CS/ECE 757: ADVANCED COMPUTER ARCHITECTURE II
COMPUTER SCIENCES DEPARTMENT
UNIVERSITY OF WISCONSIN—MADISON

Prof. Mark D. Hill

Midterm Examination I
In-Class
Wednesday, February 26, 2014
Weight: 25%

1:15 minutes.

CLOSED BOOK, etc., but one cheat sheet allowed (two-sided 8.5x11 page).

The exam is *two-sided* and has **EIGHT** pages, including two blank pages at the end.

Plan your time carefully, since some problems are longer than others.

NAME: ______________________KEY________________________

ID# ____

Problem Number	**Maximum Points**	**Actual Points**
1	12	
2	12	
3	12	
4	12	
5	12	
Total	60	

Problem 1: Parallel Programming (12 points)

(a) OpenMP allows loop-level parallelism where parallel loop iterations are ***statically or dynamically scheduled***. What is static scheduling? What is dynamic scheduling? Under what circumstances is static scheduling preferred to dynamic? Under what circumstances is dynamic preferred to static?

With static scheduling the OpenMP compiler and runtime assign work to threads by dividing up the loop iterations evenly, e.g., N/P iterations per thread for N iterations with P threads.

With dynamically scheduling, one or more iterations get assigned to a thread at runtime when a thread finishes its prior iterations. This gets implemented in the runtime with some kind of work queue that usually adds communication and synchronization overhead for getting more work.

Static scheduling is preferred to dynamic when the work per iteration is similar, so that a static assignment balances work sufficiently and avoids work queue overheads.

Dynamic scheduling is preferred to static when the work per iteration is unpredictable, so that work queue overheads are worth paying to achieve better load balance.

(b) ***Parallel Random Access Machine (PRAM)*** provides a model of computation. What is the PRAM model? Explain three (or more) different, important ways that PRAM is an inaccurate model of a shared-memory multiprocessor using snooping cache coherence.

A PRAM uses:

- *An infinite number of processors*
- *That operate in lock step*
- *Making all accesses to a shared memory*
- *At unit latency with arbitrarily-high bandwidth.*

PRAM difference an SMP, as an SMP has:

- *Finite processors*
- *That make unpredictable progress (requiring synchronization)*
- *Caches that are sensitive to locality and made invisible by coherence*
- *Non-unit memory latencies and finite bus bandwidth.*

Problem 2: Consistency (12 points)

Assume all memory and register values are initially zero. Each execution of the following code leaves registers (r1, r2, r3, r4) with a set of values (?, ?, ?, ?), but different executions leave different values.

```
Processor P1        Processor P2
S1: x = 2;          S2: y = 3;
L1: r1 = x;         L3: r3 = y;
L2: r2 = y;         L4: r4 = x;
```

(a) For all alternative executions allowed by ***sequential consistency (SC),*** what are the final registers values? One value set is given.

```
(r1, r2, r3, r4) = (2, 3, 3, 2)  S1 before L4 and S2 before L2
(r1, r2, r3, r4) = (2, 0, 3, 2)  S1 before L4 and L2 before S2
(r1, r2, r3, r4) = (2, 3, 3, 0)  L4 before L1 and S2 before L2
```

Note that program order from S1 to L1 always requires that r1 = 2.

Note that program order from S2 to L3 always requires that r3 = 3.

(b) For all alternative executions allowed by ***total store order (TSO),*** what are the final registers values?

```
(r1, r2, r3, r4) = (2, 3, 3, 2)  Like SC: S1 before L4 and S2 before L2
(r1, r2, r3, r4) = (2, 0, 3, 2)  Like SC: S1 before L4 and L2 before S2
(r1, r2, r3, r4) = (2, 3, 3, 0)  Like SC: L4 before L1 and S2 before L2
(r1, r2, r3, r4) = (2, 0, 3, 0)  New for TSO
```

Note that program order from S1 to L1 always requires that r1 = 2.

Note that program order from S2 to L3 always requires that r3 = 3.

(c) What is ***Sequential Consistency for Data Race Free (SC for DRF)?*** How is it like a strong memory (consistency) model like SC? How is like a relaxed memory (consistency) model like XC?

SC for DRF provides programmers for the illusion of SC for programs that are data-race-free, but is complex or undefined with programs have data races.

SC for DRF provides programmers to the strong SC model when data are avoided.

SC for DRF allows the system to re-order accesses—like a relaxed model—provide that don't cross synchronization.

Problem 3: Coherence (12 points)

(a) MESI coherence protocols includes the ***Exclusive (E)*** in addition to Modified (M), Shared (S), and Invalid (I). When does an MESI protocol enter the E state? How does it leave the E state? What benefit does E state provide? What are E-state drawbacks?

MESI enters the E state when a core requests an S copy & no other processor has any copy.

A core leaves the E state four ways:

- *Go to M when the same core does a write,*
- *Go to I when the same core does a replacement,*
- *Go to S when another core seeks a S copy, and*
- *Go to I when another core seeks a M copy.*

E state allows read misses that are soon followed by a write from the same core to use one coherence transaction instead or two (e.g., for private data).

E state requires a mechanism to determine that no one has a block and adds the complexity of more coherence states—one stable and more transient—and transitions.

(b) To perform an atomic read–modify–write instruction (e.g., test-and-set), must a core always communicate with the other cores? Why or why not?

Not true. If a core has the block containing lock L in state M, it can perform a test-and-set L from its local cache with no further communication.

(c) Discuss possible races with cache ***writebacks*** (M→I) depending on whether a systems has (i) Atomic Requests and Atomic Transactions or (ii) just Atomic Transactions. Hint: recall transient state ***MI^A***.

With (i), a core issues a PUTM, immediately sends the data to memory, and immediately transitions to I. (Table 7.5). No races are possible.

With (ii), a core issues a PUTM, transitions to MI^A and awaits its PUTM. In the common, case it sees its own PUTM and then sends the data to memory and transitions to I. It is also, possible that before it sees its own PUTM, it sees another core's GETS (or GETM). For GETS (GETM), it provides the data and transitions to II^A until it sees its own GETM and transitions to I.

Problem 4: From Your Reader (12 points)

(a) Hill and Marty [Computer 2008] provide three models for multicore chips. What are they and how do they differ?

The symmetric model requires that all cores use the same resources whose total equals the chip resources.

The asymmetric model allows one core with use more resources than the other equal cores where the total of both core types equals the chip resources.

The dynamic model allows one "core" to use up to all of chip resources or, at a different time, spend all the chip resources on equal base cores.

(b) Compare and contrast the data parallel programming model advocated by Hillis and Steele [CACM 1986] with the shared memory programming model?

Both models assume a shared address space.

The data parallel model assumes the all work proceeds in lock step at if either (a) there is one PC for all data items or (b) there is a thread per datum and an implicit barrier between each step.

The shared memory model assumes that threads proceed asynchronously. To restore needed order, programmers must add synchronization: locks, condition variables, barriers, etc.

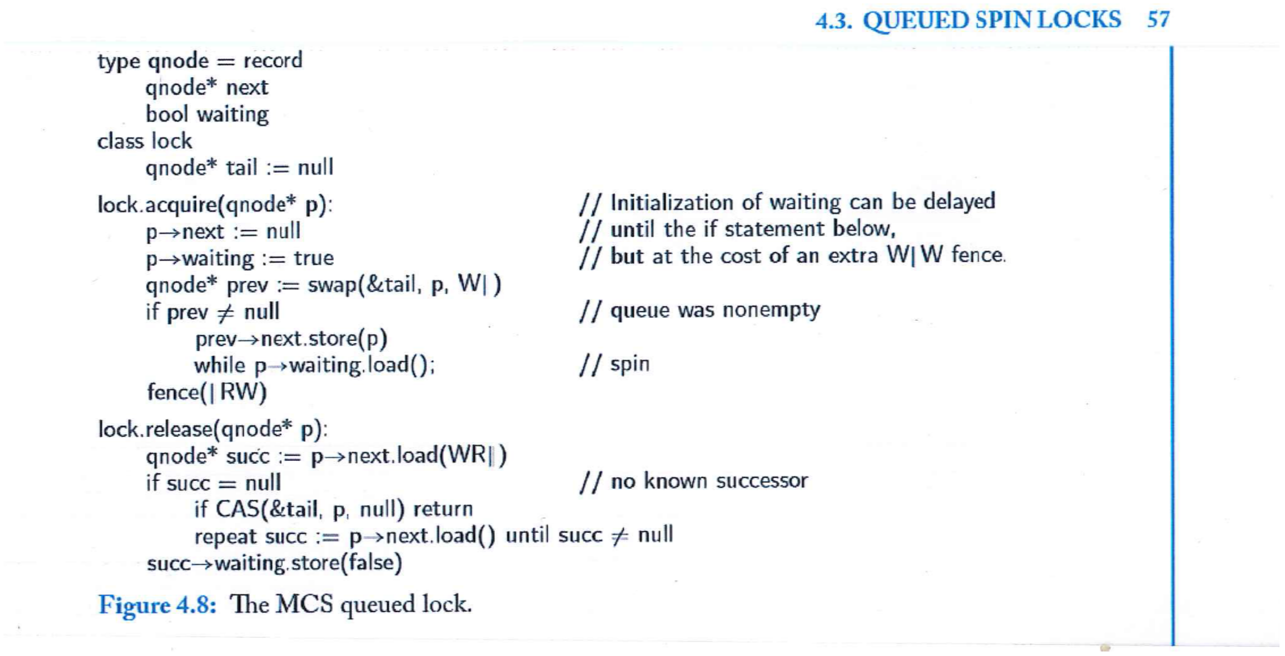
(c) Charlesworth [Micro 1998] describes the Sun E10000. What interconnect(s) does the E10000 use to send coherence requests and receive data responses? How does coherence work since it doesn't use a single shared bus?

The E10000 broadcasts coherence request on four logical bus interleaved by address that are implement with trees.

The E10000 sends data responses point-to-point via a data crossbar.

The four buses provide a total order for snooping. A request that wins arbitration at cycle i is after requests at cycle j < i and before requests at cycle k > i. The one to four request on the same cycle can to arbitrarily ordered (e.g., by bus number) because they are to different, non-conflicting addresses. The order of data messages has no effect on coherence order.

Problem 5: MCS Queued Lock (12 points)



4.3. QUEUED SPIN LOCKS 57

```
type qnode = record
    qnode* next
    bool waiting
class lock
    qnode* tail := null
lock.acquire(qnode* p):                    // Initialization of waiting can be delayed
    p→next := null                         // until the if statement below,
    p→waiting := true                      // but at the cost of an extra W|W fence.
    qnode* prev := swap(&tail, p, W| )
    if prev ≠ null                         // queue was nonempty
        prev→next.store(p)
        while p→waiting.load();            // spin
    fence(| RW)
lock.release(qnode* p):
    qnode* succ := p→next.load(WR| )
    if succ = null                         // no known successor
        if CAS(&tail, p, null) return
        repeat succ := p→next.load() until succ ≠ null
    succ→waiting.store(false)
```

Figure 4.8: The MCS queued lock.

(a) What are the advantages and disadvantages of an ***MCS queue-based lock*** (pseudo-code above) vs. a test-and-test-and-set lock?

MCS locks have requestors provide a queue node (-), may be slower if the lock is free (-), allows local spinning even when the lock is handed off among others (+) and provide first-come-first-serve handoff (+ or -).

Test-and-test-and-set locks require no queue node (+), are fast if the lock is free (+), allows local spinning when the lock is held but cause traffic when the lock is handed off among others (-) & do not guarantee who gets the lock among competing requestors (+/-).

(b) An initially free lock L will start with memory address L having this initially null ***linked list***:

`L [null]`

1. Let thread 1 provide a qnode at address A to obtain lock L. Draw the new linked list state (including pointers and values within the qnode).

   `L [ A ] ;  A [held | null]`

2. Let thread 2 provide a qnode at address B to queue for lock L. Draw the new linked list state.

   `L [ B ] ; A [held | B] ; B [held | null]`

3. Let thread 1 release lock L and thread 2 obtain it. Draw the new linked list state.

   `L [ B ] ; B [held | null]`

Scratch Sheet 1 of 2 (in case you need additional space for some of your answers)

Scratch Sheet 2 of 2 (in case you need additional space for some of your answers)