Name: Roll No.:

Finals
CS5300: Parallel and Concurrent Programming
Autumn 2020

Instructions for the exam:

- You must submit your final answer copy as pdf.
- You should avoid submitting scan copies of hand-written notes. Only if you wish to attach any figure, you can attach the scans of the figures in your pdf.
- Please justify your answers.

Q1 **(5 points)**. Some platforms provide a pair of instructions that work in concert to help build critical sections. On the MIPS architecture, for example, the load-linked and store-conditional instructions can be used in tandem to build locks and other concurrent structures. The C pseudocode for these instructions is shown below. Alpha, PowerPC, and ARM provide similar instructions.

Listing 1: Load Link Store Conditional

```
1
2  int LoadLinked(int *ptr) {
3      return *ptr;
4  }
5
6  int StoreConditional(int *ptr, int value) {
7      if (no one has updated *ptr since the last LoadLink to this address) {
8          *ptr = value;
9          return 1; // success!
10     }
11     else {
12         return 0; // failed to update
13     }
14 }
```

Please show the the consensus number of this instruction-set is ∞ similar to CAS operation.

Q2 **(5 points)**. Please explain how CLH lock uses a space of $O(L + n)$ for L locks with n threads/processes.

Q3 **(5 points)**. The current implementation of tryLock creates a new node in every invocation. What is the problem with reusing the nodes in this tryLock implementation?

Q4 **(6 points)**. Design an isLocked() method that tests whether any thread is holding a lock (but does not acquire the lock). Give implementations for

(a) The CLH queue lock [3 pts]

(b) The MCS queue lock. [3 pts]

Please explain your answers.

Q5 **(7 points)**. Consider the following wait-free 'contains' variant for the lazy synchronization implementation of a set as discussed in the class.

Listing 2: Modified Wait-Free Contains for Lazy Synchronization

```
1  public boolean contains(T item) {
2      int key = item.hashCode();
3
4      Node pred = this.head; // sentinel node;
5      Node curr = pred.next;
6
7      while (curr.key < key) {
8          pred = curr;
9          curr = curr.next;
10     }
11
12     return (curr.key == key);
13 }
```

As you can see this contains implementation does not check the 'marked' bit. Please describe the linearization points of all the three methods - contains, add and remove.

Q6 **(6 points)**. Consider the non-blocking implementation of a concurrent set based on linked-list. As discussed in the class, this implementation uses 'helping' among the threads.

(a) Please explain how if the threads did not help each other, then the implementation will not be *lock-free*. Specifically, explain how removing lines 16-21 from find() method affect the algorithm? [3 pts]

(b) Without 'helping' will the implementation be *obstruction-free*? [3 pts]

Q7 **(14 points)**. Consider the 'remove' method in the lazy synchronization implementation of a set as discussed in the class.

(a) Consider a variant in which we have the following line after Line 18 as follows: *curr = null;* The motive behding saving such a line can speed up the garbage collection process in Java.

Please show that this change will result in other undesirable side-affects in working of the concurrent set. [4 pts]

(b) Consider implemeting this lazy list in C++ language. Note that C++ does not have automatic garbage collection like in Java. So how can ensure that there is no *memory leak* in the remove method. In other words, how can you free the removed memory nodes (like the 'curr' node that has phsyically removed in the remove method). [10 pts]

Q8 **(15 points)**. In the class we discussed a parallel implementation of graph coloring using barriers. Please answer the following while justifying your answers:

(a) Please develop a pseudocode of this algorithm. [3 pts]

(b) How will your algorithm behave if we did not use the barriers. [4 pts]

(c) Can you develop a non-blocking version of this algorithm? [8 pts]