
COMP 515: Advanced Compilation for Vector and Parallel Processors

Vivek Sarkar
Department of Computer Science
Rice University
vsarkar@rice.edu



Acknowledgments

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

Dependence: Theory and Practice

Allen and Kennedy, Chapter 2

Dependence: Theory and Practice

What shall we cover in this chapter?

- Introduction to Dependences
 - Loop-carried and Loop-independent Dependences
 - Simple Dependence Testing
 - Parallelization and Vectorization
-

Dependences

- We will concentrate on data dependences
- Chapter 7 deals with control dependences

- Simple example of data dependence:

S_1 `PI = 3.14`

S_2 `R = 5.0`

S_3 `AREA = PI * R ** 2`

- Statement S_3 cannot be moved before either S_1 or S_2 without compromising correct results
-

Dependences

- **Formally:**

There is a data dependence from statement S_1 to statement S_2 (S_2 depends on S_1) if:

1. Both statements access the same memory location and at least one of them stores onto it, and
2. There is a feasible run-time execution path from S_1 to S_2



Load Store Classification

- Quick review of dependences classified in terms of load-store order:
 1. True dependences (RAW hazard)
 - S_2 depends on S_1 is denoted by $S_1 \delta S_2$
 2. Antidependence (WAR hazard)
 - S_2 depends on S_1 is denoted by $S_1 \delta^{-1} S_2$
 3. Output dependence (WAW hazard)
 - S_2 depends on S_1 is denoted by $S_1 \delta^0 S_2$
-

Dependence in Loops

- Let us look at two different loops:

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

```
DO I = 1, N
S1  A(I+2) = A(I) + B(I)
ENDDO
```

- In both cases, statement S_1 depends on itself
 - However, there is a significant difference
 - We need a formalism to describe and distinguish such dependences
-

Iteration Numbers

- The iteration number of a loop is equal to the value of the loop index
- Definition:
 - For an arbitrary loop in which the loop index I runs from L to U in steps of S , the iteration number i of a specific iteration is equal to the index value I on that iteration

Example:

```
DO I = 0, 10, 2
S1    <some statement>
ENDDO
```

Iteration Vectors

What do we do for nested loops?

- Need to consider the nesting level of a loop
 - Nesting level of a loop is equal to one more than the number of loops that enclose it.
 - Given a nest of n loops, the iteration vector i of a particular iteration of the innermost loop is a vector of integers that contains the iteration numbers for each of the loops in order of nesting level.
 - Thus, the iteration vector is: $\{i_1, i_2, \dots, i_n\}$
where i_k , $1 \leq k \leq n$ represents the iteration number for the loop at nesting level k
-

Iteration Vectors

Example:

```
DO I = 1, 2
  DO J = 1, 2
    S1      <some statement>
  ENDDO
ENDDO
```

- The iteration vector $S_1[(2, 1)]$ denotes the instance of S_1 executed during the 2nd iteration of the I loop and the 1st iteration of the J loop
-

Ordering of Iteration Vectors

- **Iteration Space:** The set of all possible iteration vectors for a statement

Example:

```
DO I = 1, 2
  DO J = 1, 2
    S1      <some statement>
  ENDDO
ENDDO
```

- The iteration space for S_1 is $\{ (1,1), (1,2), (2,1), (2,2) \}$
-

Ordering of Iteration Vectors

- Useful to define an ordering for iteration vectors
 - Define an intuitive, lexicographic order
 - Iteration i precedes iteration j , denoted $i < j$, iff:
 1. $i[i:n-1] < j[1:n-1]$, or
 2. $i[1:n-1] = j[1:n-1]$ and $i_n < j_n$
-

Formal Definition of Loop Dependence

- **Theorem 2.1 Loop Dependence:**
There exists a dependence from statements S_1 to statement S_2 in a common nest of loops if and only if there exist two iteration vectors i and j for the nest, such that
 - (1) $i < j$ or $i = j$ and there is a path from S_1 to S_2 in the body of the loop,
 - (2) statement S_1 accesses memory location M on iteration i and statement S_2 accesses location M on iteration j , and
 - (3) one of these accesses is a write.

 - Follows from the definition of dependence
-

Reordering Transformations

- A reordering transformation is any program transformation that merely changes the order of execution of the code, without adding or deleting any executions of any statement
 - A reordering transformation does not eliminate dependences
 - A reordering transformation *preserves* a dependence if it preserves the relative execution order of the source and sink of that dependence.
 - Fundamental Theorem of Dependence:
 - Any reordering transformation that preserves every dependence in a program preserves the meaning of that program
 - Proof by contradiction. Theorem 2.2 in the book.
 - A transformation is said to be *valid* or *legal* for the program to which it applies if it preserves all dependences in the program.
-

Distance Vectors

- 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 shall normalize distance vectors for loops in which the index step size is not equal to 1.
-

Distance Vector Example

Example:

```
DO I = 1, N
  DO J = 1, M
    DO K = 1, L
      S1      A(I+1, J, K-1) = A(I, J, K) + 10
    ENDDO
  ENDDO
ENDDO
```

- S_1 has a true dependence on itself.
—Distance Vector: $(1, 0, -1)$
 - Are there any anti or output dependences in this example?
-

Direction Vectors

- 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

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

- A direction vector element *summarizes* a set of distances
-

Direction Vector Example

Example:

```
DO I = 1, N
  DO J = 1, M
    DO K = 1, L
      S1      A(I+1, J, f(K)) = A(I, J, K) + 10
    ENDDO
  ENDDO
ENDDO
```

- S_1 has a true dependence on itself.
 - Direction Vector: $(1, 0, *)$ or $(<, =, *)$
 - S_1 has an output dependence on itself.
 - Direction Vector: $(0, 0, *)$ or $(=, =, *)$
-

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 S_1 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
-

Loop-carried and Loop-independent Dependences

- Loop-independent and loop-carried dependence partition all possible data dependences!
 - Note that if $S_1 \delta S_2$, then S_1 executes before S_2 . This can happen only if:
 - The difference vector for the dependence is less than 0, or
 - The difference vector equals 0 and S_1 occurs before S_2 textually

...precisely the criteria for loop-carried and 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  $i_1 = L_1, U_1, S_1$ 
  DO  $i_2 = L_2, U_2, S_2$ 
    ...
    DO  $i_n = L_n, U_n, S_n$ 
       $S_1$        $A(f_1(i_1, \dots, i_n), \dots, f_m(i_1, \dots, i_n)) = \dots$ 
       $S_2$        $\dots = A(g_1(i_1, \dots, i_n), \dots, g_m(i_1, \dots, i_n))$ 
    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:
$$f_i(\alpha) = g_i(\beta) \text{ for all } i, 1 \leq i \leq m$$
 - Direct application of Loop Dependence Theorem
-

Homework

- Reading list for next class
 - Chapter 2, Dependence: Theory and Practice
- Homework assignment for discussion in next class
 - Exercises 2.2 and 2.3