Final Project

Concurrent Programming

Due: 11:59pm, December 7, 2020

Summary: In this project you will implement one or a few parallel algorithms (you have a choice) and analyze their performance.

Implementation Options: You have the choice to implement a selection of the following parallel algorithms, each worth a set of points. Overall, the final project is worth 250 points. You can implement up to 300 points for 50 points extra credit. Your options are:

Concurrent Tree (250 pts) Implement a key-value store implemented as either a sorted tree or skip list. Your structure should use fine-grained synchronization (no global lock, the recommendation is to use hand-over-hand locking). The structure should support `get`, `put`, and `delete`, as well range queries that take two keys and return the key/value pairs between them (range queries need not be linearizable, but you should be able to describe what values might be seen). Your data structure should not leak memory. Your write-up should include experimental explorations across high contention (many threads accessing the same key) and low contention (uniform access pattern) cases. **For an additional 50 pts**, implement the same tree using reader-writer locks such that readers can execute in parallel, and compare the performance.

Concurrent Containers (250 pts) Implement several concurrent stack and queue algorithms, including a SGL stack and queue, the Treiber stack, and the M&S queue. You must also implement two of the following (or, **for an additional 50 points**, all three): (1) all the above stacks with an elimination (`https://people.csail.mit.edu/shanir/publications/Lock_Free.pdf`), (2) a flat-combining stack and queue (`http://mcg.cs.tau.ac.il/papers/spaa2010-fc.pdf`), (3) or the baskets queue (`https://people.csail.mit.edu/shanir/publications/Baskets%20Queue.pdf`). Your data structures should not leak memory, but are allowed to use blocking garbage reclamation schemes (e.g. epoch based reclamation). Your write-up should include experimental explorations of data structure throughput across varying thread counts.

Transactions (250 points) Implement an banking system that supports a fixed size set of accounts, balance queries of individual accounts, and (non-durable) transactions betwen these accounts. Implement transactions in your system using a single global lock, two-phase locking, C++ software transactional memory (supported by GCC), hardware transactional memory (with global lock fall back), and some optimistic concurrency control mechanism (`https://www.eecs.harvard.edu/~htk/publication/1981-tods-kung-robinson.pdf`, p.218 lists three methods, also described here: `https://inst.eecs.berkeley.edu/~cs186/fa06/lecs/21cc3.pdf`). Your write-up should include experimental explorations of throughput across varying thread counts and both high and low contention cases, comparing all mechanisms.

LIFO locks (50 pts) Using your Lab 2 counter micro-benchmark, implement a LIFO lock. Analyze its performance relative to TAS and FIFO locks across thread count.

Code Requirements / Restrictions: This is an open project — you are welcome to use whatever libraries you would like. However, you need to implement the main algorithm(s).

Project write-up: Your write-up will be longer this time, probably around five pages including charts. It should include:

- A description of your algorithm
- Experimental results as required by the prompt

- Analysis of results using `perf` as necessary to support explanations
- A description of your code organization
- A description of every file submitted
- Compilation instructions
- Execution instructions, particulary for any results presented in the write-up
- Any extant bugs

Code style: Your code should be readable and commented so that the grader can understand what's going on.

Submission: You will submit a zip file of your lab to canvas. When unpacked, the directory should contain all files required to build and run your program, along with a brief write-up. Pay particular attention to the requirements for compilation and execution, as some testing will be done using automatic scripts.

Compilation and Execution:
Your submitted zip file should contain a Makefile and the project should build using a single `make` command. Executables generated should provide execution instructions when given a `-h` flag.

Grading: Your assignment will be graded as follows:

Implementation (70%) Your code should work and meet the project criteria. Incomplete/failing code will be docked points. Your submission should include sufficient unit test cases that we can verify your code is correct.

Lab write-up and code readability (30%) Lab write-ups and readable code that meet the requirements will get full marks. Incomplete write-ups or unreadable code will be docked points.

Recall that late submissions will be penalized 1% per two hours late (up to two days late), and will only be accepted until the final exam. Canvas submissions include the submission time.