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

Dependence: Theory and Practice

Allen and Kennedy, Chapter 2

Distance Vectors (Recap)

- Consider a dependence in a loop nest of n loops
  - Statement S_1 on iteration i is the source of the dependence
  - Statement S_2 on iteration j is the sink of the dependence

- The distance vector is a vector of length n d(i,j) such that: $d(i,j)_k = j_k - i_k$

- We normalize distance vectors for loops in which the index step size is not equal to 1
  - It's usually simpler to convert all loops to have a step of +1 before computing distance vectors

Direction Vectors (Recap)

- Definition 2.10 in the book:

  Suppose that there is a dependence from statement S_1 on iteration i of a loop nest of n loops and statement S_2 on iteration j, then the dependence direction vector is D(i,j) is defined as a vector of length n such that

$$D(i,j)_k = \begin{cases} \text{"<" if } i_k < j_k & \text{equivalently, if } d(i,j)_k > 0 \\ \text{"=" if } i_k = j_k & \text{equivalently, if } d(i,j)_k = 0 \\ \text{">" if } i_k > j_k & \text{equivalently, if } d(i,j)_k < 0 \end{cases}$$

- A direction vector element summarizes a set of distances
  - "<" summarizes the set {1, 2, 3, …}
  - "=" summarizes the singleton set { 0 }
  - ">" summarizes the set {-1, -2, -3, …}
  - and so on …

Implausible Distance & Direction Vectors

- A distance vector is implausible if its leftmost nonzero element is negative i.e., if the vector is lexicographically less than the zero vector
- Likewise, a direction vector is implausible if its leftmost non "=" component is not "<"
- No dependence in a sequential program can have an implausible distance or direction vector as this would imply that the sink of the dependence occurs before the source.

Direction Vector Transformation

- Theorem 2.3. Direction Vector Transformation. Let T be a transformation that is applied to a loop nest and that does not rearrange the statements in the body of the loop. Then the transformation is valid if, after it is applied, none of the direction vectors for dependences with source and sink in the nest has a leftmost non- "=" component that is ">" i.e., none of the transformed direction vectors become implausible.

- Follows from Fundamental Theorem of Dependence:
  - All dependences exist
  - None of the dependences have been reversed

Loop-carried and Loop-independent Dependences

- If in a loop statement S_2 depends on S_1, then there are two possible ways of this dependence occurring:

1. S_1 and S_2 execute on different iterations
   - This is called a loop-carried dependence.
2. S_1 and S_2 execute on the same iteration
   - This is called a loop-independent dependence.

Loop-carried dependence

- Definition 2.11
- Statement S_2 has a loop-carried dependence on statement S_1 if and only if S_1 references location M on iteration i, S_2 references M on iteration j and $d(i,j) > 0$ i.e., D(i,j) contains a "<" as leftmost non "=" component and is lexicographically positive.

Example:

```
DO I = 1, N
S1        A(I+1) = F(I)
S2        F(I+1) = A(I)
ENDDO
```

Loop-carried dependence

- Level of a loop-carried dependence is the index of the leftmost non-“=” of D(i,j) for the dependence.

For instance:

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

- Direction vector for S1 is (=, =, <)
- Level of the dependence is 3
- A level-k dependence between S_1 and S_2 is denoted by $S_1 \delta_k S_2$

Loop-carried Transformations

- Theorem 2.4 Any reordering transformation that (1) preserves the iteration order of the level-k loop, (2) does not interchange any loop at level $<$ k to a level $>$ k, and (3) does not interchange any loop at level $>$ k to a position $<$ k, must preserve all level-k dependences.
- Proof:
  - D(i, j) has a "$<$" in the k^{th} position and "=" in positions 1 through k-1

  $\Rightarrow$ Source and sink of dependence are in the same iteration of loops 1 through k-1

  $\Rightarrow$ Cannot change the sense of the dependence by a reordering of iterations of those loops

- As a result of the theorem, powerful transformations can be applied

Loop-carried Transformations

Example:

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

can be transformed to:

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

Loop-independent dependences

- Definition 2.15. Statement S_2 has a loop-independent dependence on statement S_1 if and only if there exist two iteration vectors i and j such that:
  1) Statement S_1 refers to memory location M on iteration i, S_2 refers to M on iteration j, and i = j.
  2) There is a control flow path from S_1 to S_2 within the iteration.

Example:

```
DO I = 1, 10
S1      A(I) = ...
S2      ... = A(I)
ENDDO
```

Loop-independent dependences

More complicated example:

```
DO I = 1, 9
S1        A(I) = ...
S2      ...  = A(10-I)
ENDDO
```

- No common loop is necessary. For instance:

```
DO I = 1, 10
S1      A(I) = ...
ENDDO
DO I = 1, 10
S2      ...  = A(20-I)
ENDDO
```

Loop-independent dependences

- Theorem 2.5. If there is a loop-independent dependence from S_1 to S_2, any reordering transformation that does not move statement instances between iterations and preserves the relative order of S_1 and S_2 in the loop body preserves that dependence.

- S_2 depends on S_1 with a loop independent dependence is denoted by $S_1 \delta_\infty S_2$
- Note that the direction vector will have entries that are all "=" for loop independent dependences

Simple Dependence Testing

- Theorem 2.7: Let a and b be iteration vectors within the iteration space of the following loop nest:

```
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
```

Simple 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 S_1 to S_2 if and only if there exist values of α and β such that (1) α is lexicographically less than or equal to β and (2) the following system of dependence equations is satisfied:

  $f_i(\alpha) = g_i(\beta)$ for all i, $1 \leq i \leq m$
- Direct application of Loop Dependence Theorem