Assignment 3

ECSE 420: Parallel Computing

Due: December 7, 2017

Submission instructions: Students are to work in groups of two on the assignment.

Students must also submit a pdf file that contains the following information: student's names, student ids, instructions on how to run each file and the associated question solved. Students are also expected to submit a zip file containing their source code. This code must compile and run without error. Code must be well formatted, commented, and follow the google java style guide. For more information, see the ECSE420AssignmentSubmissionInstructions.pdf in the Assignment section on myCourses.

Questions

1. (8 marks) Chapter 7, Locks
   1.1. Fig. 1 shows an alternative implementation of CLHLock in which a thread reuses its own node instead of its predecessor node. Explain how this implementation can go wrong.

```
1  public class BadCLHLock implements Lock {
2    // most recent lock holder
3    AtomicReference<Qnode> tail;
4    // thread-local variable
5    ThreadLocal<Qnode> myNode;
6    public void lock() {
7      Qnode qnode = myNode.get();
8      qnode.locked = true;          // I'm not done
9      // Make me the new tail, and find my predecessor
10     Qnode pred = tail.getAndSet(qnode);
11     // spin while predecessor holds lock
12     while (pred.locked) {}
13   }
14   public void unlock() {
15     // reuse my node next time
16     myNode.get().locked = false;
17   }
18   static class Qnode { // Queue node inner class
19     public boolean locked = false;
20   }
21 }
```

Figure 1

2. (18 marks) Chapter 9, examining the fine-grained algorithm

2.1. Provide the code for the *contains()* method missing from the fine-grained algorithm described in chapter 9.
2.2. Write a test to verify the *contains()* method and explain why your implementation is correct.

3. (12 marks) Chapter 10, designing a bounded lock-based queue
   3.1. Design a bounded lock-based queue implementation using an array instead of a linked list. Allow parallelism by using two separate locks for head and tail.
   3.2. Try to transform your algorithm to be lock-free. Where do you run into difficulty?

4. (32 marks) Chapter 16, matrix vector multiplication
   4.1. Implement a sequential matrix vector multiplication
   4.2. Give an efficient and highly parallel multithreaded algorithm for multiplying an n × n matrix A by a length-n vector x that achieves work Θ(n2) and critical path Θ(log n).
   4.3. Write a test program that measures the execution time for multiplying a 2,000 by 2,000 matrix with a corresponding 2000 wide vector using the parallel method and sequential method, respectively. Discuss the execution time of the two methods and compute the speedup. Be sure to indicate how many threads are being used.
   4.4. Analyze and discuss the work and critical-path length of your implementation, and give the *parallelism*.

Total: 70 marks