EECS 570 Midterm Exam – Solutions
Winter 2020

Name: _______________________________________        Uniqname: ____________________

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>Question</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Short answers</td>
<td>/ 14</td>
</tr>
<tr>
<td>2</td>
<td>Amdahl's law</td>
<td>/ 14</td>
</tr>
<tr>
<td>3</td>
<td>Scheduling</td>
<td>/ 6</td>
</tr>
<tr>
<td>4</td>
<td>Transactional Memory</td>
<td>/ 20</td>
</tr>
<tr>
<td>5</td>
<td>Speculative lock elision</td>
<td>/ 6</td>
</tr>
<tr>
<td>6</td>
<td>Synchronization</td>
<td>/ 20</td>
</tr>
<tr>
<td>7</td>
<td>Coherence</td>
<td>/ 20</td>
</tr>
<tr>
<td>Total</td>
<td></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 90 minutes for the exam.
• There are 10 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 [14 points]**

a) Give an example reason why a parallel program might not achieve linear speedup in a multicore system. (Linear = speedup of $p$ with $p$ processors) [2 points]

    synchronization overhead; load imbalance; sequential portion of the algorithm creating an Amdahl bottleneck

b) Can increasing the last level cache size reduce performance? Why? Assume cache associativity and block size remains the same. [2 points]

    Larger LLC has higher hit time, so reduces performance if the application's working set fits in a smaller LLC.

c) State one advantage of message passing over shared memory. [2 points]

    scales to larger systems, more efficient communication, ease of programming

d) Between online transaction processing (OLTP) and scientific computing workloads, which is more likely to benefit from hardware mechanisms that allow for higher intra-thread memory level parallelism (MLP) -- that is, support multiple outstanding misses from each thread? Explain. [2 points]

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

e) Explain why maintaining inclusion between an L1 instruction cache and a unified L2 cache might be a bad idea. [2 points]

    Instructions are read only, so no benefit to maintaining cache coherence; reduces effective L2 cache capacity; incurs more evictions from L1 cache

f) 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 LRU replacement and that all addresses map to a single set in each cache level. [4 points]

    A B A C A D A E
2) Amdahl's Law [14 points]

Consider a large-scale scientific simulation that has 5% sequential code, while the remaining 95% is embarrassingly parallel.

a) What is the maximum possible speedup that can be achieved for the program? [2 points]

\[ 1/0.05 = 20x \]

b) What is the speedup that can be achieved in a system with \( p \) processors? [4 points]

\[ 1/(0.05+0.95/p) \]

Consider a system in which the cost of memory components is normalized to 1 and the cost of non-memory components scale linearly with the number of processors, i.e. cost of non-memory components in a system with \( 'p' \) processors is \( p \). Assume that memory cost stays the same when the number of processors is scaled.

c) What is the costup of the system with \( 'p' \) processors? [4 points]

\[
\begin{align*}
\text{Cost}_p &= 1+p \\
\text{Cost}_1 &= 2 \\
\text{Costup} &= (1+p)/2
\end{align*}
\]

d) Given the expression for speedup you determined in part (b) and costup in part (c), what is the largest number of processors for which scaling is cost-effective for this program? [4 points]

\[
\begin{align*}
\text{Speedup} > \text{Costup} \\
1/(0.05+0.95/p) > (1+p)/2 \\
(p-19)(p-1) < 0 \\
p < 19
\end{align*}
\]

18 is the maximum cost-effective number of processors.
3) Scheduling [6 points]

\[ T_p(S) : \text{Time on } P \text{ processors using scheduler } S \]
\[ T_p : \text{Time on } P \text{ processors using best scheduler} \]
\[ T_1 : \text{Time on a single processor (sequential cost)} \]
\[ T_\infty : \text{Time assuming infinite resources} \]

Given a directed acyclic graph \( G \) for an application representing its tasks and dependencies, a scheduler \( S \), and \( P \) processors, provide a brief proof sketch to establish the following:

\[ T_p(S) \leq T_1/P + T_\infty \]

Consider the task graph (DAG) with \( W \) total nodes and critical path of depth \( D \).

\[ T_1 = W, \quad \text{all nodes will execute sequentially, assume one time step per node} \]
\[ T_\infty = D, \quad \text{each task along critical path will still be executed sequentially} \]

With \( P \) processors, assuming no dependencies, we can execute \( P \) tasks per time step.
\#complete steps \( \leq W/P \)

When we cannot execute \( P \) tasks due to dependencies, one of the tasks being executed must be on the critical path; i.e. on every incomplete step, we reduce pending tasks on critical path by one.
\#incomplete steps \( \leq D \)

Since every time step, we either take a complete step executing \( P \) tasks, or reduce critical path length by one, we are guaranteed to finish within \( W/P + D \) time steps.

\[ T_p(S) \leq W/P + D = T_1/P + T_\infty \]

[Lecture 1, Slide 57]
4) **Transactional Memory [20 points]**

Consider a multi-threaded program with 16 threads. Each thread is given a distinct thread id tid in [0,15] and performs a square operation on a set of integers in an array A as shown in the code below. The total number of integers N in the array is a multiple of 16. Each array element in A is a 4-byte integer.

```c
begin_transaction();
for (i = 0; i < N; i += 16)
    square(A[i+tid]);
end_transaction();
```

Suppose this program executes on a hardware transactional memory (HTM) system with 16 cores, each thread running on one core. The HTM system buffers speculative updates in a 1MB L2 cache with a 64B cache line size and implements an eager conflict detection and eager version management scheme.

a) Identify a performance inefficiency that will arise when the program is executed on the HTM system, but not in a conventional lock-based implementation. [3 points]

Due to false sharing, transactions will frequently abort even though there are no true data conflicts.

b) Briefly describe how the program could be altered to resolve the inefficiency. [3 points]

Ensure that each thread operates on a contiguous block of integers rather than every 16th integer.

c) State one advantage and one disadvantage of **eager** conflict detection as compared to **lazy** conflict detection in transactional memory systems. [2 points]

**Advantage:** conflicting transactions abort immediately, do not waste resources for work that is later invalidated.

**Disadvantage:** overhead of detection on every memory access, unnecessary abort on temporally silent writes.

d) State one advantage and one disadvantage of hardware transactional memory as compared to software transactional memory. [2 points]

**Advantage:** higher performance

**Disadvantage:** difficult to support unbounded transaction sizes
Consider the following two transactions that are executed concurrently in two processors. [10 points]

Initial state: \( X = 0 ; Y = 0; \)

\[
\begin{align*}
T1 & & T2 \\
\text{begin} & & \text{begin} \\
M1: X = 1 & & N1: Y = 1 \\
M2: Y = 2 & & N2: X = 2 \\
\text{end} & & \text{end}
\end{align*}
\]

Consider the following memory states after speculatively executing the two transactions concurrently, \textit{before} the transactions are committed. For each state, argue whether or not the execution is \textbf{feasible} in a modern out-of-order processor. If it is feasible, determine whether or not the execution of transactions is \textbf{serializable} (no need to roll-back).

i) Memory state: \( X = 2 \quad Y = 1 \)
   Feasible: Yes / No  
   Serializable: Yes / No  
   Reason: T2 executes after T1

ii) Memory state: \( X = 2 \quad Y = 2 \)
    Feasible: Yes / No  
    Serializable: Yes / No  
    Reason: M1; N1; M2; N2  
    T1 and T2 conflict; the problem asks for \textit{feasible} intermediate state, not committed state

iii) Memory state: \( X = 0 \quad Y = 0 \)
   Feasible: Yes / No  
   Serializable: Yes / No  
   Reason: T1 and T2 must execute

iv) Memory state: \( X = 1 \quad Y = 1 \)
   Feasible: Yes / No  
   Serializable: Yes / No  
   Reason: M2; N2; M1; N1  
   M2 and N2 are executed first out-of-order
5) **Speculative Lock Elision [6 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. [2 points]

Circle one:  Higher / Lower / About the same

Reason:  SLE elides acquisitions of the lock in the common case when 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 single lock. [2 points]

Circle one:  Higher / Lower / About the same

Reason:  The threads all conflict when trying to increment the shared counter, hence, speculation will always fail, resulting in many rollbacks.

c) A system running a multi-programmed workload of several single-threaded benchmarks from the SPEC CPU integer suite. [2 points]

Circle one:  Higher / Lower / About the same

Reason:  The single-threaded programs contain no locks.
6) **Synchronization [20 points]**

a) 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]

```plaintext
global (shared) count : integer = P //P is # of CPUs
procedure central_barrier
  if fetch_and_decrement(&count) == 1
     count := P
  else
     repeat until count == P
```

Consider the situation where one or more threads are waiting at the barrier (on line 6) and the last arriving thread executes lines 3 and 4, resetting count to P and proceeding past the barrier. That thread might continue executing to the next use of the same barrier, where it decrements P before all threads have proceeded past line 6. These threads are then “stuck” at the previous barrier, and the system is deadlocked (count !!= P and will never be decremented to 1 since some threads cannot proceed out of the first barrier).

The barrier can be fixed by having waiting threads spin on a separate boolean variable, “global_sense”. Each thread maintains a local variable “local_sense” that is toggled upon its arrival at the barrier. It then spin until “global_sense” matches “local_sense”. The last-arriving thread (detected via count) toggles global_sense. [Sense-Reversing Barriers, Lecture 5].

b) Among the following lock algorithms, circle the locks: [2 points × 5]

i) that provide fairness.

   - test&set
   - test&test&set
   - ticket lock
   - array-based lock
   - MCS lock

ii) that require the number of threads that might acquire the lock to be known in advance.

   - test&set
   - test&test&set
   - ticket lock
   - array-based lock
   - MCS lock

iii) for which Speculative Lock Elision could elide uncontended lock acquisitions.

   - test&set
   - test&test&set
   - ticket lock
   - array-based lock
   - MCS lock

iv) where forward progress might be stopped if the operating system were to deschedule a thread waiting to acquire the lock.

   - test&set
   - test&test&set
   - ticket lock
   - array-based lock
   - MCS lock

v) require the instruction set architecture to provide an atomic memory operation of some kind in order to implement the lock.

   - test&set
   - test&test&set
   - ticket lock
   - array-based lock
   - MCS lock
7) Optimizing Snoop-Based Cache Coherence [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 addresses cached in each half. Of course, this approach has a very high storage overhead.

Propose a more storage-efficient snoop filter design that is imprecise but still guarantee correctness (coherence).

See, JETTY by Moshovos et. al., and Region Scout by Cantin et. al.

i) What does your snoop filter predict?

ii) Describe your snoop filter design with a diagram.
iii) Does your snoop filter have false positives? If so, how do you ensure correctness? Is there a performance cost due to a false positive?

iv) Does your snoop filter have false negatives? If so, how do you ensure correctness? Is there a performance cost due to a false negative?

v) 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?)

vi) 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?)