COMP 515: Advanced Compilation for Vector and Parallel Processors

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

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

Worksheet #1 (Lecture 1)

```
DO I = 1, N
   T = A[I]          S1
   A[I] = B[I]       S2
   B[I] = T          S3
ENDDO
```

- Using Bernstein conditions, identify pairs of statement instances that can exhibit one of the following conditions (a different pair for each condition)
   1. R1 writes into a memory location that R2 reads
   2. R2 writes into a memory location that R1 reads
   3. Both R1 and R2 write to the same memory location
   4. None of the above

Worksheet #2 (Lecture 1)

Name: ____________________ Netid: ____________________

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

1. Assuming a uniprocessor cache with one word per cache line, and unbounded ("infinite") capacity, how many cache misses (memory accesses) are incurred by the above code?

2. How does your answer change if the cache can only hold 4 words?

Dependence: Theory and Practice

Allen and Kennedy, Chapter 2

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

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 (for sequential programs):

  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 = Read After Write)
     - S_1 performs a write and S_2 performs a read
     - S_2 depends on S_1 is denoted by $S_1 \, \delta \, S_2$

  2. Antidependence (WAR = Write After Read)
     - S_1 performs a read and S_2 performs a write
     - S_2 depends on S_1 is denoted by $S_1 \, \delta^{-1} \, S_2$

  3. Output dependence (WAW = Write After Write)
     - S_1 performs a write and S_2 performs a write
     - 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 // Iter nos = 0, 2, 4, 6, 8, 10
      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: $\{i_1, i_2, ..., i_n\}$
  where i_k, $1 \leq k \leq m$ 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
  - LLT = Lexicographically Less Than
  - LGT = Lexicographically Greater Than

- For two vectors of equal length, $X = \langle x_1, x_2, \ldots, x_n \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$, LLT(X,Y) = true if and only if
  - There exists an index i such that $x_i < y_i$, and $xj = yj$ for all $1 <= j < I$

- Define LGT(X,Y) = true similarly

- Given two iteration vectors, X and Y, X < Y if LLT(X, Y) = true

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 Vector Example

Example:

```
DO I = 1, N
   DO J = 1, M
      DO K = 2, L
S1          A(I+1, J, K-1) = A(I, J, K) + 10
      ENDDO
   ENDDO
ENDDO
```

- S_1 has a true dependence on itself.
  - E.g., from (1, 1, 3) to (2, 1, 2)
  - Distance Vector: (1, 0, -1)
  - Distance between two instances in a dependence
- Are there any anti or output dependences in this example? Time for Worksheet 1!

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 (<, =, *)
  - * represents any possible distance/direction value
- 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.

Homework #1

- Homework #1, written assignment
  - Solve exercises 2.2 and 2.3 in book
    - In 2.2, dependence "type" refers to flow/anti/output
    - In 2.3, you should indicate which loops can be parallelized e.g., loop I, loop J, and/or loop K
  - Due in class on Sep 8th
  - Honor Code Policy: All submitted homeworks are expected to be the result of your individual effort. You are free to discuss course material and approaches to problems with your other classmates and the professors, but you should never misrepresent someone else's work as your own. If you use any material from external sources, you must provide proper attribution.

Problems 2.2 and 2.3

2.2 Construct all direction vectors for the following loop and indicate the type of dependence (true/anti/output) associated with each.

```
DO K = 1, 100
        DO J = 1, 100
                DO I = 1, 100
                        A(I+1, J, K) = A(I, J, 5) + B
                END DO
        END DO
END DO
```

2.3: Can the loop in Exercise 2.2 be parallelized? If so give a parallel version

Worksheet 1 (can be done in groups)

Name: ______________________ **Netid: ____________________**

Example:

```
DO I = 1, N
   DO J = 1, M
      DO K = 2, L
S1          A(I+1, J, K-1) = A(I, J, K) + 10
      ENDDO
   ENDDO
ENDDO
```

- Are there any anti or output dependences in this example? If so, list them. If not, explain why not.