EECS 570 Midterm Exam - SOLUTIONS
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. x minutes per problem).
- There are x 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]

> Moore’s law continues to provide more transistors, but threshold, and thus supply voltages no longer scale down each generation. Hence, to leverage the additional transistors, designers reduce clock frequency and invest the power headroom to activate twice as many cores. Because of the quadratic relationship between power a voltage, a small voltage/frequency reduction allows a doubling of the number of transistors that switch each cycle, enabling overall throughput improvements from multicore.

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)? [4 points]

> Scientific applications. OLTP spends most of its time chasing pointers, and, hence, does not benefit from intra-thread memory level parallelism.

c) Contrast **Eager** and **Lazy** version management in transactional memory systems. [4 points]

> Eager version management stores transactional (speculative) memory values in place in the memory system, and keeps old (committed) values in a side data structure that is restored upon abort. Lazy keeps speculative values on the side and leaves committed values in place.
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 all addresses map to a single set in each cache level. [4 points]

\[ A B A C A D A E \]

e) State one advantage of message passing over shared memory. [4 points]

Message passing scales to larger systems.

Message passing makes communication a first-order aspect of programs, which typically leads to more efficient communication.
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]

   SLE improves performance. SLE elides acquisitions of the lock in the common case that inserts do not access the same hash table bucket. Hence, inserts can proceed in parallel.

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]

   SLE will likely worsen performance. The threads all conflict when trying to increment the shared counter, hence, speculation will nearly always fail, resulting in many rollbacks.

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

   SLE will have no effect, since the single-threaded programs contain no locks.

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]

   SLE improves performance. SLE elides acquisitions of the lock in the common case that operating system calls access different kernel data structures.
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]

**Advantage:** directory has precise sharer list, which avoids unnecessary invalidations. Also, certain race cases become simpler, as there is no possibility of a "stale" invalidation racing with a new read request.

**Disadvantage:** extra messages when private data is evicted.

c) Assume the following program segments are executed on three processors of a multiprocessor machine. Initially before execution, all memory variables 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?

- u = 0  v = 0  w = 0
- u = 0  v = 0  w = 1
- u = 0  v = 1  w = 0
- u = 0  v = 1  w = 1
- u = 1  v = 0  w = 0
- u = 1  v = 0  w = 1
- u = 1  v = 1  w = 0
- u = 1  v = 1  w = 1

What are the final states that are NOT possible under SC?

- u = 1  v = 1  w = 0
4) Synchronization [22 points]

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

In a list-based lock, the storage requirement is proportional to the number of nodes currently waiting for the lock, rather than the maximum number of nodes that might wait for the lock. In other words, the storage for the maximum number of waiting threads does not need to be allocated in advance.

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

Each operation has a finite bound on the number of steps the algorithm will execute before the operation completes. In other words, each thread is guaranteed to make progress on its task within a bounded number of steps.

c) Can a wait-free 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]

No. fetch&add has a consensus number of 1, but an augmented stack has a consensus number of infinity. Hence, Herlihy's impossibility results (Theorem 1) imply that fetch&add cannot implement a wait-free queue supporting a peek operation.
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.) [10 points]

```c
struct lock_node {
    lock_node * next;
    bool is_locked;
}

acquire(lock_node * lock, lock_node * my_node) {
    my_node->next = null;
    lock_node * pred = fetch_and_store(lock, my_node);
    if (pred != null) {
        my_node->is_locked = true;
        pred->next = my_node;
        while (my_node->is_locked);
    }
}

release(lock_node * lock, lock_node * my_node) {
    if (my_node->next == null) {
        return; //no successor
    }
    my_node->next->is_locked = false;
}
```

This code overlooks the possibility that a successor can add itself to my_node->next after the test on line 15. Within the body of the if statement (on line 16), the code must use a compare_and_swap operation to try to swap a null onto lock, in place of my_node. If the CAS succeeds, the lock has successfully been released. Otherwise, there is a race, and the release function must spin waiting until my_node->next becomes valid. Then, it can execute line 18 to release the lock to the next processor in line.
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 current owner 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)