EECS 570 Midterm Exam  
Winter 2021

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>/15</td>
</tr>
<tr>
<td>2</td>
<td>/10</td>
</tr>
<tr>
<td>3</td>
<td>/20</td>
</tr>
<tr>
<td>4</td>
<td>/25</td>
</tr>
<tr>
<td>5</td>
<td>/30</td>
</tr>
<tr>
<td>Total</td>
<td>/100</td>
</tr>
</tbody>
</table>

NOTES:
- Open lecture notes and open papers assigned for reading.
- Calculators are allowed.
- Do not use web sources.
- Be sure to show work and explain what you’ve done when asked to do so.
1) **Short Questions [15 pts]**

a. Can transactional memory be used to replace barrier synchronizations? Explain. [3 pts]

b. Is it possible for a concurrent program that uses transactions to perform worse than its single-threaded execution? Explain with an example, if necessary. [3 pts]

c. Describe briefly how GPUs enable fast warp-level context switches. Specify two main reasons why a similar functionality (fast thread context switch) is less beneficial for CPUs, so much so that none of the commercially available CPU hardware have that support. [6 pts]

d. What is the main difference between SIMD vs SIMT from a programming perspective? [3 pts]
2. **Parallelism [10 points]**

(a) Say we have an application with three steps that run sequentially - step A, step B, and step C each take 35%, 25%, and 40% of the total program runtime respectively. You read the source code and realize that each step can be parallelized. But due to how the data is organized there are limits to how many cores each step can effectively use:

- Step A can run on at most 8 processors
- Step B can run on at most 10 processors
- Step C can run on at most 4 processors.

If you only have time to optimize one of these steps, which would you choose and why? [4 points]
(b) Say we had a programmable multi-core processor that could reconfigure itself in a negligible amount of time to change its number and size of cores. Assume two valid configurations:

Big-Few: Few big cores  
Small-Many: Many small cores

A data-center needs to process tasks with different QoS constraints (specified in terms of latency and throughput). Determine a policy that a “controller” in a data-center can use to configure its programmable multi-core processors and schedule tasks on them. [6 points]
3) Cache Coherency [20 points]

Upgrade Predictor

In shared-memory programs, accesses that are guarded by locks often exhibit migratory sharing patterns -- i.e., data is 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 places 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 to eliminate the first bus read 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.)

- Draw a diagram for your predictor (or describe in detail)

- Describe the learning process
• Describe the lookup and upgrade prediction process

• What happens upon a misprediction?
4) Synchronization [25 points]

(a) Explain a scenario where spin based synchronization will be better than blocking based synchronization. [1 pt]

(b) Explain a scenario where blocking-based synchronization is better than spin-based synchronization. [1 pt]

(c) Improve the fairness of a standard Test-and-Test-and-Set lock implementation using Random(x,y) which returns a random integer in the range of [x,y]. Assume Random(x,y) is truly random and cheap. In addition, you can introduce only one additional global variable.

Rewrite the code below and explain your changes. [11 points]

```
// Test-and-Test-and-Set Skeleton Code
acquire(lock_ptr):
    while (true):
        // Perform “test”
        load [lock_ptr] -> original_value
        if (original_value == UNLOCKED):
            // Perform “test-and-set”
            old = compare_and_swap(lock_ptr, UNLOCKED, LOCKED)
            if (old == UNLOCKED):
                break // lock acquired!
            // keep spinning, back to top of while loop

release(lock_ptr):
    store[lock_ptr] <- UNLOCKED
```
// Your implementation

acquire(lock_ptr):

release(lock_ptr):
(d) Analyze the following properties for your Randomized Test-and-Test-and-Set implementation, and compare it with test-and-test-and-set lock and ticket locks. [12 points]

i) **Performance**: Communication (coherence) overhead on a release during high lock contention:

ii) **Performance**: Acquire latency during low lock contention:

iii) **Fairness:**
5. Transactional Memory for Detecting Data Races [30 pts]

(a) Say Intel releases a processor with hardware transactional memory (HTM-U) feature that supports unbounded transactions. Its design extends cache coherence protocol to detect conflicts. It handles unbounded transactions by serializing threads (executing one transaction at a time).

Unfortunately, all shared-memory multi-threaded software until now have been written using legacy synchronizations (locks, signal-wait, barriers, etc.). Obviously, they don’t use transactions, and so they will not benefit from HTM-U without additional effort.

Legacy parallel software may contain data-race concurrency bugs, and they will continue to be executed on legacy systems. Your goal is to develop an automatic debugging tool that uses HTM-U for finding data-races in legacy software as they manifest when these programs are executed. You have the freedom to change any part of the system - hardware, compiler, or runtime (OS). But make sure that any data-race that can manifest in the original system can manifest in the modified system -- this property is essential to make sure that your tool can find data-race bugs in the legacy program while running on legacy systems.

**Definition of a dynamic data-race:** A program execution has a data-race if two threads access the same memory location, at least one of those accesses is a write, and there is no happens-before order between those two accesses due to programmer defined synchronization in the original program (using legacy synchronization operations).

**Sketch your main idea:**
(Make sure to specify what constitutes a transaction in the modified legacy program)

**Hardware changes, if any:**
Compiler changes, if any:
(Hint: compiler allows you to emit extra code into the binary for analyzing a program when it executes):

OS changes, if any:

Does your design guarantee to detect all data-races that manifest dynamically? Explain.

Does your design report any false data-races (data-races that did not manifest in the execution)? Explain.

(b) Say Intel’s HTM does not support unbounded transactions. How would this limit your capability in detecting data-races?
(c) If you want to help improve the correctness and performance of legacy software while executing on an HTM-U system, what automatic program transformations would you recommend? Be specific about how you propose to handle each of the legacy synchronization operations: locks, barriers, signal-wait.