Concurrent Programming

COMP 409, Winter 2018

Assignment 3

Due date: Wednesday, March 28, 2018
6pm

These instructions require you use Java. All code should be well-commented, in a professional style, with appropriate variables names, indenting, etc. Your code must be clear and readable. **Marks will be <u>very generously</u> deducted for bad style or lack of clarity.**

All shared variable access must be properly protected by synchronization—there must be *no race conditions,* but also no unnecessary use of synchronization. Unless otherwise specified, your programs should aim to be efficient, and exhibit high parallelism, maximizing the ability of threads to execute concurrently. Please stick closely to the described input and output formats.

1. In this question you will evaluate and compare the performance of a lock-free queue over a blocking one. Note that we did not go over the design of concurrent queues directly in class. A detailed implementation design for both blocking and lock-free unbounded queues are given in the text, chapters 10.4 and 10.5 respectively. (Reading 10.2–10.3 may also be useful for background.)

   (a) Create a program which accepts three positive integer arguments, p, q, and n. First, implement and validate a blocking unbounded queue (do not use any built-in versions). Launch p threads that create and add items to the queue and q threads that dequeue items. Each item should include a unique (integer) id, and two timestamps, one for when it was added to the queue, and one for when it was removed. A thread sleeps for ≈10ms between successive queue operations. After a dequeue thread has successfully dequeued n items it terminates. **9**

   After all dequeuing threads have terminated, terminate the enqueue threads and show the timestamps on the fully dequeued items to verify that the queue properly modelled a FIFO queue. Your code should emit as final output a list sorted (ascending) by timestamp, each line showing the operation-name (`"enq"` or `"deq"`) and item id,

   (b) Now, create a similar a program as above, same input, output and behaviour, but based on a *lock-free* unbounded queue implementation. Again, do not use any built-in versions. **12**

   (c) Finally, compare performance. Disable output, and add timing code to measure the duration of the simulation itself, up until all dequeuing threads complete. Set $p = q$ and with $q = 2$ find an n with a significant runtime (a second or more). Execute each of your two versions of the queue at least 10 times to find an average time, and do that for at least 3 larger settings of q. **4**

   In a separate document from your code, use a plot or table to show the relative performance of your two implementations. The text claims the lock-free implementation is substantially faster. Do you agree? Provide some explanation of your results.

2. Speculative, or optimistic approaches to concurrency achieve better performance by using extra threads without strong guarantees that the parallel execution produces complete or correct results. **15**

   Consider the following approach to *graph-colouring*. In this problem, the task is to assign colours to nodes in a graph, such that no two adjacent nodes have the same colour. Finding a minimal number of colours to do this is difficult, but heuristic solutions can perform well, and are known to benefit from parallelism.

   The following algorithm, based on the design of Gebremedhin and Mann is quite simple. We need an input undirected graph, with vertices *ordered* in some fashion (eg from unique ids). We will assign colours as integers > 0 (using 0 to mean uncoloured). Our main loop is as follows:

```
Require: A graph formed of a set $V$ of Node objects, each containing an integer colour value, and an adjacency
    list. Each vertex should have an initial colour of 0.
1: Conflicting = $V$
2: while Conflicting $\neq \emptyset$ do
3:     Assign()
4:     Conflicting = DetectConflicts()
5: end while
```

Within that main loop are two parallelizable functions. The first is `Assign`:

```
Require: Conflicting must be partitioned among the threads.
1: for each subset Conf$_i$ of Conflicting concurrently do
2:     for all $v$ in Conf$_i$ do
3:         Set $v$.colour to the smallest colour not used by any adjacent Node
4:     end for
5: end for
```

The second multithreaded function is `DetectConflicts`:

```
Require: Conflicting must be partitioned among the threads. We also need a new (global) empty set, New-
    Conflicts
1: for each subset Conf$_i$ of Conflicting concurrently do
2:     for all $v$ in Conf$_i$ do
3:         (Atomically) add $v$ to NewConflicts if it has the same colour as any adjacent node $u$, and $u < v$.
4:     end for
5: end for
6: return NewConflicts
```

Define a program which accepts 3 command-line parameters, $n > 3$ (the number of nodes in the graph), $e > 0$ (the number of undirected edges in the graph), and $t > 0$ (the number of threads to use). It should do the following,

(a) Sequentially construct a random graph of n nodes and e edges. Edges are made between random pairs of nodes.

(b) Colour the graph, according to the above algorithm, using t threads. Do not use blocking synchronization other than thread joining. For synchronization only use volatile and atomics and hardware primitives within `java.util.concurrent.atomic`.

(c) Print out the time taken to colour the graph (not including the time to construct it).

(d) Verify the graph is properly coloured. Print out the maximum node degree, and the maximum colour used.

Your design should achieve speedup over $t = 1$. Select n, and e such that it takes significant time to execute with $t = 1$ (more than 10s; note that you will need a big and relatively dense graph). Provide a separate document comparing performance for $t = 2, 4, 8$, based on average time over at least 5 runs of each configuration.

What to hand in

Submit your assignment to *MyCourses*. Note that clock accuracy varies, and late assignments will not be accepted without a medical note: **do not wait until the last minute**. Assignments must be submitted on the due date **before 6pm**.

Where possible hand in only **source code** files containing code you write. Do not submit compiled binaries or .class files. For any written answer questions, submit either an ASCII text document or a .pdf file *with all fonts embedded*. Do not submit .doc or .docx files. Images (plots or scans) are acceptable in all common graphic file formats.

This assignment is worth 10% of your final grade. **$\overline{40}$**