REMINDEERS

- Homework 7 due December 1.
- Last class: Tuesday, December 8 (December 10 last day of classes)
- Final exam: Thursday, December 17, 4:00pm - 7:00pm, BRR 1095; accumulative, closed books, closed notes. CONFLICTS? [cs211], [cs336]
- Extension for project 2?
**Dependence — Overview**

**dependence relation:** Describes all *statement–to–statement execution orderings* for a sequential program that must be preserved if the meaning of the program is to remain the same.

There are two sources of dependences:

**data dependence**

\[
S_1 \quad \text{pi} = 3.14 \\
S_2 \quad r = 5.0 \\
S_3 \quad \text{area} = \text{pi} \times r^{**2}
\]

**control dependence**

\[
S_1 \quad \text{if (t .ne. 0.0) then} \\
S_2 \quad a = a/t \\
\quad \text{endif}
\]

How to preserve the meaning of these programs?
Execute the statements in an order that preserves the original *load/store* order.
Theorem

Any reordering transformation that preserves every dependence (i.e., visits first the source, and then the sink of the dependence) in a program preserves the meaning of that program.

Note: Dependence starts with the notion of a sequential execution, i.e., starts with a sequential program.
Dependence — Overview

**Definition** — There is a data dependence from statement $S_1$ to statement $S_2$ ($S_1 \delta S_2$) if

1. Both statements access the same memory location, and
2. There is a run–time execution path from $S_1$ to $S_2$.

**Data dependence classification**

“$S_2$ depends on $S_1$” — $S_1 \delta S_2$

- **true (flow) dependence**
  occurs when $S_1$ writes a memory location that $S_2$ later reads

- **anti dependence**
  occurs when $S_1$ reads a memory location that $S_2$ later writes

- **output dependence**
  occurs when $S_1$ writes a memory location that $S_2$ later writes

- **input dependence**
  occurs when $S_1$ reads a memory location that $S_2$ later reads. Note: Input dependences do not restrict statement (load/store) order!

CS 314

lecture 24, fall’15, page 4
Dependence — Where do we need it?

We restrict our discussion to data dependence for scalar and subscripted variables (no pointers and no control dependence).

Examples:

```fortran
    do I = 1, 100
        do J = 1, 100
            A(I,J) = A(I,J) + 1
        enddo
    enddo

doi I = 1, 99
    do J = 1, 100
        A(I,J) = A(I+1,J) + 1
    enddo
enddo
```

**vectorization**

```fortran
    A(1:100:1,1:100:1) = A(1:100:1,1:100:1) + 1
    A(1:99,1:100) = A(2:100,1:100) + 1
```

**parallelization**

```fortran
    doall I = 1, 100
        doall J = 1, 100
            A(I,J) = A(I,J) + 1
        enddo
    enddo

    implicit barrier sync.

enddo
```

```fortran
    doall I = 1, 99
        doall J = 1, 100
            A(I,J) = A(I+1,J) + 1
        enddo
    enddo

    implicit barrier sync.

enddo
```

CS 314  
lecture 24, fall’15, page 5
Dependence Analysis

Question

Do two variable references never/maybe/always access the same memory location?

Benefits

• improves alias analysis
• enables loop transformations

Motivation

• classic optimizations
• instruction scheduling
• data locality (register/cache reuse)
• vectorization, parallelization

Obstacles

• array references
• pointer references
Vectorization vs. Parallelization

**vectorization** — Find parallelism in innermost loops; fine–grain parallelism

**parallelization** — Find parallelism in outermost loops; coarse–grain parallelism

- Parallelization is considered more complex than vectorization, since finding coarse–grain parallelism requires more analysis (e.g., interprocedural analysis).

- Automatic vectorizers have been very successful
Next Lecture

Things to do:

• Dependence analysis
• OpenMP
• More on automatic vectorization / parallelization
• Work on homework 7!