by = c>\)
  For \(I\)
  
  \[
  \]
  first subscript \(\rightarrow 2i-i'=N+1\)

- Dependence Point: \(<x,y>\)

Delta Test

Procedure delta(subscr, constr)

  Init constraint vector \(C\) to \(<\text{none}>\)
  while exist untested SIV subscripts in subscr
    apply SIV test to all untested SIV subscripts
    return independence, or derive new constraint vector \(C'\).
    \(C' \leftarrow C \cap C'\)
    if \(C' = \emptyset\) then return independence
    else if \(C \neq C'\) then
      \(C \leftarrow C'\)
      propagate \(C\) into MIV subscripts
  test remaining MIV subscripts using MIV tests
  return independence if no solution
  while exist untested RDIV subscripts
    test and propagate RDIV constants
  test remaining MIV subscripts using MIV tests
  intersect direction vectors with \(C\), and return
Examples

For I
For J
A[I+1,I+J] = ... = A[I, I+J-1]

Apply SIV to yield:

\[ \Delta I = 1 \]
\[ I_0 + J_0 = I_0 + \Delta I + J_0 + \Delta J - 1 \]
\[ 0 = \Delta I + \Delta J - 1 \]
\[ \Delta J = 0 \]

For I, For J, For K

Apply SIV to yield:

\[ \Delta I = 1 \]
\[ J_0 - I_0 = J_0 + \Delta J - I_0 - \Delta I \]
\[ 0 = \Delta J - \Delta I \]
\[ 0 = \Delta J - 1 \]
\[ \Delta J = 1 \]
\[ \Delta K = -1 \]

RDIV

- Propagating MIV constraints is often expensive
- Restricted double index variable constraints are a special case which can often be done exactly.

For I
For J

RDIV form: I+c1, J+c2 & J+c3, I+c4
- Use Delta across subscripts.

\[ I+1 = J+\Delta J \]
\[ J+2 = I+\Delta I \rightarrow \Delta I + \Delta J = 3 \]

Merging Results

- After we test all subscripts we have vectors for each partition. Now we need to merge these into a set of direction vectors for the memory reference
- Since we partitioned into separable sets we can do cross-product of vectors from each partition.
- Start with a single vector = (*,*,...,*) of length depth of loop nest.
- Foreach partition, for each index involved in vector create new set from old vector-these indicies x this set

Example Merge

For I
For J
S_1 A[J-1] = ...
S_2 ... = A[J]

For 1st subscript in A using S_1 as source and S_2 as target: J has DV of -1
Merge -1 into (*,*) \( \rightarrow \) (*,-1). What does this mean?
- (\,<,-1\,): true dep in outer loop
- (\,=,-1\,): anti-dep from S_2 to S_1 \( \rightarrow \) (\,=,1\,)
- (\,>,-1\,): anti-dep from S_2 to S_1 in outer loop \( \rightarrow \) (\,<,-1\,)
Next Time...

- Improving cache locality using dependence information