Class Information

- Seventh homework submission extension: Friday, April 18, before class.
- Second project extension: **Monday, April 21, 11:59pm**. There will be a chilled water outage during parts of April 15 and April 16.

Review: 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!

Review: Dependence — Basics

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 — Where do we need it?

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

Examples:

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

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

vectorization

```
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

```
doall I = 1, 100
  doall J = 1, 100
    A(I,J) = A(I,J) + 1
  enddo
  implicit barrier sync.
enddo
implicit barrier sync.
```

```
do I = 1, 99
   doall J = 1, 100
      A(I,J) = A(I+1,J) + 1
   enddo
   implicit barrier sync.
enddo
```

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

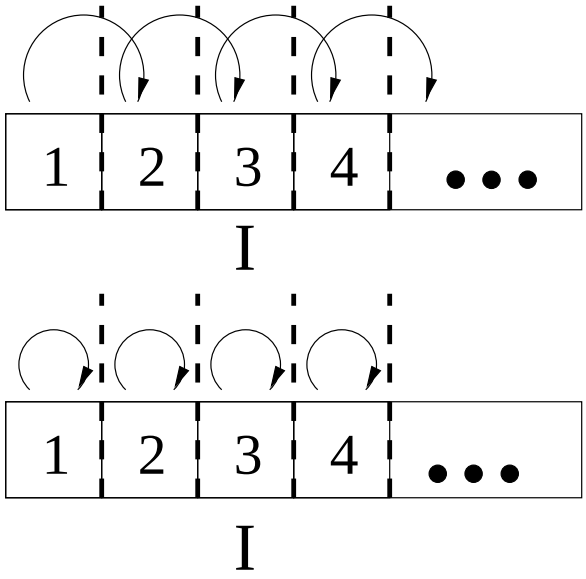
vectorization — Find parallelism in innermost loops; fine–grain parallelism

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

Dependence Analysis for Array References

```
do I = 1, 100
   A(I) =
      = A(I-1)
enddo
```




```
do I = 1, 100
   A(I) =
      = A(I)
enddo
```


A loop-independent dependence exists regardless of the loop structure. The source and sink of the dependence occur on the same loop iteration.

A loop-carried dependence is induced by the iterations of a loop. The source and sink of the dependence occur on different loop iterations.

Loop-carried dependences can inhibit parallelization and loop transformations