
EEC 581 Computer Architecture

Lec 11 – Synchronization and Memory Consistency Models (4.5 & 4.6)

Chansu Yu
Electrical and Computer Engineering
Cleveland State University

Acknowledgement ...

- Part of class notes are from
 - David Patterson
 - Electrical Engineering and Computer Sciences
 - University of California, Berkeley
 - <http://www.eecs.berkeley.edu/~pattsn>
 - <http://www-inst.eecs.berkeley.edu/~cs252>

Review

- Caches contain all information on state of cached memory blocks
- Snooping cache over shared medium for smaller MP by invalidating other cached copies on write
- Sharing cached data \Rightarrow Coherence (values returned by a read), Consistency (when a written value will be returned by a read)

11/15/2007

3

Outline

- Review
- Synchronization
- Relaxed Consistency Models
- Fallacies and Pitfalls
- Cautionary Tale
- Conclusion

11/15/2007

4

Synchronization

- **Why Synchronize?**

- Need to know when it is safe for different processes to use shared data

- Example

```
lock(L);          /* wait until L is free */
critical section;
unlock(L);
```

- **Issues for Synchronization:**

- Uninterruptable instruction to fetch and update memory (atomic operation)
- For large scale MPs, synchronization can be a bottleneck; techniques to reduce contention and latency of synchronization

11/15/2007

5

Software Solution (?)

- **Procedure lock(L)**

```
wait: lw   R1, L           ;
      bne  R1, #0, wait    ; if L=#1 (locked), wait
      sw   L, #1           ; now, 'lock' it myself
```

- **Procedure lock(L) is wrong!**

- Is it wrong in MP or wrong in both UP & MP?

- **Required capability**

- Atomically read and modify a memory location

11/15/2007

6

Basic Hardware Primitive

- **Atomic instruction**
test&set R1, X ; R1=X and X=1 (locked)
; if X=1, R1=1 & X=1
; if X=0, R1=0 & X=1
; Thus, X=1 and R1=previous value of X
- **Procedure lock(L)**
wait: test&set R1, L ; L=1 and R1=prev. L
bne R1, #0, wait ; if L=#1 (locked), wait
- **Correct, but busy-waiting (spin-waiting)**

11/15/2007

7

Other Basic Hardware Primitives

- **Alternatives: Swap, Fetch-and-increment**
- **Recent alternatives: a pair of instructions**
 - Load-locked (ll) and store-conditional (sc)
- **Procedure lock(L)**
wait: ll R1, L ;
sc #1, L ; L=#1 if no intervening
; write after 'll'
bne R1, #0, wait ; if L=#1 (locked), wait
- **Correct, but busy-waiting (spin-waiting)**

11/15/2007

8

Busy-Waiting

- **Spin locks: processor continuously tries to acquire, spinning around a loop trying to get the lock**
- **What about MP with cache coherency?**
 - Want to spin on cache copy to avoid full memory latency
 - Likely to get cache hits for such variables
- **Problem: exchange includes a write, which invalidates all other copies; this generates considerable bus traffic**

11/15/2007

9

Busy-Waiting

- **Example**
 - Assume P1 locked the variable and in the critical section
 - Assume also P2 and P3 want to get into
 - P2 and P3 execute procedure lock(L), which means “write hit (miss)” repeatedly, leading to “cache invalidate” with each other

11/15/2007

10

Better Solutions

- Procedure lock(L)
wait: test&set R1, L ; L=1 and R1=prev. L
 bne R1, #0, wait ; if L=#1 (locked), wait
- Procedure lock(L)
wait: lw R1, L
 bne R1, #0, wait
 test&set R1, L ; L=1 and R1=prev. L
 bne R1, #0, wait ; if L=#1 (locked), wait
- Correct, and not busy-waiting

11/15/2007

11

Better Solutions

- Procedure lock(L)
wait: ll R1, L ;
 sc #1, L ; L=#1 if no intervening write after 'll'
 bne R1, #0, wait ; if L=#1 (locked), wait
- Procedure lock(L)
wait: ll R1, L
 bne R1, #0, wait
 sc #1, L ; L=#1 if no intervening write after 'll'
 bne R1, #0, wait ; if L=#1 (locked), wait
- Correct, and not busy-waiting

11/15/2007

12

Outline

- Review
- Synchronization
- Relaxed Consistency Models
- Fallacies and Pitfalls
- Cautionary Tale
- Conclusion

11/15/2007

13

Another MP Issue: Memory Consistency Models

- **What is consistency? When must a processor see the new value?**

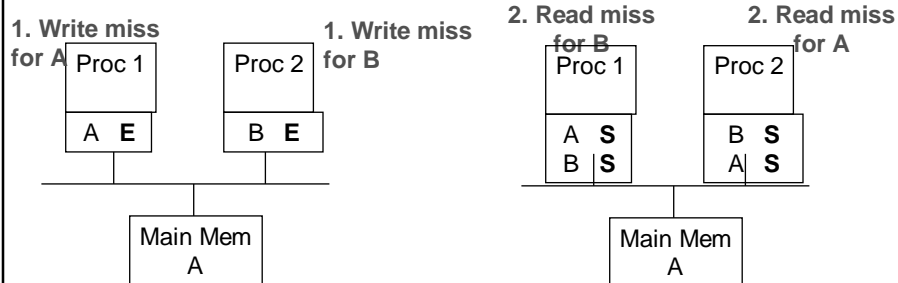
P1: A = 0;	P2: B = 0;
.....
Read B	Read A
.....
A = 1;	B = 1;
L1: if (B == 0) ...	L2: if (A == 0) ...

- **Impossible for both if statements L1 & L2 to be true?**
 - What if write invalidate is delayed & processor continues?

11/15/2007

14

Another MP Issue: Memory Consistency Models



- What is consistency? When must a processor see the new value? e.g., seems that

P1: A = 0;

....
Read B

....
A = 1;

L1: if (B == 0) ...

P2: B = 0;

....
Read A

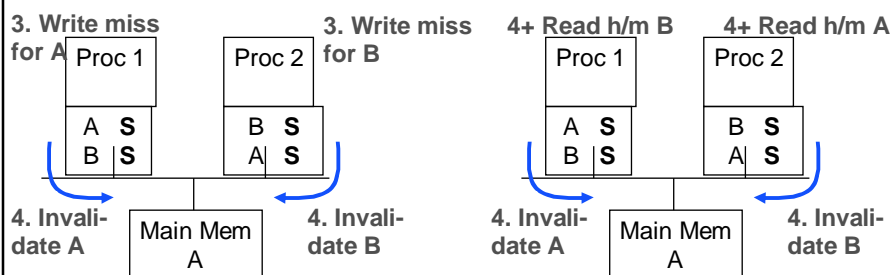
....
B = 1;

L2: if (A == 0) ...

11/15/2007

15

Another MP Issue: Memory Consistency Models



- What is consistency? When must a processor see the new value? e.g., seems that

P1: A = 0;

....
Read B

....
A = 1;

L1: if (B == 0) ...

P2: B = 0;

....
Read A

....
B = 1;

L2: if (A == 0) ...

11/15/2007

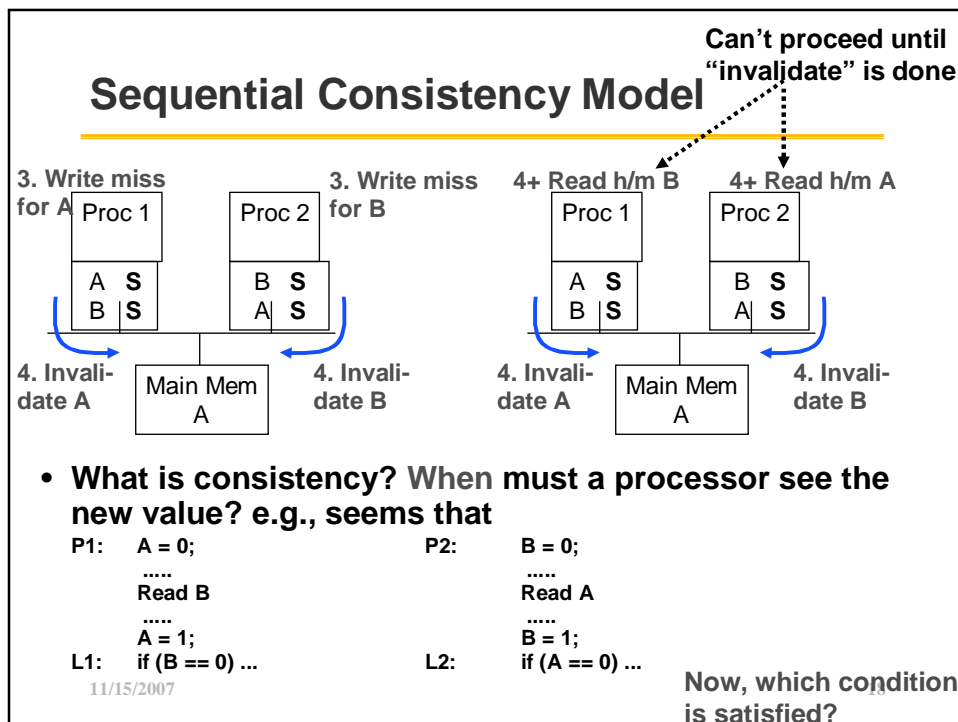
16

Another MP Issue: Memory Consistency Models

- **Memory consistency models:**
what are the rules for such cases?
- **Sequential consistency (SC):** result of any execution is the same as if the accesses of each processor were kept in order and the accesses among different processors were interleaved \Rightarrow assignments before ifs above
 - SC: delay all memory accesses until all invalidates done

11/15/2007

17



Memory Consistency Model

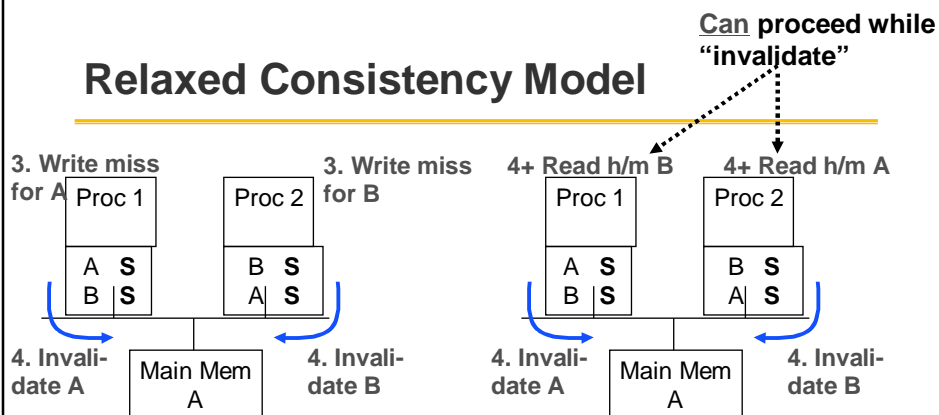
- **Faster execution to “sequential consistency”**
- **Not an issue for most programs; they are synchronized**
 - A program is synchronized if all access to shared data are ordered by synchronization operations


```
write (x)
...
release (s) {unlock}
...
acquire (s) {lock}
...
read(x)
```
- **Only those programs willing to be nondeterministic are not synchronized: “data race”: outcome f(proc. speed)**

11/15/2007

19

Relaxed Consistency Model



- **What is consistency? When must a processor see the new value? e.g., seems that**

P1: A = 0;
.....
Read B
.....
A = 1;
L1: if (B == 0) ...

P2: B = 0;
.....
Read A
.....
B = 1;
L2: if (A == 0) ...

Performance improves with OOO processors.

But, unpredictable results...

11/15/2007

20

Relaxed Consistency Model

3. Write miss for A

4. Invalidate A

3. Write miss for B

4. Invalidate B

4+ Read h/m B

4. Invalidate A

4+ Read h/m A

4. Invalidate B

Can proceed while "invalidate"

- **What is consistency? When must a processor see the new value? e.g., seems that**

<p>P1: A = 0; Read B A = 1; L1: if (B == 0) ...</p>	<p>P2: B = 0; Read A B = 1; ←acquire(s) L2: if (A == 0) ... ←release(s)</p>	<p>} RC</p> <p>} SC</p>
---	---	-------------------------

11/15/2007
21

Memory Consistency Model

- **Faster execution to “sequential consistency”**
- **Not an issue for most programs; they are synchronized**
 - A program is synchronized if all access to shared data are ordered by synchronization operations

```

write (x)
...
release (s) {unlock}
...
acquire (s) {lock}
...
read(x)

```
- **Acquire (read shared variables)**
 - Delay memory accesses that follow the acquire operation until the acquire completes
- **Release (write the shared variables)**
 - Grant access to the new values of data before it

11/15/2007
22

Relaxed Consistency Models

- **Key idea: allow reads and writes to complete out of order, but to use synchronization operations to enforce ordering, so that a synchronized program behaves as if the processor were sequentially consistent**
 - By relaxing orderings, may obtain performance advantages
 - Also specifies range of legal compiler optimizations on shared data
 - Unless synchronization points are clearly defined and programs are synchronized, compiler could not interchange read and write of 2 shared data items because might affect the semantics of the program

11/15/2007

23

Relaxed Consistency Models

- **SC requires maintaining all four possible orderings,**
 - $R \rightarrow R$,
 - $R \rightarrow W$,
 - $W \rightarrow R$ (all writes completed before next read) ,
 - $W \rightarrow W$ (all writes completed before next write)
- **Relaxed model relax some of them**
 - Relaxing $W \rightarrow R$: Because retains ordering among writes, many programs that operate under sequential consistency operate under this model, without additional synchronization.
 - Relaxing $W \rightarrow W$
 - Relaxing $R \rightarrow W$ and $R \rightarrow R$ (PowerPC, Alpha)

11/15/2007

24

Mark Hill observation

- **Instead, use speculation to hide latency from strict consistency model**
 - If processor receives invalidation for memory reference before it is committed, processor uses speculation recovery to back out computation and restart with invalidated memory reference
- 1. **Aggressive implementation of sequential consistency or processor consistency gains most of advantage of more relaxed models**
- 2. **Implementation adds little to implementation cost of speculative processor**
- 3. **Allows the programmer to reason using the simpler programming models**

11/15/2007

25

Answers to 1995 Questions about Parallelism

- In the 1995 edition of this text, we concluded the chapter with a discussion of two then current controversial issues.
 1. **What architecture would very large scale, microprocessor-based multiprocessors use?**
 2. **What was the role for multiprocessing in the future of microprocessor architecture?**

Answer 1. Large scale multiprocessors did not become a major and growing market ⇒ clusters of single microprocessors or moderate SMPs

Answer 2. Astonishingly clear. For at least for the next 5 years, future MPU performance comes from the exploitation of TLP through multicore processors vs. exploiting more ILP

11/15/2007

26

Cautionary Tale

- **Key to success of birth and development of ILP in 1980s and 1990s was software in the form of optimizing compilers that could exploit ILP**
- **Similarly, successful exploitation of TLP will depend as much on the development of suitable software systems as it will on the contributions of computer architects**
- **Given the slow progress on parallel software in the past 30+ years, it is likely that exploiting TLP broadly will remain challenging for years to come**

11/15/2007

27

And in Conclusion ...

- **Snooping and Directory Protocols similar; bus makes snooping easier because of broadcast (snooping \Rightarrow uniform memory access)**
- **Directory has extra data structure to keep track of state of all cache blocks**
- **Distributing directory**
 - \Rightarrow **scalable shared address multiprocessor**
 - \Rightarrow **Cache coherent, Non uniform memory access**
- **MPs are highly effective for multiprogrammed workloads**
- **MPs proved effective for intensive commercial workloads, such as OLTP (assuming enough I/O to be CPU-limited), DSS applications (where query optimization is critical), and large-scale, web searching applications**

11/15/2007

28