EECS 570 Midterm Exam
Winter 2013

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>/ 32</td>
</tr>
<tr>
<td>2</td>
<td>/ 24</td>
</tr>
<tr>
<td>3</td>
<td>/ 24</td>
</tr>
<tr>
<td>4</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. 22 minutes per problem).
- There are 9 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 [32 points]

a) Why is the granularity of cache coherence an important design parameter? [4 points]

b) What is "false sharing"? Suggest at least one way to mitigate false sharing. [4 points]

c) Provide an advantage and disadvantage of fine-grained locking over coarse-grained locking. [8 points]

d) Consider a large-scale scientific simulation that has 15% sequential code, while the remaining 85% is embarrassingly parallel. When running this application on an 8-core machine, what is the maximum speedup that can be achieved? [4 points]
e) Briefly describe a scenario where increasing last level cache size can reduce performance [4 points]

f) Contrast Eager and Lazy conflict detection in transactional memory systems. [4 points]

g) Can a wait-free queue that supports “push,” “pop” and “peek” operations be implemented using compare-and-swap operations in a system with more than 2 processors? Informally explain the reasoning for your answer. [4 points]
2) QOLB [24 points]

a) Which portions of the Lock Transfer time do Queue-based locks help mitigate

   (1) Arbitration
   (2) Lock Transfer
   (3) Both
   (4) None

   Please provide a brief explanation to support your choice. [4 points]

b) What is "local spinning" on a lock variable? Why is it important? [4 points]

c) Is it possible to use Synchronous Prefetch to completely hide lock transfer times? Provide a brief explanation. [4 points]

d) Explain a scenario where collocation of lock and data is harmful in the case of a Test-and-Set lock. [4 points]
e) Why does collocation of lock and data work better for a Test-and-Test-and-Set (TTS) lock than a Test-and-Set (TS) lock. [4 points]

f) State one major limitation of a QOLB lock. [4 points]
3) Directory-Based Coherence [24 points]

a) Describe how a self-downgrade (sometimes also called flush) transaction can lead to performance improvement. [4 points]

b) Describe a scenario under which cruise missile invalidations result in an improvement in read access latency. [4 points]

c) Describe a scenario under which cruise missile invalidations result in worse read access latency. [4 points]

d) Explain what is meant by “live-lock” in a cache coherence protocol. Give an example scenario where a live-lock might occur. [4 points]
e) Identify and briefly describe **four (4)** alternative ways of storing the set of sharers for directory-based cache coherence protocols. [8 points]
4) Snoop Filter [20 points]

One of the key limitations of bus-based symmetric multiprocessors is that all CPUs must listen to the bus and perform snoop actions (check their cache tags to see if they must intervene in a memory request) for every single memory transaction that appears on the bus. This snoop traffic occupies substantial tag array bandwidth and energy, but the vast majority of snoops miss in the cache.

One approach that industry has used to improve the scalability of bus-based systems is to split the shared bus into two halves (each connecting to half the processors/memories) and introduce a *snoop filter* between the two busses. The goal of the snoop filter is to identify memory transactions that do not need to be propagated from one bus to the other because the snoop filter can be certain that no cache connected to the other bus has a copy of the cache line. A trivial way to implement a snoop filter is to maintain a complete directory of all address. Of course, this approach has a very high storage overhead. Propose a more storage-efficient snoop filter design.

- Draw a diagram for your snoop filter (What state does it contain?)
- Argue why your snoop filter will be able to filter out many coherence requests with reasonable storage (Why should we believe that it will work?)
- Describe the lookup and learning processes (What are the steps of a lookup and learning? How does the snoop filter discover addresses it can filter?)
- In what ways can your snoop filter mispredict? How is correctness still maintained?
(additional space for question #4)