# 1 Introduction

You're given a linked list data structure, but it's not designed for multi-threaded use. In other words, it is not *thread-safe*. Your job is to make it safe, and, to the extent this is possible with a linked list, make it fast.

# 2 Preliminaries

This homework requires access to a multi-core machine, ideally with 4 hardware threads or more. Three servers are available for your use for this assignment: Kentsfield-1 and 2, and words. To access the kentsfields, which are hiding behind a firewall, use these commands:

| | |
|---|---|
| Kentsfield-1 | `ssh -p 8021 netid@bits-head.cs.uic.edu` |
| Kentsfield-2 | `ssh -p 8034 netid@bits-head.cs.uic.edu` |
| words | `ssh -p 8190 netid@bits-head.cs.uic.edu` |

your UIC netid password applies. Watch out: if someone else is using the same machine as you at the same time, your performance results will be heavily affected. Before running a performance experiment, take a quick lock with `top`.

# 3 Assignment

The template for this assignment is available at `https://github.com/uicperformance/list`. To build the program, simply run `make` in the checked-out repository folder.

The program, `test_list_single_thread` crashes, hangs in an infinite loop, or exits with an error report when run with `-n 2` or higher. This is due to a race condition: the program works fine with one thread.

## 3.1 Establish performance baseline, and prepare LaTeXreport

Using gnuplot, generate a plot of the total `find`-only throughput of benchmark_list_single_thred for a single thread, and using 4 concurrent threads. Use the `-u 0` flag to configure the benchmark to only do queries. On the x-axis, vary the size of the list between 1–1,000,000 entries. On the y-axis, show the number of find operations performed per second. Show one line for a single thread, and one line for 4 threads. Add more lines for other thread counts if you would like. Make sure axes and lines are correctly labeled, with names and units as appropriate. Below is a little start on a gnuplot script to plot the data in the file `readonly.txt`, which holds the data to be plotted: one line per datapoint, space separated.

```
set terminal "pdfcairo"; // this could be different on Linux
set output "myplot.pdf";
set xlabel 'Threads';
set logscale x;
```

```
plot 'readonly.txt' using 1:($2/1000000) ti 'Read-only' with li
```

There is a make target called report.pdf, and an associated report.tex in the template folder. Update report.tex throughout to reflect your turn-in, and add the first plot to your report, with an appropriate caption. Remove anything in the report template that doesn't belong.

The read-only performance is an upper bound on the performance of this data structure, since in this scenario, each core can independently access the data structure without synchronization or other communication with other cores.

## 3.2 A Concurrent List with a Big Lock

In any workload that includes writes, synchronization becomes necessary to preserve consistency. The homework template comes with a small test suite, and the performance benchmark program. Create a new list implementation, in a file called list_bfl.c, that uses a standard pthread lock `pthread_mutex_t` to ensure thread-safe list access. Use the test program `test_list.c` to verify that your list does not perform illegal operations or lose data, then produce the same type of plot as in §3.1, but with the BFL implementation. Remember to include all code in your turn-in folder.

In your report, provide a brief analysis of any differences between the two plots, and a hypothesis explaining the differences. In particular, note any differences in behavior as you go from 1–4 threads, and from small to large lists.

If you aren't confident in your hypothesis, try using your hypothesis to make a prediction, and verify the prediction by experiment. Any such prediction and experiment will not be graded.

## 3.3 Spinlock vs. Mutex

Run the same experiment as above, but with `taskset c=1`, to limit the program to using a single cpu. That is, run with 1 or 4 threads, but only on one CPU. Include the matching plot in your report.

Then create new Makefile targets, called test_bfl_spin and benchmark_bfl_spin, which uses a `pthread_spinlock_t` instead of a `pthread_mutex_t`. Don't make a new version of list_bfl.c. Instead, update list_bfl.c to use #ifdefs and/or -D flags to gcc in the Makefile target to select between the two lock types. Plot the performance of bfl_spin, with 1 and 4 threads, using a single core, and in a separate plot, with no core limits.

Briefly analyze the difference between spinlocks and mutex locks in your report. What explains the difference in behavior? When is one choice better than the other?

## 3.4 Readers-writers lock

Again, add Makefile targets called test_bfl_rw and benchmark_bfl_rw. This one uses the `pthread_rwlock_t` lock, again without creating a new bfl.c, but using

-D and #ifdefs to generalize the code to support the three lock types.

Produce one plot for benchmark_bfl_rw, with no core limits and 4 threads, but with three lines showing 90% updates, 50% updates, and 10% updates. Compare against the spinlock results, and state your conclusions in the report.

### 3.5 A more concurrent list

So far, we've used a single lock to protect the entire list. For small lists, using the right lock type can make a difference, but as the list grows, the computational complexity of a linked list dominates performance, and the single lock (usually) means that only ane thread at a time can operate on the list.

Create a new version, called `hhl.c`, and matching Makefile targets test_hhl and benchmark_hhl. Update the code to include one lock per node, and use hand-over-hand locking with spinlocks, so that more than one thread can access the list concurrently. Produce a plot as above, but with five lines, for 1,2,4,8,and 16 threads respectively. For this plot, you will need to run the benchmark program on the machine `words`, as the other two only have 4 hardware threads. Compare against the other methods, state your conclusions.

### 3.6 turn-in instructions

Similar to the previous homework, turn-in is by git classroom, using this invitation link `https://classroom.github.com/a/WjiyORhg`. Include `report.pdf` in the root folder of your submission.

### 3.7 Double-check your submission

To make sure that you submitted everything you think you submitted (`git` can be a little tricky until you get used to it), *git clone* your turn-in repository into a fresh folder, check that your `report.pdf` is in the submission folder, and that all the Makefile targets build correctly.