CSE 490/590 Computer Architecture

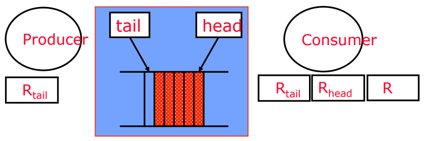
Synchronization and Consistency I

Steve Ko
Computer Sciences and Engineering
University at Buffalo

Last time…

- Multithreading executes instructions from different threads
- Coarse-grained multithreading switches threads on cache misses
- Most of the OoO superscalar units are idle.
- SMT utilizes most of the circuitry already present.
- Levels of multithreading
  - OoO superscalar
  - Fine-grained
  - Coarse-grained
  - Multiprocessing
  - SMT

A Producer-Consumer Example



Producer posting Item x:
```
     Load R_tail, (tail)
     Store (R_tail), x
     R_tail=R_tail+1
     Store (tail), R_tail
```

Consumer:
```
       Load R_head, (head)
spin:  Load R_tail, (tail)
       if R_head==R_tail goto spin
       Load R, (R_head)
       R_head=R_head+1
       Store (head), R_head
       process(R)
```

The program is written assuming instructions are executed in order.

Problems?

A Producer-Consumer Example

continued

Producer posting Item x:
```
     Load R_tail, (tail)
1    Store (R_tail), x
     R_tail=R_tail+1
2    Store (tail), R_tail
```

Consumer:
```
       Load R_head, (head)
spin:  Load R_tail, (tail)        3
       if R_head==R_tail goto spin
       Load R, (R_head)           4
       R_head=R_head+1
       Store (head), R_head
       process(R)
```

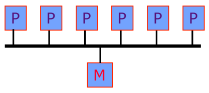
Can the tail pointer get updated before the item x is stored?

Programmer assumes that if 3 happens after 2, then 4 happens after 1.

Problem sequences are:
2, 3, 4, 1
4, 1, 2, 3

Sequential Consistency

A Memory Model



" A system is *sequentially consistent* if the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in the order specified by the program"
Leslie Lamport

Sequential Consistency =
arbitrary *order-preserving interleaving* of memory references of sequential programs

Sequential Consistency

Sequential concurrent tasks: T1, T2
Shared variables: X, Y (initially X = 0, Y = 10)

T1:
```
    Store (X), 1   (X = 1)
    Store (Y), 11  (Y = 11)
```

T2:
```
    Load R1, (Y)
    Store (Y'), R1  (Y'= Y)
    Load R2, (X)
    Store (X'), R2  (X'= X)
```

what are the legitimate answers for X' and Y' ?

(X',Y') ε {(1,11), (0,10), (1,10), (0,11)} ?

C 1

Sequential Consistency

Sequential consistency imposes more memory ordering constraints than those imposed by uniprocessor program dependencies ($\longrightarrow$)

What are these in our example ?

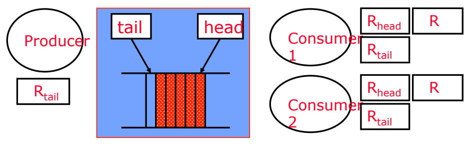
```
T1:                                T2:
  Store (X), 1   (X = 1)             Load R1, (Y)
  Store (Y), 11  (Y = 11)            Store (Y'), R1  (Y'= Y)
                                     Load R2, (X)
                                     Store (X'), R2  (X'= X)
```

$\longrightarrow$ additional SC requirements

Does (can) a system with caches or out-of-order execution capability provide a *sequentially consistent* view of the memory ?

more on this later

Multiple Consumer Example



Producer posting Item x:

```
    Load Rtail, (tail)
    Store (Rtail), x
    Rtail=Rtail+1
    Store (tail), Rtail
```

Consumer:

```
spin: Load Rhead, (head)
      Load Rtail, (tail)
      if Rhead==Rtail goto spin
      Load R, (Rhead)
      Rhead=Rhead+1
      Store (head), Rhead
      process(R)
```

Critical section: Needs to be executed atomically by one consumer ⇒ locks

What is wrong with this code?

Locks or Semaphores

E. W. Dijkstra, 1965

A *semaphore* is a non-negative integer, with the following operations:

P(s): *if s>0, decrement s by 1, otherwise wait*

V(s): *increment s by 1 and wake up one of the waiting processes*

P's and V's must be executed atomically, i.e., without
- *interruptions* or
- *interleaved accesses to s* by other processors

```
Process i
    P(s)
        <critical section>
    V(s)
```

initial value of s determines the maximum no. of processes in the critical section

Implementation of Semaphores

Semaphores (mutual exclusion) can be implemented using ordinary Load and Store instructions in the Sequential Consistency memory model. However, protocols for mutual exclusion are difficult to design...

Simpler solution:
atomic read-modify-write instructions

Examples: *m is a memory location, R is a register*

```
Test&Set (m), R:
    R ← M[m];
    if R==0 then
        M[m] ← 1;
```

```
Fetch&Add (m), RV, R:
    R ← M[m];
    M[m] ← R + RV;
```

```
Swap (m), R:
    Rt ← M[m];
    M[m] ← R;
    R ← Rt;
```

CSE 490/590 Administrivia

- Quiz 2 (Friday 4/8): After midterm until today
- Project 1 regrading
  - Email and talk to both me and Jangyoung
- Project 2 revision up soon (with some clarification)
  - Always email me and the TAs together for project-related questions
  - 5th (define-your-own) deadline this Wed

Acknowledgements

- These slides heavily contain material developed and copyright by
  - Krste Asanovic (MIT/UCB)
  - David Patterson (UCB)
- And also by:
  - Arvind (MIT)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)

- MIT material derived from course 6.823
- UCB material derived from course CS252