EECS 570 Midterm Exam
Winter 2011

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>/ 12</td>
</tr>
<tr>
<td>3</td>
<td>/ 28</td>
</tr>
<tr>
<td>4</td>
<td>/ 20</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) Explain briefly why it might be advantageous to run applications on parallel computer
systems even if the application achieves only sub-linear speedup in the number of processors.
[4 points]

b) Of online transaction processing (OLTP) and decision support system (DSS) database
workloads, which is more likely to benefit from hardware mechanisms that accelerate
synchronization operations? [4 points]

c) State the conditions under which a transaction must be aborted/retried in a hardware
transactional memory system. [4 points]

d) Give an advantage and a disadvantage of maintaining inclusion in multi-level cache
hierarchies in a multiprocessor system. [8 points]
2) Applications [12 points]

Describe briefly three characteristics that distinguish typical commercial server application programs from scientific/HPC (High-Performance Computing) applications? [12 points]
3) **Synchronization [28 points]**

a) Write an assembler sequence using the compare-and-swap instruction (CAS) that increments a counter that is shared among multiple processors running on a shared memory multiprocessor.

Definition of the CAS instruction:

```
cas [mem], %r1, %r2
```

The content of the memory word at address `[mem]` is compared to the value in register `%r1`. If they are equal, the memory word at address `[mem]` and the value of register `%r2` are swapped. If `%r1` and `[mem]` are not equal, `[mem]` is loaded into `%r2` and the memory state remains unchanged. *(Thanks to Andreas Nowatzyk for contributing this question.)* [10 points]

b) Explain the advantage of a list-based lock over a ticket lock. [4 points]
c) Under what circumstances might a test-and-test-and-set lock be preferable to a ticket or array/list based lock? [4 points]


d) Explain why the following barrier 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]

1:   global (shared) count : integer = P //P is # of CPUs
2:   procedure central_barrier
3:       if fetch_and_decrement(&count) == 1
4:           count := P
5:       else
6:           repeat until count == P
4) Cache Coherence [18 points]

a) Complete the following coherence state transition diagram for an MOSI coherence protocol on a symmetric multiprocessor using the action / reaction symbols below. (Do not introduce additional states - you do not need to worry about transient states for this question). An example transition has been filled in for you. [12 points]

PrRd - Processor Read
PrWr - Processor Write
PrEvict - Processor Evict (due to cache replacement)
BusRd - Bus Read
BusRdX - Bus read exclusive (i.e., read-and-invalidate)
BusInv - Bus invalidate
BusDataReply - Bus cache-to-cache data transfer
BusWB - Bus Writeback

PrRd / BusRd

I

S

O

M
b) Give an example of a circumstance where update-based cache coherence is preferable to invalidation-based cache coherence. [4 points]

c) Explain why persistent requests are necessary in Token Coherence. [4 points]
In shared memory programs, accesses that are guarded by locks often exhibit migratory sharing patterns -- i.e., data are often read and modified on one processor before migrating to another processor that subsequently enters the critical section. Unfortunately, for migratory data, conventional coherence protocols are inefficient because the read to the data transitions the data into the "shared" state and the write requires an extra bus transaction to invalidate other copies, resulting in two consecutive bus transactions to read and write.

Design a table-based Upgrade Predictor that would identify such read/write sequences (i.e., an instruction PC pair) to the same address and upon future visits to the read, would upgrade the read to a write by sending a write request to the cache, thereby avoiding the extra bus transaction. (Hint 1: The tables should not be indexed by data addresses because index space would be huge. Hint 2: Only read/write sequences by one processor without intervening accesses by other processors are migratory. Hint 3: This prediction problem is very similar to that solved by memory dependence predictors, which predict load-store memory dependences in modern superscalar processors to prevent load instructions from speculatively bypassing older unresolved store instructions in the load-store queue that write to the same address.)

(Thanks to Babak Falsafi for contributing this question.)

- Draw a diagram for your predictor
- Describe the learning process
- Describe the lookup and upgrade prediction process
- What happens upon a misprediction?
(additional space for question #5)