EECS 570 *Midterm Exam - SOLUTIONS*

Winter 2016

Name: _____ unique name: _____

Sign the honor code:

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

_____

Scores:

#	Points
1	**/ 8**
2	**/ 14**
3	**/ 12**
4	**/ 20**
5	**/ 24**
6	**/ 22**
Total	***/ 100***

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 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) Transactional Memory [8 points]

Consider a multi-threaded program that operates with T threads on an array A of size N elements. Each thread has to perform a binary filtering operation on each pair of consecutive array elements. Each thread must begin working on elements at index 0 and 1, followed by 1 and 2, and so on, until the end of the array. The threads must have exclusive access to the pair of elements they are filtering (as they may write to either element), but the order in which the threads apply their filters is immaterial to the success of the program.

Consider the following two ways to implement the program:

Transactional memory: Each thread does the following:

```
Begin Tx {
    for i in range(0,N-1):
        filter(A[i], A[i+1]);
} End Tx
```

Hand-over-hand locking: Assume each array element A[i] is protected by a dedicated lock L[i]. Each thread does the following:

```
acquire(L[0]);
for i in range(0,N-1) {
  acquire(L[i+1]);
  filter(A[i], A[i+1]);
  release(L[i]);
}
release(L[N]);
```

Suppose each filter operation takes a constant time C and the latency of the synchronization operations (begin and end transactions in transactional memory and lock acquire and release in hand-over-hand locking) is negligible.

a) Write an algebraic expression (in terms of C, N, and T) for how long the **transactional memory** program takes to complete. Explain your formula. [4 points]

All the transactions on different threads collide, effectively serializing them.
Total time per transaction = C*N. Total program time = C*N*T.

b) Write and explain an expression for the **hand-over-hand locking** program. [4 points]

All the threads filter the array elements in a pipelined fashion.
Total program time = C*N + C*(T-1)

2) Data Level Parallelism [14 points]

a) Briefly explain what happens when branch instructions are executed in GPU code. In particular, how is performance impacted? [4 points]

When the threads in a warp take different paths at a branch, both paths must be explored. The branch condition is evaluated as a predicate and then individual lanes execute no-ops on the not-taken branch path. The performance impact is that the utilization of the GPU lanes is reduced, as only a subset of lanes execute as each branch path is run.

b) Are GPU memory subsystems typically optimized primarily to minimize latency or maximize bandwidth? [4 points]

maximize bandwidth

c) Briefly explain the difference between the Single-instruction multiple-thread (SIMT) programming model of GPUs and the single-instruction multiple-data (SIMD) model used in CPUs, such as the Xeon Phi. [6 points]

In SIMD, there is a single thread of execution with a single program counter. Within this execution thread, individual SIMD instructions operate on wide registers that contain vectors, performing several arithmetic operations in parallel.

In the SIMT model, there are conceptually many independent threads, each with its own independent program counter. The hardware aligns several of these threads in a warp and executes them in lock-step to improve efficiency.

3) Ahmdahl's Law [12 points]

Consider a program where 20% of its execution is serial and the remainder is "embarrassingly parallel" (i.e., its performance scales linearly in the number of cores for an arbitrary number of cores). The performance of the serial portion of the program is directly proportional to storage access latencies.

a) What is the maximum possible speedup that can be achieved for the program? [3 points]

5x.

b) For a system with 8 cores, what is the maximum speedup that can be achieved? [3 points]

3.33x

Recent advances in storage memory technologies can reduce access latencies in half. With the new storage technology:

c) What is the maximum possible speedup that can be achieved for the program? [3 points]

9x.

d) For a system with 8 cores, what is the maximum speedup that can be achieved? [3 points]

4.5x.

4) Coherence Protocol Optimizations [20 points]

a) Briefly explain how adding an **Owned** (O) state to a cache coherence protocol can improve performance. [4 points]

The owned state allows the cache hierarchy to avoid unnecessary writebacks upon downgrade transactions, thereby reducing writeback bandwidth pressure on the memory controller.

b) The "Migratory Sharing Optimization" discussed in class seeks to optimize cache coherence protocols for the special case of locks and data protected by locks, which tend to be read and then immediately written by one node, then read and written by another node, and so on.

i) What is the optimization and how does it improve performance? [4 points]

In response to a read request, a migratory block in M state provides a reply with write permission, allowing the reader to transition to an "E" state and avoid a subsequent upgrade transaction.

ii) How can a protocol detect a "migratory block" to which the optimization should be applied? How can the protocol detect when a block marked migratory should no longer be treated as such? [4 points]

A block should become migratory when there is an upgrade transaction with the only sharer being the previous writer of the block. Smart protocol designs will wait until two such read-upgrade sequences are observed.

A block is no longer migratory if a second reader requests a copy of the block before it is written by the first reader (which received the block in migratory M state).

c) State one **advantage** and one **disadvantage** of using a distributed linked list to track sharers in a directory-based coherence protocol. [8 points]

Advantage: Scalability. The size of a sharing list is much less constrained than in a full bit vector, as the storage required to remember sharers is provisioned along with each node added to the system.
Disadvantage: Invalidation latency. Invalidation messages must visit all sharers in series. Also, evicting a shared block is complicated, and cannot be done silently (without sending messages).
Other answers may be acceptable.

5) Synchronization [24 points]

Consider an application that has been parallelized by dividing its work into fine-grain tasks. To balance the work across cores, these tasks are added to a single task queue that is protected by a lock. Whenever a core finishes a task, it acquires the lock, pops the next task off the queue, releases the lock, and then executes the task. (This approach is often called "task-based parallelism"). Except for the lock and task queue, assume there are no shared data structures in the application.

a) Suppose tasks are fairly long, and therefore, it is unlikely for multiple cores to finish tasks at the same time. What kind of lock would you recommend to protect the task queue and why? [4 points]

A lock that minimizes the latency of an uncontended acquire operation. Test-and-set or test-and-test-and-set are both appropriate choices.

b) Suppose instead that tasks are extremely short, and the execution time of a task is about the same as a single cache-to-cache transfer. Now, what kind of lock would you recommend and why? [4 points]

A lock that scales well under high contention. Any list or queue based lock is appropriate, such as the array or MCS lock. A ticket-based lock is an acceptable answer if a good rationale is provided.

c) Consider again the case where tasks are fairly long. Would this system prefer an invalidation-based coherence protocol, such as MSI, or an update-based protocol, such as DEC Firefly? [4 points]

Invalidation. The lock and task queue head pointer are migratory and do not benefit from update

d) Suppose, instead of a lock, the task queue were implemented with a wait-free linked list. In a system with eight cores, can the wait-free linked list be implemented using compare-and-swap (CAS) instructions? [4 points]

Yes. A CAS is a universal primitive with infinite consensus number, and hence can be used to build a wait-free implementation of any data structure.

e) Consider the following code, which tries to implement a list-based lock.

```
1:   acquire(lock):
2:      I->next = null;
3:      pred = FetchAndSet(lock,I)
4:      if pred != null
5:         I->must_wait = true
6:         pred->next = I
7:         repeat while I->must_wait

8:   release(lock):
9:      if (I->next == null)
10:        lock = null
11:        return
12:     repeat while I->next == null
13:  I->next->must_wait = false
```

The code above contains an error. Describe the problem and identify the line of code that is wrong. Describe an execution sequence involving two processors that demonstrates the error. Your execution sequence should describe the interleaving of events across the two processors. For example "CPU1 initially holds the lock. CPU2 executes the acquire function up to line 5, then CPU2 executes the release to line 9, then CPU1 proceeds with line 6, …" [8 points]

The error is on line 10. Suppose CPU1 is releasing the lock and about to execute the instruction at line 10. A new acquiring CPU2 can execute lines 2-7 and spin on line 7 waiting for line 13 to be executed. However, CPU1 will never unblock CPU2, because it has already observed a null value for I->next on line 9. The fix is to change line 10 to a compare-and-swap operation "if CAS(lock, I, null)" as shown in the MCS lock code in Lecture 7. The compare-and-swap will fail in the scenario described above, causing CPU1 to reach line 12 and discover that CPU2 is waiting to be unblocked.

6) Approximate Cache Filter [22 points]

Consider a cache C, and an "approximate filter" F that is placed in front of C. The approximate filter is a hardware structure can answer questions of the form "Is a line L in C?" The catch is that the answer may sometimes be wrong (thus the name "approximate filter"). In this system, a cache lookup proceeds as follows:

- When an access to a line L occurs, F is queried whether L is present
- If F indicates that L is not in C, the request is sent to the next level cache (or main memory)
- If F says that L is in C, the cache is accessed for line L (which may still be a miss, since the answer given by F might have been wrong).

In this problem, we will consider the correctness of such a filtering scheme with various types of filters and caches. A system is said to be "correct" if, for any possible sequence of valid accesses, the system returns the most recent version of the data.

a) First consider a filter F that reports no false negatives but may report false positives. That is, if a line L is in C, F will always answer that L is in C. But if a line L is not in C, F may sometimes say that L is in C.

i) Suppose C is a read-only cache. Is the system correct? If not, propose a change to the system to make it correct. [4 points]

Correct.

ii) Suppose C is read-write cache C. Is the system correct? If not, propose a change to the system to make it correct. [4 points]

Correct.

b) Next, consider a filter F which has can report both false positives and false negatives.

i) Suppose C is a read-only cache. Is the system correct? If not, propose a change to the system to make it correct. [4 points]

Correct.

ii) Suppose C is read-write cache C. Is the system correct? If not, propose a change to the system to make it correct. [4 points]

Incorrect. The next level of the cache hierarchy could track that the line is dirty in a higher level cache and redirect the request back up to C, much like a snoop operation.

c) Under what circumstances could a filtering scheme such as those described above provide a performance advantage? Explain a simple example scenario. [6 points]

If looking in the filter is faster than looking in cache C, and the hit rate of the filter is sufficiently high, the filter accelerates misses to the next level of the cache hierarchy, improving performance.