CS5300: PARALLEL AND CONCURRENT PROGRAMMING, Fall 2020

Programming Assignment 4: Comparison of CLH and MCS Locks

SAI HARSHA KOTTAPALLI

CS17BTECH11036

GOAL

The goal of this assignment is to implement the two locking operations CLH and MCS Locks. Then compare the performance of these locks by measuring two parameters: average and worst-case waiting time for threads to obtain the locks.

DESIGN

COMMON PART

We parse input-params.txt for input to obtain the number of threads(n) and the number of CS accesses(k) by each thread. We also take the averages of exponential distributions which will be used for simulation of some work being done inside and outside CS. We then create n threads, each responsible for doing some work inside CS and outside, k times.

For maintaining Mutual Exclusion between threads accessing CS, we use Locks, namely, CLH and MCS lock.

Here, we call lock() when a thread wants to enter the CS. When a thread attains the lock successfully, it will be the only thread inside CS at that time until it completes its task in CS, upon which it calls unlock().

The logs stored as the program executes give an idea of how the execution proceeds while holding the mutual exclusion property.

Implementation in C++

CLH LOCK

We use atomic<*> class pointer for AtomicReference, exchange() for getAndSet() (whose default argument is seq_consistency). Rest of the code is directly similar to the java algorithm.

Using thread_local makes sure that the associated variable has its own value for each thread.

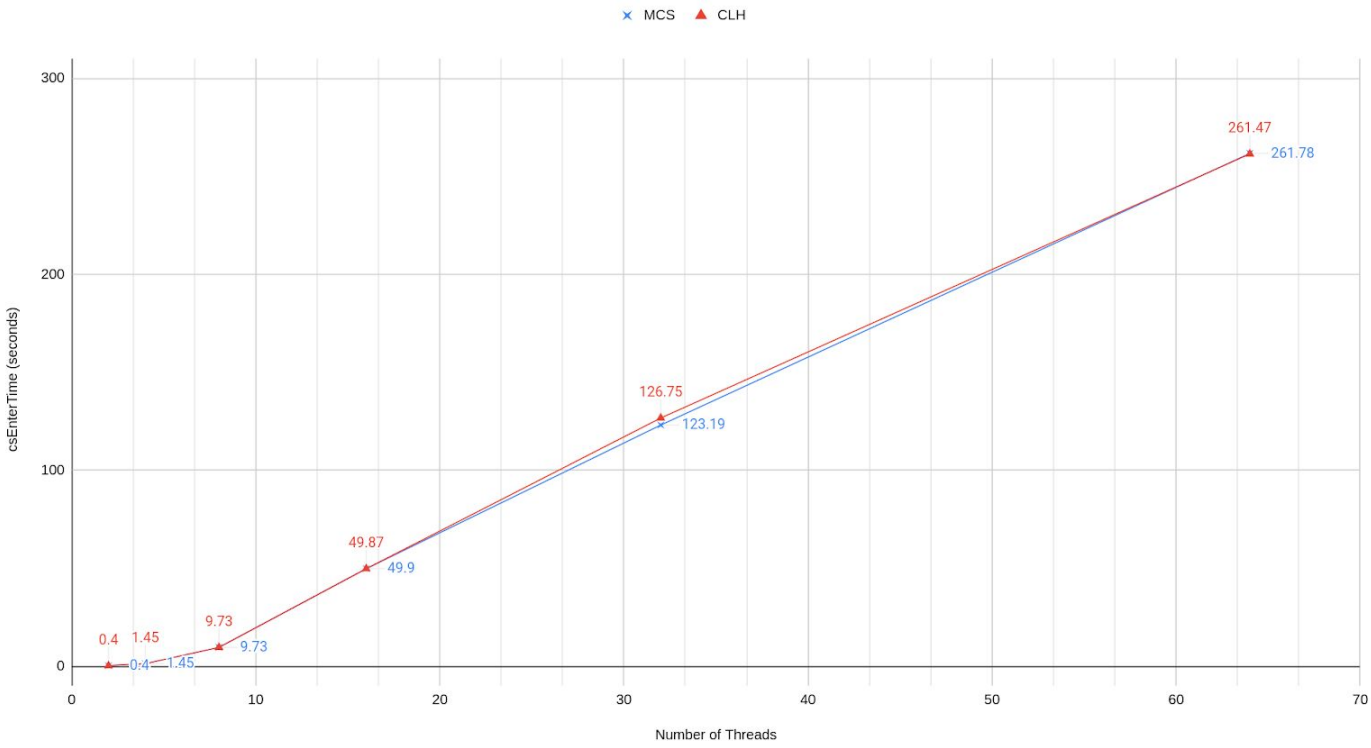
MCS LOCK

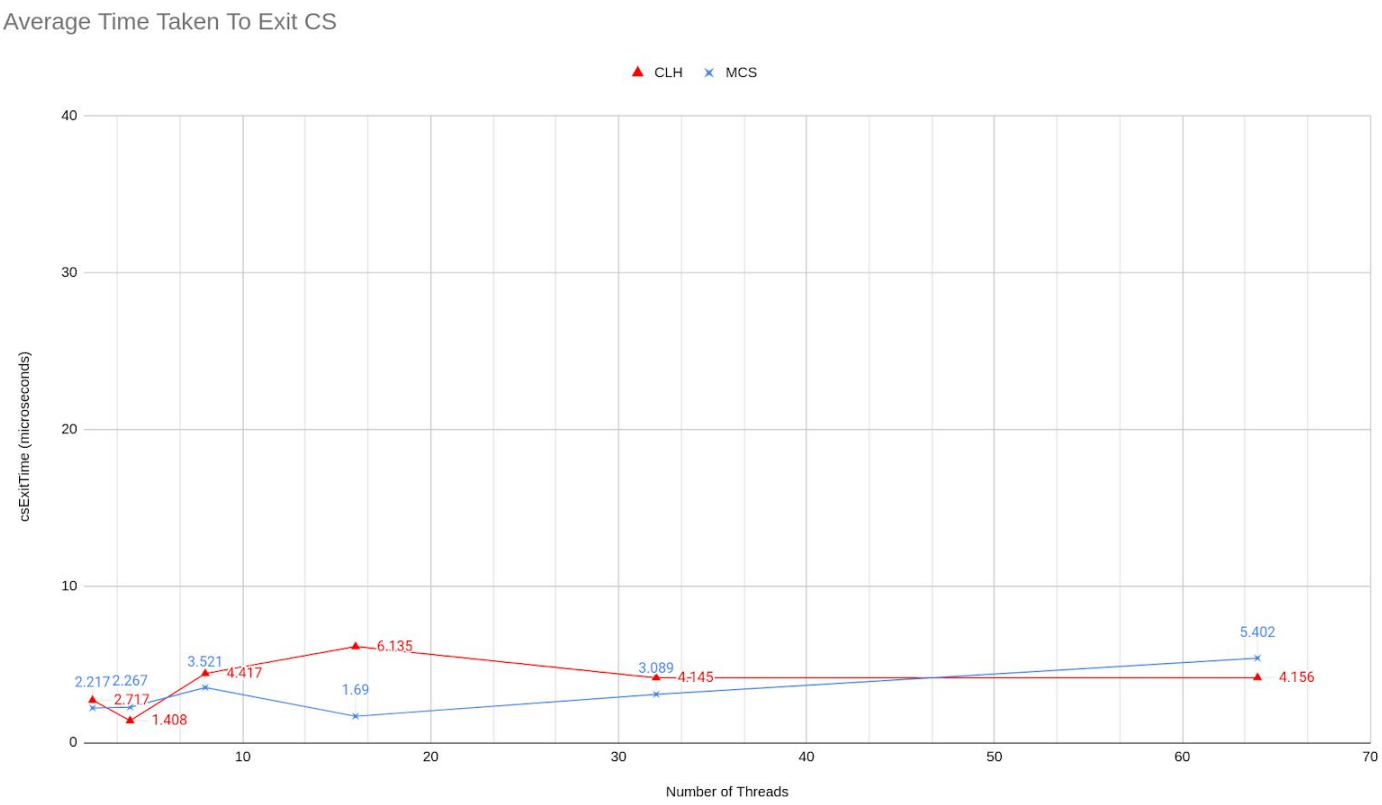
We use atomic<*> class pointer for AtomicReference, exchange() for getAndSet() (whose default argument is seq_consistency) and compare_exchange_strong() for TAS. Rest of the code is directly similar to the java algorithm.

Using thread_local makes sure that the associated variable has its own value for each thread.

GRAPHS

Average Time Taken To Enter CS





OBSERVATIONS

- I have run multiple times (5) and taken the average of the required data to plot the graph. The basic inference is that MCS performs slightly better than CLH.
- Unlock as we can see is in the order of microseconds, this would hardly matter as lock() is in the order of seconds and any advantage from this wouldn't make any significant impact.
- The boost we get from MCS is based on the architecture but never the less, since in MCS threads spins on local while CLH spins on remote, MCS is

observed to be slightly faster. On more number of threads and actual heavy load on cache-less NUMA architectures the advantage would be even more significant.

- In the simulation, sleep() does not accurately help in the simulation as thread just sleeps in this case, unlike a realistic scenario where the current thread is doing some work and the scheduler might decide to pause it for a while or similar situations which are unaccounted for. But since we are taking the time in order of seconds, it shouldn't make much difference anyway.