**1.We’ll start by redoing the measurements within this chapter. Use the call gettimeofday() to measure time within your program. How accurate is this timer? What is the smallest interval it can measure? Gain confidence in its workings, as we will need it in all subsequent questions. You can also look into other timers, such as the cycle counter available on x86 via the rdtsc instruction.**

I use gettimeofday() to measure, and it is pretty accurate.

**2. Now, build a simple concurrent counter and measure how long it takes to increment the counter many times as the number of threads increases. How many CPUs are available on the system you are using? Does this number impact your measurements at all?**

./simple\_count\_lock [num\_thread]

There are 8 CPU are available on my system. So each time increase the number of the thread, per thread to increment 1 million count increase around 20%.

**3. Next, build a version of the sloppy counter. Once again, measure its performance as the number of threads varies, as well as the threshold. Do the numbers match what you see in the chapter?**

./approximate\_counter [threshold]

It actually not match what I see in the chapter. My best guess is I didn’t use condition variable, thus, cause the threads spinning and waste cpu time.

**4. Build a version of a linked list that uses hand-over-hand locking [MS04], as cited in the chapter. You should read the paper first to understand how it works, and then implement it. Measure its performance. When does a hand-over-hand list work better than a standard list as shown in the chapter?**

The lookup time doing much better than the standard one.

Text

Description automatically generated

**5. Pick your favorite data structure, such as a B-tree or other slightly more interesting structure. Implement it, and start with a simple locking strategy such as a single lock. Measure its performance as the number of concurrent threads increases.**

I’m going to pick the stack.

Text

Description automatically generated

**6. Finally, think of a more interesting locking strategy for this favorite data structure of yours. Implement it, and measure its performance. How does it compare to the straightforward locking approach?**

Text

Description automatically generated

When I put more lock on the stack, the actual performance to push decrease. The pop performance remain the same.