Full Name:_____

CS 15-418, Spring 2004

Practice Exam 2

Instructions:

- This practice exam may help you study for Exam 2, which will be held from noon-1:20pm on Tuesday, April 6th.
- There will be *two special review sessions* to help you prepare for this exam:

  Tuesday, March 30th: We will do a quick review of the lecture material since the first exam.

  Thursday, April 1st: We will go over the answers to this practice exam, and also some practice written problems from the book which will be posted to the class web page.
- The real exam will be CLOSED BOOK, CLOSED NOTES. (In contrast with Exam 1, you will not be using a study sheet for this exam.) You may use a calculator, but not a laptop. (No wireless devices.)

Do not write below this line

Problem	Your Score	Possible Points
1		24
2		21
3		15
4		20
5		20
Total		100

Snoop-Based Multiprocessor Design

Problem 1. (24 points):

The following questions are related to the design of bus-based shared-memory multiprocessors. While keeping your answers brief, please be sure to explain the reasoning behind your answers.

A. Explain why a split-transaction bus typically requires *flow control*, whereas an atomic bus typically does not.

B. Explain what "*fetch deadlock*" is, and what one needs to do to avoid it on a split-transaction bus.

C. A colleague suggests that we could avoid the explicit tagging of requests and responses on a split-transaction bus with out-of-order responses (e.g., the way it is done with the 3-bit tags on the SGI Challenge bus) by using the data addresses as "tags" instead. Do you agree or disagree? Please explain your answer.

D. A colleague suggests that in order to exceed the performance limitations of a single bus, you could build a system with *two* complete buses (each of which handles both requests and responses). Can we use such a system and still achieve sequential memory consistency? If so, how? Please be sure to explain your answer.

Problem 2. (21 points):

The following questions are related to the design of large-scale multiprocessors with physically distributed memory. While keeping your answers brief, please be sure to explain the reasoning behind your answers.

A. Briefly describe three differences between *bus transactions* versus *network transactions*.

B. What is the performance motivation for using communication assist hardware to distinguish user-versus-system messages, compared with simply implementing network transactions via physical DMA?

C. A common challenge with large-scale distributed-memory multiprocessors is the *input buffer problem*: i.e. the possibility that too many processors will send (possibly large) messages to the same processor at the same time, exceeding its buffering capacity. Briefly describe and contrast how the input buffer problem is addressed for each of the following four types of communication:

(a) a remote read under the shared address space abstraction;

(b) synchronous message passing;

(c) optimistic asynchronous message passing;

(d) conservative asynchronous message passing.

Synchronization

Problem 3. (14 points):

The following questions are related to synchronization via locks and barriers.

A. Recall that a *ticket lock* is similar to waiting your turn at a bakery: an arriving processor first gets a unique ticket (using atomic fetch-and-increment), and then waits until the "`now serving`" counter reaches its own ticket value.

   (a) Briefly explain why a ticket lock does *not* achieve the ideal of `O(1)` traffic (per handoff of the lock).

   (b) Briefly describe a hazard of using *proportional backoff* with ticket locks.

B. Some researchers have proposed building a *hybrid* lock implementation that performs well in both low-contention and high-contention situations. Briefly describe how you might design such a hybrid lock: e.g., how would you implement the low-contention mode versus the high-contention mode, how would you switch between these modes, etc.

Interconnection Networks

Problem 4. (20 points):

The following questions are related to the design of interconnection networks.

A. One goal in interconnection network design that is sometimes overlooked is *partitionability*, which is the ability to give separate parallel programs (that are simultaneously running on different parts of the same physical machine) full access to their portion of the network bandwidth without interference from other programs.

(a) Give an example of a network topology with *good partitionability*. Explain why.

(b) Give an example of a network topology with *poor partitionability*. Explain why.

B. To avoid congestion in the network due to hotspotting, some researchers have proposed always sending a message first to a randomly chosen node, and then having this random node forward the message to the actual destination node. Do you think that this is a good idea? Why or why not?

C. What is the bisection bandwidth in a `5-ary 4-cube` with 32 bit bi-directional links? (All 32 bit-lines are dedicated to carrying data, and can do so in both directions, so that there is only one 32-bit link between any pair of nodes).

Latency Tolerance

Problem 5. (20 points):

The following questions are related to latency tolerance via prefetching and multithreading.

A. A colleague suggests that rather than prefetching data into the processor's cache, the prefetched data should be placed in a special buffer on the processor (separate from the cache) that is not subject to cache coherence. Is this a good idea or a bad idea if we want to hide multiprocessor latencies? Please explain your answer.

B. Briefly describe code examples where:

(a) multithreading (aka multiple-context processing) is likely to be more effective than prefetching in hiding latency;

(b) prefetching is likely to be more effective than multithreading in hiding latency;

C. Consider the performance of a program that exploits the "blocked" approach to multithreading (aka multiple-context processing) to hide memory latency, where the average miss latency is 100 cycles, the average run length is 15 cycles, and the average context switch latency is 5 cycles.

(a) What is the expected processor efficiency with *two* threads per processor?

(b) What is the expected processor efficiency with *ten* threads per processor?