Dependence Testing

Solving System of Diaphatine Equations

Dependence Testing

```
DO i1 = L1, U1, S1
     DO i2 = L2, U2, S2
        ...
          DO in = Ln, Un, Sn
     S1     A(f1(i1,...,in),...,fm(i1,...,in)) = ...
     S2    ... = A(g1(i1,...,in),...,gm(i1,...,in))
          ENDDO
      ...
     ENDDO
ENDDO
```

- A dependence exists from S1 to S2 if and only if there exist iteration vectors x=(x1,x2,…,xn) and y=(y1,y2,…,yn) such that

  (1) x is lexicographically less than or equal to y;

  (2) the system of ***diophatine equations*** has an integral solution:
  fi(x) = gi(y) for all 1 ≤ i ≤ m

  i.e. fi(x1,…,xn)-gi(y1,…,yn)=0 for all 1 ≤ i ≤ m
- Terminology: each fi(i1,…,in) and gi(i1,…,in) is called a ***subscript***

Example

```
DO I = 1, 10
   DO J = 1, 10
      DO K = 1, 10
S1         A(I, J, K+1) = A(I, J, K)
S2         F(I,J,K) = A(I,J,K+1)
      ENDDO
   ENDDO
ENDDO
```

- To determine the dependence between A(I,J,K+1) at iteration vector (I1,J1,K1) and A(I,J,K) at iteration vector (I2,J2,K2), solve the system of equations

  I1 = I2; J1=J2; K1+1=K2; 1 <= I1,I2,J1,J2,K1,K2 <= 10

  - Distance vector is (I2-I1,J2-J1,K2-K1)=(0,0,1)
  - Direction vector is (=,=,<)
  - The dependence is from A(I,J,K+1) to A(I,J,K) and is a true dependence

The Delta Notation

- Goal: compute iteration distance between the source and sink of a dependence

```
DO I = 1, N
   A(I + 1) = A(I) + B
ENDDO
```

  - Iteration at source/sink denoted by: $I0$ and $I0+\Delta I$
  - Forming an equality gets us: $I0 + 1 = I0 + \Delta I$
  - Solving this gives us: $\Delta I = 1$
- If a loop index does not appear, its distance is *
  - * means the union of all three directions <,>,=

```
DO I = 1, 100
DO J = 1, 100
   A(I+1) = A(I) + B(J)
```

  - The direction vector for the dependence is (<, *)

Complexity of Testing

- Find integer solutions to a system of Diophantine Equations is NP-Complete
  - Most methods consider only linear subscript expressions
- Conservative Testing
  - Try to prove absence of solutions for the dependence equations
  - Conservative, but never incorrect
- Categorizing subscript testing equations
  - ZIV if it contains no loop index variable
  - SIV if it contains only one loop index variable
  - MIV if it contains more than one loop index variables

    A(5,I+1,j) = A(1,I,k) + C

    5=1 is ZIV; I1+1=I2 is SIV; J1=K2 is MIV

Separability of subscripts

- A subscript is separable if the loop index variables it contains do not occur in other subscripts
  - Two different subscripts are ***coupled*** if they contain the same loop index variable
- Example

```
DO I = 1, 100
S1          A(I+1,I) = B(I,J) + C
S2          D(I,J+1) = A(I,I) * E
ENDDO
```

Solving each subscript equation independently would cause imprecision when subscripts are coupled

Overview of Testing Algorithm

- Partition subscripts equations (each equation contains a pair of array subscripts) into separable groups
  - Each group contains a single subscript equation or a set of coupled subscript equations
- Process each separable group
  - If the group contains a single subscript equation, classify it is ZIV, SIV, or MIV. Try to solve the equation and encode result in a dependence distance/direction vector
  - If the group contains multiple equations, try to solve them collectively
  - If at any point it can be proven that no solution exists for the group, exit and report no dependence.
- Merge all dependence/direction vectors computed in the previous steps
  - Result of each group contains iteration relations of different loop levels

Simple Subscript Tests

- ZIV Test

```
DO j = 1, 100
   A(e1) = A(e2) + B(j)
ENDDO
```

  - e1,e2 are constants or loop invariant symbols
  - If (e1-e2)!=0, then no Dependence exists
- SIV Test
  - Strong SIV Test: <a*i+c1, a*i+c2>
    - a,c1,c2 are constants or loop invariant symbols
    - For example, <4i+1,4i+5> <i+3,i>
    - Solution: d=(c2-c1)/a is integer and |d| <= |Ui-Li|
  - Weak SIV Test: <a1*i+c1, a2*i+c2>
    - a1,a2,c1,c2 are constants or loop invariant symbols
    - For example, <4i+1,2i+5> <i+3,2i>
- **NOTE: all iteration solutions must be integers and within respective loop bounds**

Simple Subscript Tests (Cont.)

- Weak-zero SIV: <a1*i+c1,c2>
  - Solution: i=(c2-c1)/a1 is integer and |i|<=|U-L|
  - Application: loop peeling

```
   DO i = 1, N
S1   Y(i, N) = Y(1, N) + Y(N, N)
   ENDDO
```

⇨

```
Y(1, N) = Y(1, N) + Y(N, N)
DO i = 2, N-1
   S1  Y(i, N) = Y(1, N) + Y(N, N)
ENDDO
Y(N, N) = Y(1, N) + Y(N, N)
```

- Weak Crossing SIV: <a*i+c1,-a*i+c2>
  - Solution: i=(c2-c1)/2a, i is integer, and |i|<=|U-L|
  - Application: loop splitting

```
   DO i = 1, N
S1   A(i) = A(N-i+1) + C
   ENDDO
```

⇨

```
DO i = 1,(N+1)/2
   A(i) = A(N-i+1) + C
ENDDO
DO i = (N+1)/2 + 1, N
   A(i) = A(N-i+1) + C
ENDDO
```

Breaking Conditions

- Sometimes a dependence exists conditionally

```
    DO I = 1, L
S1          A(I + N) = A(I) + B
    ENDDO
```

  - If L<=N, then there is no dependence from S1 to itself
  - L<=N is called the Breaking Condition

- Separate independent code for optimization

```
    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
```

Linear Diophantine Equations

- For simplicity, assume a pair of array subscripts

  f(x)=a0+a1*I1+b2*I2+…+an*In

  g(x)=b0+b1*I1+b2*I2+…+bn*In

  - a0,a1,…,an,b0,b1,…,bn are loop invariant constants
  - I1,I2,…,In are loop index variables
- To compute dependence between iteration vectors x=(x1,x2,…,xn) and y=(y1,y2,…,yn)
  - We need to solve h(x)=f(x)-g(y)=0 i.e.

  a1*x1-b1*y1 + … + an*xn-bn*yn = b0-a0

  which is a ***linear Diophantine Equation***

Solving Linear Diophantine Equations

- GCD test: existence of integer solutions
  - There exist x1,x2,…,xn,y1,y2,…,yn so that

    a1*x1-b1*y1+…+an*xn-bn*yn=b0-a0

    if and only if gcd(a1,…,an,b1,…,bn) divides b0 - a0
  - However, this does not integrate any bound information of loop index variables, and the gcd(a1,…,an,b1,…,bn) is often 1.
- Banerjee test: existence of real Solutions
  - A solution exists iff: inf(h) <= 0 <= sup(h)
    - h= **a1*x1-b1*y1+…+an*xn-bn*yn + a0-b0**

Banerjee Inequality

- Lemma 3.2. Let t,l,u,z be real numbers. If l <= z <= u, then $-t^{-}u + t^{+}l \le tz \le t^{+}u - t^{-}l$
  - where
$$a^{+} = \begin{cases} a & a \ge 0 \\ 0 & a < 0 \end{cases} \qquad a^{-} = \begin{cases} a & a < 0 \\ 0 & a \ge 0 \end{cases}$$
  - Furthermore, there are numbers z1 and z2 in [l,u] that make each of the inequalities true
- Theorem 3.3 (Banerjee)
  - Let D be a direction vector, and h be a dependence function. h = 0 can be solved in the region R iff
$$\sum_{i=1}^{n} H_i^{-}(D_i) \le b_0 - a_0 \le \sum_{i=1}^{n} H_i^{+}(D_i)$$

Where Hi = (ai*xi - bi*yi), Di (dep. direction for subscript i) indicates xi=yi, xi<yi, or xi>yi

Banerjee Inequality

- Check for all cases of Di .
  - If Di = '=', then xi=yi and hi=(ai-bi)*xi.

$$-(a_i - b_i)^- U_i + (a_i - b_i)^+ L_i = H_i^-(=) \le h \le (a_i - b_i)^+ U_i - (a_i - b_i)^- L_i = H_i^+(=)$$

  - If D_i = '<' , we have that $L_i <= x_i < y_i <= U_i$.

  Rewrite as $L_i <= x_i <= y_i -1 <= U_i - 1$

  Rewrite h as $h_i = a_i x_i - b_i y_i = a_i x_i - b_i(y_i - 1) - b_i$

  Use 3.2 to minimize $a_i x_i$ and get:

$$-a_i^-(y_i - 1) + a_i^+ L_i - b_i(y_i - 1) - b_i \le h_i \le a_i^+(y_i - 1) - a_i^- L_i - b_i(y_i - 1) - b_i$$

  Minimizing the $b_i(y_i-1)$ term then gives us:

$$-(a_i^- + b_i)^+(U_i - 1) + (a_i^- + b_i)^- L_i + a_i^+ L_i - b_i = H_i^-(<) \le h_i$$

$$\le (a_i^+ - b_i)^+(U_i - 1) - (a_i^+ - b_i)^- L_i - a_i^- L_i - b_i = H_i^+(<)$$

  - If Di = `>`, the computation is similar

Example

```
DO I = 1, N
  DO J = 1, M
    DO K = 1, 100
      A(I,K) = A(I+J,K) + B
    ENDDO
  ENDDO
ENDDO
```

- Testing (I, I+J) for D = (=,<,*):

$$H_1^-(=) + H_2^-(<) = -(1-0)^- N + (1-1)^+ 1 - (0^- + 1)^+ (M-1) + [(0^- + 1)^- + 0^+]1 - 1 = -M \le 0$$

$$\le H_1^+(=) + H_2^+(<) = (1-1)^+ N - (1-1)^- 1 + (0^+ - 1)^+ (M-1) - [(0^+ - 1)^- + 0^-]1 - 1 \le -2$$

- This is impossible, so the dependency doesn't exist.

Complex Iteration Spaces

- The iteration space is not always rectangular
  - Triangular: One of the loop bounds is a function of at least one other loop index
  - Trapezoidal: Both the loop bounds are functions of at least one other loop index
- Example

```
      DO I = 1,N
         DO J = L0 + L1*I, U0 + U1*I
S1       A(J + d) = ……
S2       …… = A(J) + B
         ENDDO
      ENDDO
```

  - Strong SIV test gives dependence if

    |d|<=|U0-L0+(U1-L1)*I| for any iteration of I
  - Banerjee test also assumes independence of loop index variables

Trapezoidal Banerjee Test

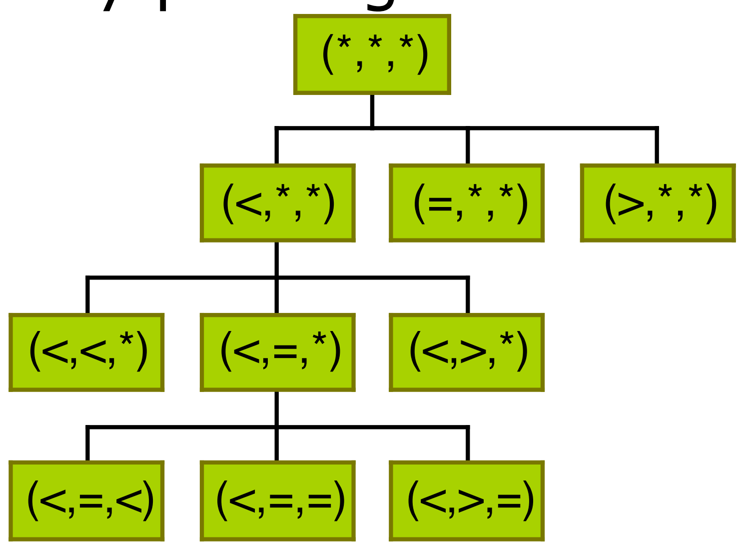
- Assume that: $$U_i = U_{i0} + \sum_{j=1}^{i-1} U_{ij} i_j$$
$$L_i = L_{i0} + \sum_{j=1}^{i-1} L_{ij} i_j$$

- Now, our bounds must change. For example:

$$H_i^-(<) = -(a_i^- + b_i)^+ \left(U_{i0} - 1 + \sum_{j=1}^{i-1} U_{ij} y_j \right) + (a_i^- + b_i)^- \left(L_{i0} + \sum_{j=1}^{i-1} L_{ij} y_j \right)$$
$$+ a_i^+ \left(L_{i0} + \sum_{j=1}^{i-1} L_{ij} x_j \right) - b_i$$

Testing Direction Vectors

- To use Banerjee test
  - Must test pair of statements for all direction vectors
  - Potentially exponential in loop nesting.
  - Can save time by pruning:



Coupled Groups

- So far, we've assumed separable subscripts.
  - We can glean information from separable subscripts, and use it to split coupled groups.
  - Most subscripts tend to be SIV in practice, where this works pretty well.
- Delta test for coupled subscript groups
  - Maintain one constraint for each loop index variable in the group.
  - Derive and propagate constraints from SIV subscripts.
  - Constraints are also propagated from MIV subscripts of the form $< a_1 i + c_1, a_2 j + c_2 >$

Delta Example

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

- The delta test gives us a distance vector of (1,1,-1) for this loop nest

More Techniques

- Solving $h(x) = 0$ is essentially an integer programming problem. Linear programming techniques can be used
  - The Omega test, I-test, etc.
- Techniques for solving systems of linear equations (e.g., Gaussian elimination) can also be adapted to compute integer solutions