Computer Systems
Organization

CSE 521/560M
Lecture 22
Prof. Patrick Crowley

Plan for Today

• Homework 4 reminder
• Questions
• Today’s discussion
Shared Memory Example

Initially all pointers = null, all integers = 0.

D1

while (there are more tasks) {
    Task = GetFromFreeList();
    Task → Data = ...;
    insert Task in task queue
}
Head = head of task queue;

D2, D3, ..., Dn

while (MyTask == null) {
    Begin Critical Section
    if (Head != null) {
        MyTask = Head;
        Head = Head → Next;
    }
    End Critical Section
}
... = MyTask → Data;

What is expected of the memory system here?

Memory Consistency Models

- Provides a formal specification of how the memory system will appear to the programmer in a shared-memory multiprocessor
- A consistency model places restrictions on the values that can be returned by a read
  - a read should return the value of the “last” write to the same memory location
- In uniprocessors, “last” is precisely defined by program order (i.e., the order in which memory operations appear in the program)
- Multiprocessors, since there are multiple programs executing, have no single program to determine order
Sequential Consistency

- Extends the notion of program order to multiprocessors
- Requires that all memory operations appear to complete one at a time, with the operations of a single processor occur in program order

Shared Memory Example

Initially all pointers = null, all integers = 0.

P1

while (there are more tasks) {
    Task = GetFromFreeList();
    Task @ Data = ...;
    insert Task in task queue
}
    Head = head of task queue;

P2, P3, ..., Pn

while (MyTask == null) {
    Begin Critical Section
    if (Head != null) {
        MyTask = Head;
        Head = Head -> Next;
    }
    End Critical Section
    ... = MyTask -> Data;

Sequential consistency requires that each read of the data field will return the new values written by P1.
The Problem with Sequential Consistency

- Enforces a strict ordering of memory operations
  - Conflicts with many uniprocessor hardware and compiler optimizations
- Uniprocessors frequently only maintain data and control dependences (allow overlapping and reordering of memory operations)
  - Code motion, loop transformations
  - Write buffers (bypassing and forwarding)
  - Lockup-free caches

The Benefit of Sequential Consistency

- Intuitive programming model
- In uniprocessors, allows a wide range of performance optimizations
Sequential Consistency Semantics

Initially $Flag1 = Flag2 = 0$

- **P1**
  - $Flag1 = 1$
  - *if* $(Flag2 = 0)$
    - *critical section*

- **P2**
  - $Flag2 = 1$
  - *if* $(Flag1 = 0)$
    - *critical section*

(a)

Initially $A = B = 0$

- **P1**
  - $A = 1$
  - *if* $(A = 0)$

- **P2**
  - $B = 1$
  - *if* $(B = 1)$
    - register1 = A

(b)

(a) Illustrates meaning of ordering at a processor,
(b) illustrates meaning of ordering between processors

Write Buffers with Bypassing

Important: maintain program order between a write and a following read
Overlapped Writes

Important: maintain program order between two write operations

Lockup-Free Caches

Important: maintain program order between two read operations
Relaxed Consistency Models

- Seek to improve performance by *not* supporting certain orderings
  - $W \to R$
  - $W \to W$
  - $R \to R$ or $W$
- Many models have been proposed

Weak Ordering

- Classifies memory operations into: *data* and *synchronization* operations
- Idea: order synchronization operations
  - Intuition: once the synchronization operations are ordered, then data operations can proceed arbitrarily
- Simple implementation: each processor keeps an operation counter
  - Increment on issue and decrement on completion
  - Do not issue a synchronization operation until the counter is zero
  - Also, no operations are issued until the synch operation completes
Assignment

• Readings
  – For Wednesday
    • None
  – For Monday
    • H&P: 8.1-8.2