COMP 515: Advanced Compilation for Vector and Parallel Processors

Prof. Krishna Palem
Prof. Vivek Sarkar
Department of Computer Science
Rice University
{palem,vsarkar}@rice.edu

https://wiki.rice.edu/confluence/display/PARPROG/COMP515

Acknowledgments

- Slides from previous offerings of COMP 515 by Prof. Ken Kennedy
  - http://www.cs.rice.edu/~ken/comp515/

Dependence Testing

Allen and Kennedy, Chapter 3 (up to Section 3.3.2)

The General Problem

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

Under what conditions is the following true for iterations α and β ?

$f_i(\alpha) = g_i(\beta)$ **for all** i, $1 \le i \le m$

Note that the number of equations equals the rank of the array,

and the number of variables is twice the number of loops that enclose both array references (two iteration vectors)

Basics: Complexity

A subscript equation is said to be

- ZIV if it contains no index (zero index variable)
- SIV if it contains only one index (single index variable)
- MIV if it contains more than one index (multiple index variables)

For Example:

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

        First subscript equation is ZIV

        Second subscript equation is SIV

        Third subscript equation is MIV
```

Terminology: Indices and Subscripts

Index: **Index variable for some loop surrounding a pair of references**

Subscript: ***A PAIR of subscript positions in a pair of array references (corresponds to dependence equation for that dimension)***

For Example:

```
A(I,j) = A(I,k) + C

        ‹I,I› is the first subscript

        ‹j,k› is the second subscript
```

Basics: Separability

- A subscript is separable if its indices do not occur in other subscripts
- If two different subscripts contain the same index they are coupled

For Example:

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

         Both subscripts are separable

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

         Second and third subscripts are coupled
```

Basics: Coupled Subscript Groups

- Why are they important?

  Ignoring coupled subscripts may lead to imprecision in dependence testing

e.g., is there a loop-carried dependence on *A* in the following loop?

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

Basics: Conservative Testing

- Consider only linear subscript expressions
- Finding integer solutions to system of linear Diophantine Equations is NP-Complete
- Most common approximation is Conservative Testing

  See if you can assert

  "No dependence exists between two subscripted references of the same array"
- Never incorrect, may be less than optimal

Dependence Testing: Overview

- Partition subscripts of a pair of array references into separable and coupled groups
- Classify each subscript as ZIV, SIV or MIV
- For each separable subscript apply single subscript test. If not done goto next step
- For each coupled group apply multiple subscript test
- If still not done, merge all direction vectors computed in the previous steps into a single set of direction vectors

Step 1: Subscript Partitioning

- **Partitions the subscripts into separable and minimal coupled groups**
- **Notations**

// *S* is a set of *m* subscript pairs $S_1, S_2, \ldots S_m$ each enclosed in

// *n* loops with indexes $I_1, I_2, \ldots I_n$, which is to be

// partitioned into separable or minimal coupled groups.

// *P* is an output variable, containing the set of partitions

// n_p is the number of partitions

Step 2: Classify as ZIV/SIV/MIV

- Easy step
- Just count the number of different indices in a subscript equation

Step 3: Applying Single Subscript Tests

- ZIV Test
- SIV Test
  - Strong SIV Test
  - Weak SIV Test
    - Weak-zero SIV
    - Weak Crossing SIV
- SIV Tests in Complex Iteration Spaces

ZIV Test

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

e1,e2 are constants or loop invariant symbols

If (e1-e2)!=0 No Dependence exists

Program analyses that can improve the accuracy of this test include constant propagation, value numbering, and symbolic "definitely different" analysis (inferring that e1 = e2 + nonzero-constant)

Strong SIV Test

- Strong SIV subscripts are of the form

$$\langle ai + c_1, ai + c_2 \rangle$$

where a ≠ 0

- For example the following are strong SIV subscripts

$$\langle i + 1, i \rangle$$

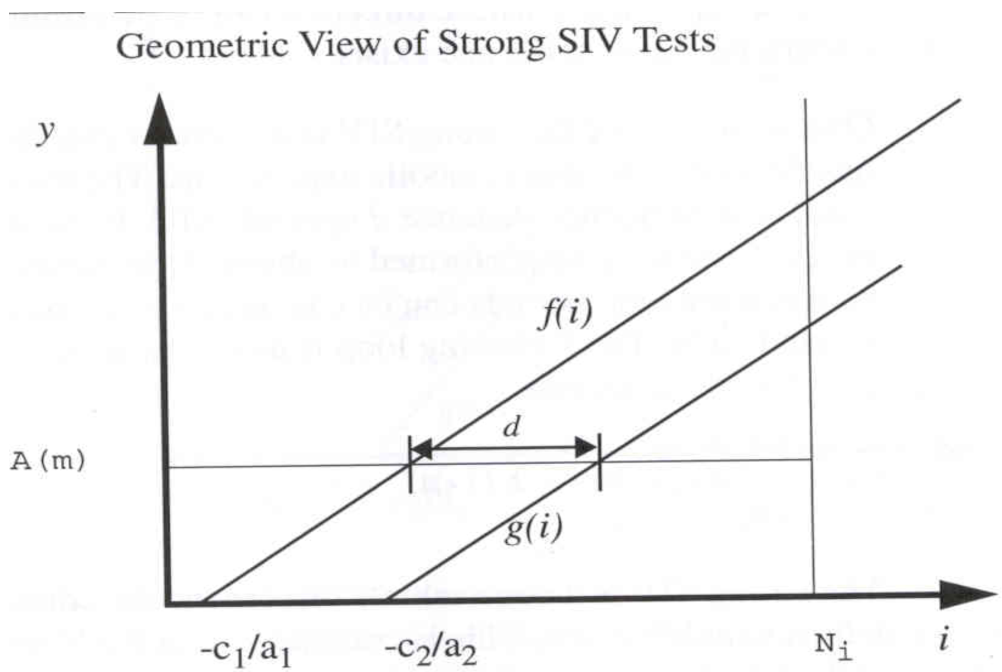
$$\langle 4i + 2, 4i + 4 \rangle$$

- Strong subscripts are also referred to as “uniformly generated”

Strong SIV Test Example

```
DO k = 1, 100
   DO j = 1, 100
S1       A(j+1,k) = ...
S2              ... = A(j,k) + 32
   ENDDO
ENDDO
```

Strong SIV Test



$$d = i' - i = \frac{c_1 - c_2}{a}$$

Dependence exists if there is an integer value of d within loop bounds,

$$|d| \leq U - L$$

Weak SIV Tests

- Weak SIV subscripts are of the form

$$\langle a_1 i + c_1, a_2 i + c_2 \rangle$$

where $a_1 \neq 0$ (without loss of generality)

- For example the following are weak SIV subscripts

$$\langle i + 1, 5 \rangle$$
$$\langle 2i + 1, i + 5 \rangle$$
$$\langle 2i + 1, -2i \rangle$$

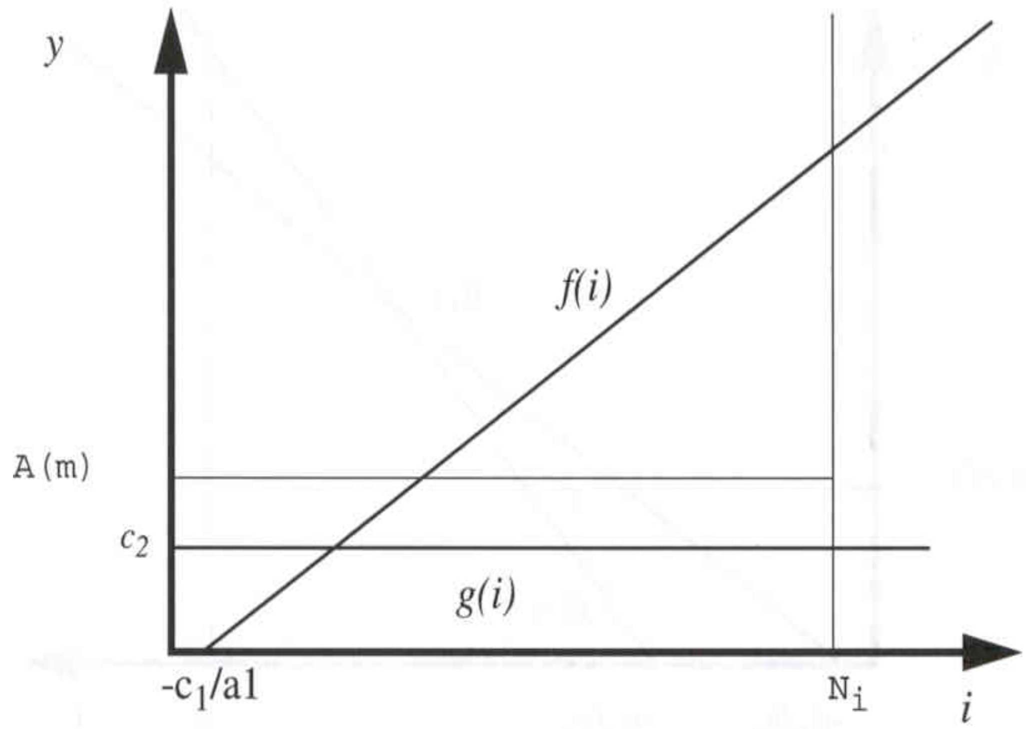
Weak-zero SIV Test

- Special case of Weak SIV where one of the coefficients (a_2) of the index is zero
- The test consists merely of checking whether the solution is an integer and is within loop bounds

$$i = \frac{c_2 - c_1}{a_1}$$

Weak-zero SIV Test

Geometric View of Weak-zero SIV Subscripts



Weak-zero SIV & Loop Peeling

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

Can be loop peeled to...

```
        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 Test

- Special case of Weak SIV where the coefficients of the index are equal in magnitude but opposite in sign i.e., $a_2 = -a_1$
- The test consists merely of checking whether the solution index

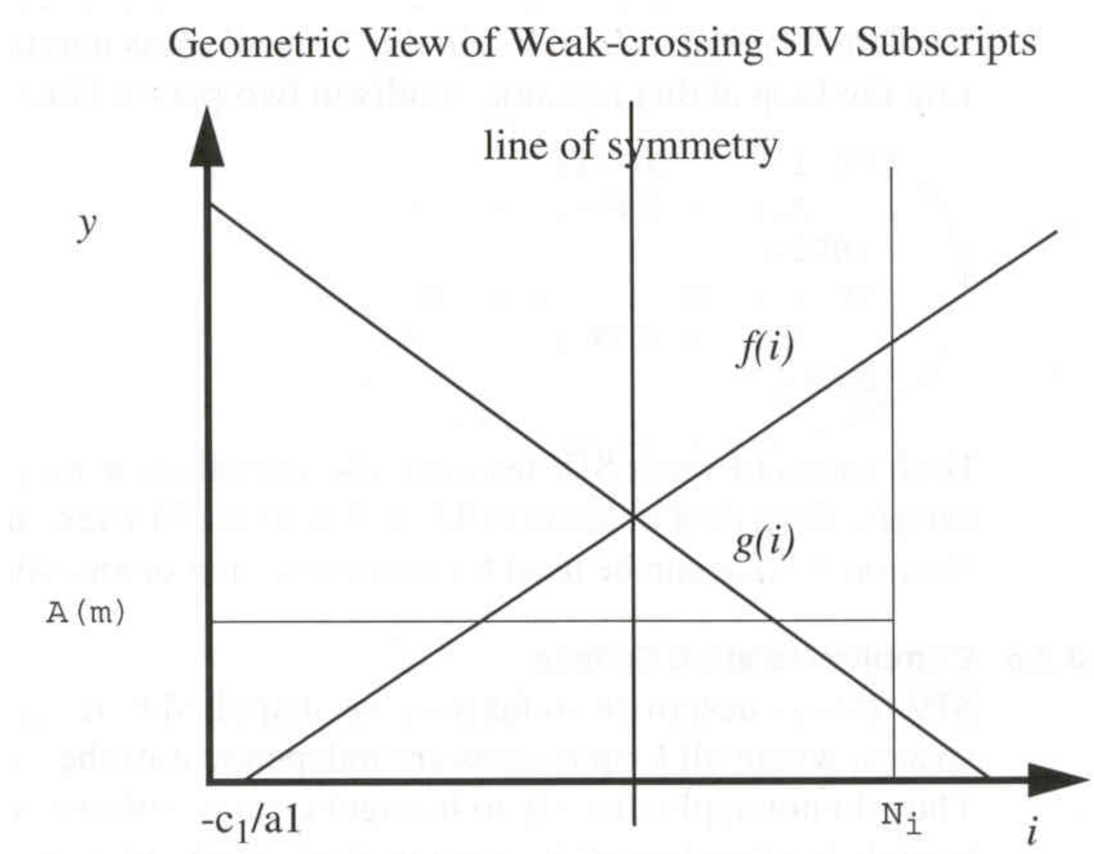
  is 1. within loop bounds and is

  2. either an integer or has a non-integer

  part equal to 1/2

$$i = \frac{c_2 - c_1}{2a_1}$$

Weak-crossing SIV Test



Weak-crossing SIV & Loop Splitting

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

This loop can be split into...

```
          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
```

Complex Iteration Spaces

- Till now we have applied the tests only to rectangular iteration spaces
- These tests can also be extended to apply to triangular or trapezoidal loops
  - Triangular: One of the loop bounds is a function of at least one outer loop index
  - Trapezoidal: Both the loop bounds are functions of at least one outer loop index

Complex Iteration Spaces

- For example consider this special case of a strong SIV subscript

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

Complex Iteration Spaces

- **Strong SIV test gives dependence if**

$$|d| \le U_0 - L_0 + (U_1 - L_1)I$$

$$I \ge \frac{|d| - (U_0 - L_0)}{U_1 - L_1}$$

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

Index Set Splitting

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

For values of $I < \frac{|d| - (U_0 - L_0)}{U_1 - L_1} = \frac{20 - (-1)}{1} = 21$

there is no dependence

Index Set Splitting

- 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

Coupling makes these tests imprecise

```
  DO I = 1,100
          DO J = 1, I
S1                A(J+20,I) = A(J,19) + B
          ENDDO
  ENDDO
```

- We will report dependence even if there isn't any
- But such cases are very rare

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 compiler 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
```