EECS 570 Midterm Exam  
Winter 2012

Name: ________________________________  unique name: ________________

Sign the honor code:

I have neither given nor received aid on this exam nor observed anyone else doing so.

____________________________________

Scores:

<table>
<thead>
<tr>
<th>#</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>/ 20</td>
</tr>
<tr>
<td>2</td>
<td>/ 16</td>
</tr>
<tr>
<td>3</td>
<td>/ 22</td>
</tr>
<tr>
<td>4</td>
<td>/ 22</td>
</tr>
<tr>
<td>5</td>
<td>/ 20</td>
</tr>
<tr>
<td>Total</td>
<td>/ 100</td>
</tr>
</tbody>
</table>

NOTES:

- Closed book, closed notes.
- Calculators are allowed, but no PDAs, Portables, Cell phones, etc.
- Don’t spend too much time on any one problem.
- You have about 90 minutes for the exam (avg. 18 minutes per problem).
- There are 10 pages in the exam (including this one). Please ensure you have all pages.
- Be sure to show work and explain what you’ve done when asked to do so.
1) Short Answer [20 points]

a) Argue briefly why the “power wall” has led to a shift towards multicore processors. [4 points]

b) Of online transaction processing (OLTP) and scientific computing workloads, which is more likely to benefit from hardware mechanisms that allow multiple outstanding misses from each thread (e.g., an advanced load-store queue)? Explain. [4 points]

c) Contrast Eager and Lazy version management in transactional memory systems. [4 points]
d) Inclusion can lead to pathologically poor replacement decisions. Give an example of an access sequence, at most 8 accesses in length, which, if repeated over and over, will result in more misses under an inclusive hierarchy than a non-inclusive hierarchy. Assume a two-way L1 cache and 4-way L2 cache. Indicate the sequence of addresses to be accessed, using letters to indicate distinct addresses. Assume LRU replacement and that all addresses map to a single set in each cache level. [4 points]

e) State one advantage of message passing over shared memory. [4 points]
2) **Speculative Lock Elision [16 points]**

Consider each of the following programs, explain whether speculative lock elision will result in higher performance, lower performance, or about the same performance. Explain your answer.

a)  An application where several threads are adding elements to a large hash table that is protected by a single global lock. [4 points]

b)  An application where several threads are each incrementing a single shared counter as fast as they can. The counter is protected by a lock. [4 points]

c)  A system running a multi-programmed workload of several benchmarks from the (individually single-threaded) SPEC CPU integer suite. [4 points]

d)  A commercial program that makes frequent use of system calls in the operating system. The operating system data structures are protected by a single global lock. [4 points]
3) Coherence & Consistency [22 points]

a) Draw a picture of a directory coherence protocol exchange that can result in a deadlock if all cache coherence messages travel in a single protocol lane. Use circles to indicate nodes that participate in the exchange and labeled arrows to indicate messages. Mark the directory node with an “H” (for Home) and use appropriate labels for the other participating nodes (e.g., “R” for reader, “W” for writer, etc.) Indicate the initial state of the coherence protocol at each node (you can use any protocol variant, e.g., MSI, and whatever initial state is most convenient). Number the head and tail of each arrow to indicate the order that messages are sent and received in the scenario that leads to a deadlock.

Write a brief accompanying explanation that describes the scenario in enough detail to understand the diagram and indicates which message is blocking a node’s input queue and which message is “stuck” waiting behind it.

Finally, explain which message(s) must be switched to a different protocol lane to resolve the deadlock. [8 points]
b) Many recent proposals for cache coherence in chip multiprocessors make use of clean eviction messages (Put-S messages in Sorin et al). State an advantage and a disadvantage of using these messages. [6 points]

c) Assume the following program segments are executed on three processors of a multiprocessor machine. Initially before execution, all memory variables (A,B,u,v,w) are equal to 0. [8 points]

<table>
<thead>
<tr>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A=1</td>
<td>u = A</td>
<td>v = B</td>
</tr>
<tr>
<td>B = 1</td>
<td>w = A</td>
<td></td>
</tr>
</tbody>
</table>

What are the possible final states of u, v, and w under a sequentially consistent model?

What are the final states that are NOT possible under SC?
4) Synchronization [22 points]

a) Explain the advantage of a list-based lock over an array-based lock. [4 points]

b) Briefly define what it means for a parallel algorithm to be “wait free.” [4 points]

c) Can a wait-free augmented queue that supports “push,” “pop” and “peek” operations be implemented using only load, store, and fetch-and-add operations in a system with more than 2 processors? Informally explain the reasoning for your answer. [4 points]
d) Explain why the following lock implementation is incorrect. Describe briefly how it can be fixed (Note: it is not necessary to write corrected pseudo-code provided your explanation of how to fix the implementation is clear.) As a reminder, `fetch_and_store(A,B)` returns the memory value at location A, and atomically stores the value of B to memory. [10 points]

```c
1:   struct lock_node {
2:       lock_node * next;
3:       bool is_locked;
4:   }
5:   acquire(lock_node * lock, lock_node * my_node) {
6:       my_node->next = null;
7:       lock_node * pred = fetch_and_store(lock, my_node);
8:       if (pred != null) {
9:           my_node->is_locked = true;
10:          pred->next = my_node;
11:          while (!my_node->is_locked);
12:       }
13:   }
14:  release(lock_node * lock, lock_node * my_node) {
15:     if (my_node->next == null) {
16:         return; //no successor
17:     }
18:     my_node->next->is_locked = false;
19:  }
```
5) Destination Set Predictor [20 points]

One of the main advantages of Token Coherence is that it allows “performance protocols” that move tokens on the basis of predictions to try to optimize coherence transactions without the need to modify the underlying “correctness substrate” on which coherence is based. One such optimization, called “destination set prediction” tries to identify (1) for read requests, the identity of the likely owner to have last written the block and (2) for write requests, the identity of the most recent writer and set of sharers have read-only copies of the block.

Propose a design for a Destination Set Predictor. Argue why your design is likely to be accurate (i.e., predict correct destination sets) and require tractable storage (i.e., why it can be effective with reasonably sized prediction tables).

• Draw a diagram for your predictor (what state does it contain?)
• Describe the learning process (how are the predictor tables trained?)
• Describe the lookup and prediction process (when is it accessed, what is its output?)
• What happens upon a misprediction?
(additional space for question #5)