ITMC403 Parallel and Distributed Computing

Dependence analysis

Fundamental Assumption

- **When can two statements execute in parallel?**
  - **On one processor:**
    - statement 1;
    - statement 2;
  - **On two processors:**
    - <u>processor 1</u>: <u>processor 2</u>:
    - statement 1; statement 2;
- Processors execute **independently**: no control over order of execution between processors

Fundamental Assumption (Cont.)

- **When can two statements execute in parallel?**
  - **On two processors:**
  - Possibility 1
    - processor 1: / processor 2:

processor 1:	processor 2:
statement 1;	---------------
---------------	statement 2;

  - Possibility 2

processor 1:	processor 2:
---------------	statement 1;
statement 2;	---------------

Their order of execution must not matter! In other words,

- statement1; statement2;
- **must be equivalent to**
  - statement2; statement1;

Examples

- EXAMPLE 1
  - *a = 1;* → Statements can be executed in parallel.
  - *b = 2;*
- EXAMPLE 2
  - *a = 1;* → Statements cannot be executed in parallel
  - *b = a;* → Program modifications may make it possible.
- EXAMPLE 3
  - *a = f(x);* → May not be wise to change the program
  - *b = a;* → (sequential execution would take longer).
- EXAMPLE 4
  - *b = a;* → Statements cannot be executed in parallel.
  - *a = 1;*
- EXAMPLE 5
  - *a = 1* → Statements cannot be executed in parallel.
  - *a = 2*

Types of Dependences

True (flow) dependence –RAW read after write

- Statements ***S1***, ***S2***
  - ***S2*** has a **true dependence** on ***S1***
    - **iff**
  - ***S2*** reads a value written by ***S1***
- Denoted by *S1 d S2*

Example:

The first statement writes into a location that is read by the second.

S_1 $X = \ldots$

S_2 $\ldots = X$

We write $S_1\ d\ S_2$.

Types of Dependences (cont.)

Anti-dependence –WAR write after read

- Statements ***S1***, ***S2***
  - ***S2*** has a **anti-dependence** on ***S1***
    - **iff**
  - ***S2*** writes a value read by ***S1***
- Denoted by $S1\ d^{-1}\ S2$

Example:

The first statement reads from a location into which the second statement writes.

S_1 $\quad ... = X$

S_2 $\quad X = ...$

An anti-dependence is denote by $S_1\ d^{-1}\ S_2$

Types of Dependences (cont.)

Output dependence –WAW write after write

- Statements ***S1***, ***S2***

  - ***S2*** has a **output dependence** on ***S1***
    - **iff**
  - ***S2*** writes a value written by ***S1***

- Denoted by *S1 d* ^{0} *S2*

Example:

both statements write into the same location.

S_1 $X = \ldots$

S_2 $X = \ldots$

We write $S_1 \; d^{0} \; S_2$

When can 2 statements execute in parallel?

S1 and ***S2*** can execute in parallel

iff

there are **no dependences** between ***S1*** and ***S2***

- **true dependences**
- **anti-dependences**
- **output dependences**

Some dependences can be removed.

Data Dependence in Loops

Parallelism often occurs in loops.

```
for(i=0; i<100; i++)
    a[i] = i;
```

- No dependences.
- Iterations can be executed in parallel.

Data Dependence in Loops (cont.)

Parallelism often occurs in loops.

```
for(i=0; i<100; i++) {
    a[i] = i;
    b[i] = 2*i;
}
```

Iterations and statements can be executed in parallel.

Data Dependence in Loops (cont.)

Parallelism often occurs in loops.

```
for( i=1; i<100; i++ )
        a[i] = f(a[i-1]);
```

- Dependence between a[i] and a[i-1].
- Loop iterations are not parallelizable.

Data Dependence in Loops (cont.)

Parallelism often occurs in loops.

```
for( i=1; i<100; i++ )
        a[i] = f(a[i-1]);
```

- Dependence between a[i] and a[i-1].
- Loop iterations are not parallelizable.

Data Dependence in Loops (cont.)

- **Loop-Carried Dependence**
  - A loop-carried dependence is a dependence that is present only if the statements occur in two different instances of a loop
  - Otherwise, we call it a loop-independent dependence
  - Loop-carried dependences limit loop iteration parallelization

Data Dependence in Loops (cont.)

- **Loop-Carried Dependence**

```
for(i=1; i<100; i++ )
        for(j=1; j<100; j++ )
                a[i][j] = f(a[i][j-1]);
```

- Loop-independent dependence on i.
- Loop-carried dependence on j.
- Outer loop can be parallelized, inner loop cannot.

Data Dependence in Loops (cont.)

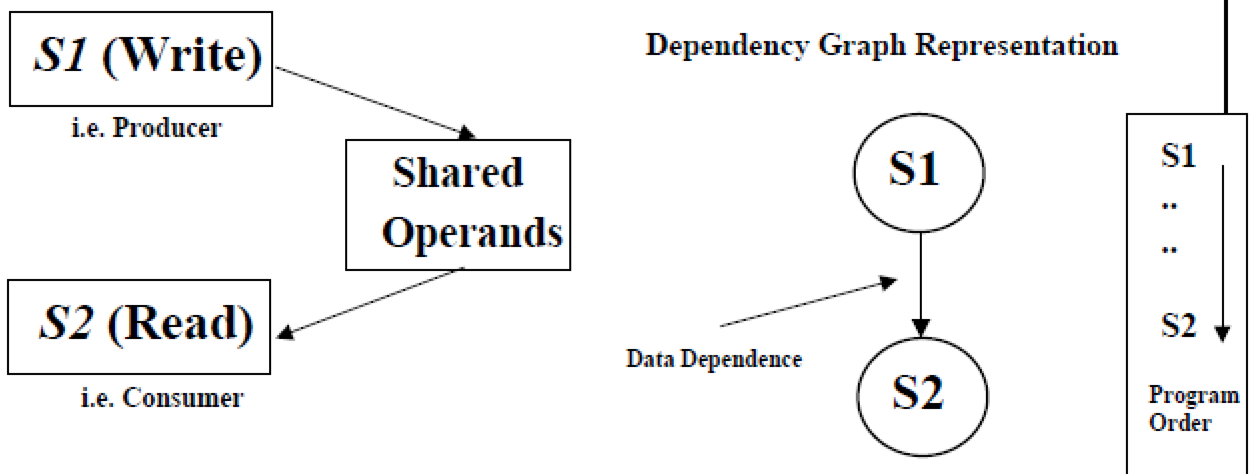
- **Loop-Carried Dependence**

```
for( j=1; j<100; j++ )
        for( i=1; i<100; i++ )
                a[i][j] = f(a[i][j-1]);
```

- Inner loop can be parallelized, outer loop cannot.
- Less desirable situation (finer-grain parallelism).
- Loop interchange is sometimes possible.

(True) Data (or Flow) Dependence

- **Assume task S2 follows task S1 in sequential program order**
- **Task S1 produces one or more results used by task S2,**
  - Then task S2 is said to be data dependent on task S1
- **Changing the relative execution order of tasks S1, S2 in the parallel program violates this data dependence and results in incorrect execution.**



Task S2 is data dependent on task S1