Chapter 2: Dependence, Parallelization and Vectorization

By

Prof. Ken Kennedy

Presented by

Hazim A. Shafi

COMP 515

February 2, 1995

Data Dependence

Motivation:

- Expose parallelism
- Maintain Correctness

Definition:

A dependence between statements S_1 and S_2 exists *iff*:

1. There is an execution path from S_1 to S_2.
2. *Both* S_1 and S_2 access the same memory location *and* at least one is a write.
   - S_1 reads a location written to by S_2
   - S_2 reads a location written to by S_1
   - Both S_1 and S_2 write to the same location

Data Dependences

- *True Dependence*

  Example:

  ```
  S1 PI = 3.14
  S2 R = 5.0
  S3 AREA = PI * R ** 2
  ```

  - S_2 writes to `R` and S_3 later reads `R`
  - S_3 depends on S_2 denoted: $S_2 \quad \delta \quad S_3$

Data Dependences (cont.)

- *Anti-Dependence*

  Example:

  ```
  S₁ PI = 3.14
  S₂ R = 5.0
  S₃ AREA = PI * R ** 2
  S₄ PI = 22 / 7
  ```

  - S_4 writes PI which is read by S_3
  - S_4 cannot precede S_3
  - $S_3\ \delta^{-1}\ S_4$

Data Dependences (cont.)

- *Output Dependence*

  Example:

```
S1 X = 1
S2 .   .   .
S3 X = 2
S4 W = X * Y
```

  - S_1 writes X
  - S_3 rewrites X
  - S_4 should read X written by S_3
  - Notation: $S_1\ \delta^o\ S_3$

Loops: Notation

Iteration Vectors:

Contain an entry for each loop control variable.

Example:

```
DO I = 1, 10
     DO J = 1 , 20
         S1
     ENDDO
ENDDO
```

To represent the execution of S_1 when I=2 and J=3, the iteration vector would be = (2,3).

- Notice that the vector entries start with the outermost loop variable at the left.

Loops: Notation (cont.)

Assume:

1. $\mathbf{i}$ is an iteration vector.
2. $\mathbf{i}_k$ is the kth element of $\mathbf{i}$
3. $\mathbf{i}[\mathbf{1}:k]$ contains leftmost k elements of $\mathbf{i}$

Then,

$\mathbf{i}<\mathbf{j}$ (iteration $\mathbf{i}$ *precedes* iteration $\mathbf{j}$) *iff*:

1. $\mathbf{i[1:n\text{-}1]<j[1:n\text{-}1]}$
2. $\mathbf{i[1:n\text{-}1] = j[1:n\text{-}1]}$ **and** $\mathbf{i}_n < \mathbf{j}_n$

Examples:

- (1,2,3,4,5) < (2,2,3,4,5)
- (1,1,2,3,4) < (1,1,2,4,4)

Reordering Transformations:

Definition:

A *reordering transformation* is any program transformation that merely changes the order of execution of the code, without adding or deleting any statements.

Fundamental Theorem of Dependence:

Any reordering transformation that preserves every dependence in a program preserves the meaning of the program.

Results:

1. **Bernstein's Conditions:**

   Two non-intersecting regions R_1 and R_2 can be run in parallel without any synchronization if *none* of these conditions apply

   (a) R_1 writes into a location that R_2 reads.

   (b) R_1 reads from a location that R_2 writes.

   (c) R_1 and R_2 write into the same location.

2. **Fundamental Theorem of Parallelization**

   Two groups of statements, G_1 and G_2 in a loop-free program can be safely executed in parallel if there is *no dependence* from a statement in one group to a statement in the other

Distance and Direction Vectors

Distance Vectors:

- **Definition:**

  For two n-length iteration vectors $\mathbf{i}$, $\mathbf{j}$ the *distance vector* $\mathbf{d(i,j)}$ is an n-length vector with:

  $\mathbf{d(i,j)}_k = \mathbf{j}_k - \mathbf{i}_k$

- **Lemma:**

  $\mathbf{i} < \mathbf{j}$ *iff* $\mathbf{d(i,j)} > \mathbf{0}$

Direction Vectors:

- **Definition:**

  For 2 n-length iteration vectors **i, j**, the *direction vector* $\mathbf{D(i,j)}$ is an n-length vector such that:

  $\mathbf{D(i,j)}_k =$

  $$\begin{array}{l} \text{“}<\text{” if } \mathbf{d(i,j)} > 0 \\ \text{“}=\text{” if } \mathbf{d(i,j)} = 0 \\ \text{“}>\text{” if } \mathbf{d(i,j)} < 0 \end{array}$$

Distance and Direction Vectors

Example:

```
DO I = 1, 10
   DO J = 1, 20
     DO K = 1, 30
       a(I,J,K) = a(I-1,J,K+1) * b(I,J,K+1)
     ENDDO
   ENDDO
ENDDO
```

$\mathbf{D(i,j)} = (<,=,>)$

- **Lemma:**

  i<j *iff* the leftmost non-"=" component of **D(i,j)** is "<"

- **Theorem:**

  Let **T** be a transformation that does not rearrange the statements in the body of the loop. **T** is legal if *none* of the direction vectors for dependences with the source and sink in the nest has a leftmost non-"=" component that is ">" after **T** is applied.

Distance and Direction Vectors

Example:

```
DO I = 1, N
    DO J = 1, M
        a(I,J) = a(I-1,J) * b(I,J)
    ENDDO
ENDDO
```

Direction vector is $(<,=)$

Transformation 1:

```
DO I = N, 1, -1
    DO J = 1, M
        a(I,J) = a(I-1,J) * b(I,J)
    ENDDO
ENDDO
```

Direction vector is $(>,=) \Rightarrow$ This is *invalid!*

Transformation 2:

```
DO J = 1, M
    DO I = 1, M
        a(I,J) = a(I-1,J) * b(I,J)
    ENDDO
ENDDO
```

Direction vector is $(=,<) \Rightarrow$ This is *valid!*

Loop-carried Dependence:

Definition:

S_2 has a *loop-carried dependence* on S_1 *iff* S_1 references location M on iteration $\mathbf{i}$, S_2 references M on iteration $\mathbf{j}$ and $\mathbf{d(i,j)} > 0$

$\Rightarrow$ $\mathbf{D(i,j)}$ has "$<$" leftmost non-"$=$" component.

$\Rightarrow$ index of "$<$" is the *level* of this loop-carried dependence.

Example:

```
DO I = 1, 10
     S1   a(I+1) = f(I)
     S2   f(I+1) = a(I)
ENDDO
```

$\Rightarrow S_1 \ \delta_1 \ S_2 \ \mathit{and} \ S_2 \ \delta_1 \ S_1$

Theorem:

Any reordering transformation that does not alter loops 1 through k preserves any level-k dependence.

Example:

```
DO I = 1, 10
  DO J = 1, 10
    DO K = 1, 10
S1      A(I, J, K) = A(I-1, J-2, K-3)
    ENDDO
  ENDDO
ENDDO
```

Valid Transformation:

```
DO I = 1, 10
  DO K = 10, 1, -1
    DO J = 1, 10
S1      A(I, J, K) = A(I-1, J-2, K-3)
    ENDDO
  ENDDO
ENDDO
```

Loop-Independent Dependences:

S_2 has a *loop-independent dependence* on S_1 *iff* there exists two iteration vectors **i, j** such that:

1. S_1 refers to memory location M in iteration **i**; S_2 refers to the same location on iteration **j**; and **d(i,j)** =0
2. There is a control flow path from S_1 to S_2 within the iteration

$\Rightarrow$ Notation: $S_1 \ \delta_\infty \ S_2$

Theorem:

If S_2 has a loop-independent dependence on S_1, then any valid reordering transformation must have S_2 follow S_1

Conclusions:

Compiler has to test for and maintain all dependences.

What's next?

- Parallelization.
- Vectorization.