Lauren Yeung, 104456482
Albert Wang, 504457700
CS 111, Winter 2016
Professor Eggert

Lab 4

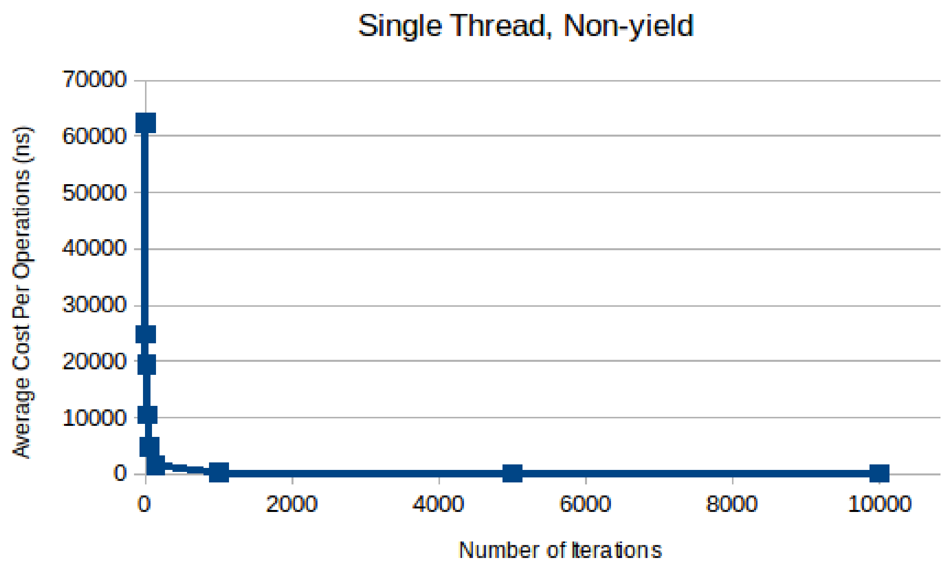
QUESTIONS 1.1

Fails at 10 threads and 1000 iterations.
1. It takes this many threads and iterations to fail because a higher number of iterations and threads increase the chance of race conditions.
With a greater number of threads, more threads are modifying the global counter, especially for a greater number of iterations.

2. A significantly smaller number of iterations seldom fails because it lowers the chance of collision. Smaller number of iterations also means that context switching is less likely to affect the global variable, as the changes may negate each other.

One Thread, Non Yielded, Unprotected

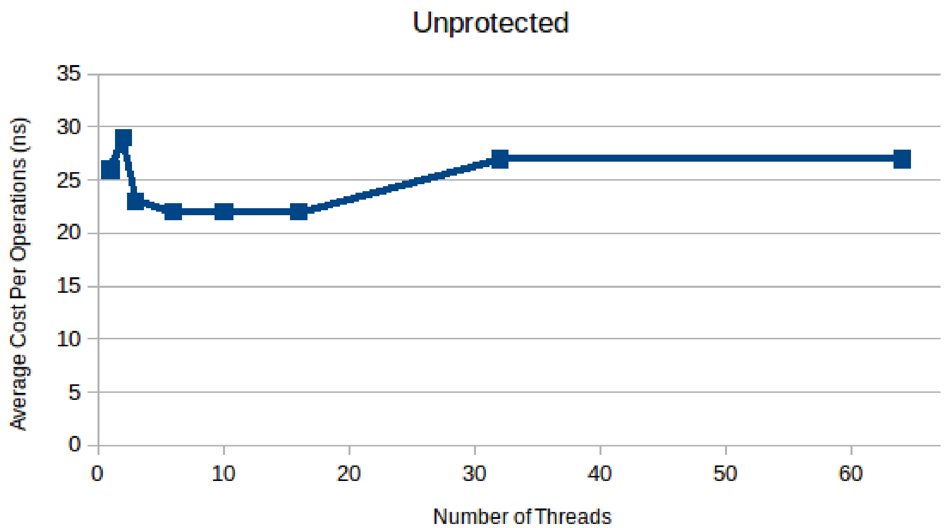
Number of Iterations	Average Cost Per Operations (ns)
5	62342
10	24743
15	19399
30	10549
60	4873
150	1574
1000	303
5000	51
10000	49



The other runs were conducted with yield on and iterations at 10,000.

Unprotected

Number of Threads	Average Cost Per Operations (ns)
1	26
2	29
3	23
6	22
10	22
16	22
32	27
64	27



QUESTIONS 1.2

1. The average cost per operation drops with increasing iterations because the ratio of work done compared to thread operation. Economies of scale show that the time for creating a thread becomes a smaller ratio of the overall cost for operations for increasing iterations.

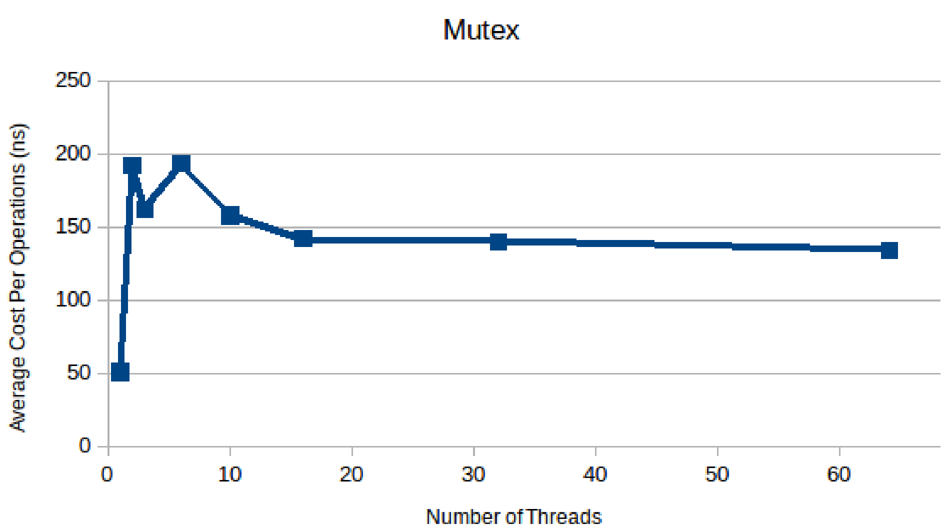
2. The "correct" cost is the overall cost for operations subtracting the time taken for thread creation.

3. --yield runs much slower because of context switching; threads have to wait as the CPU switches from one thread to another.

4. Using --yield, we can not get valid timings because we cannot tell how much time was spent switching between threads. We cannot isolate or time when that happens.

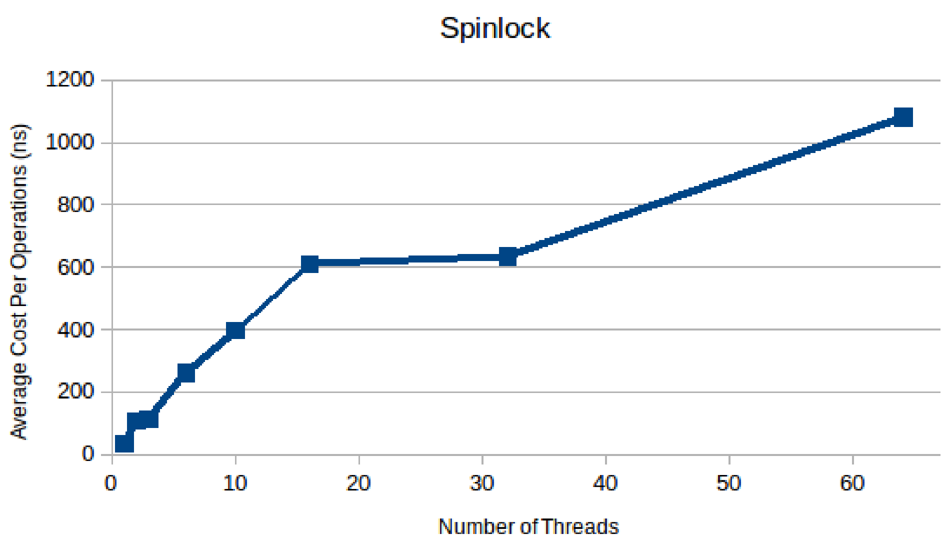
Mutex

Number of Threads	Average Cost Per Operations (ns)
1	51
2	192
3	162
6	193
10	158
16	142
32	140
64	134



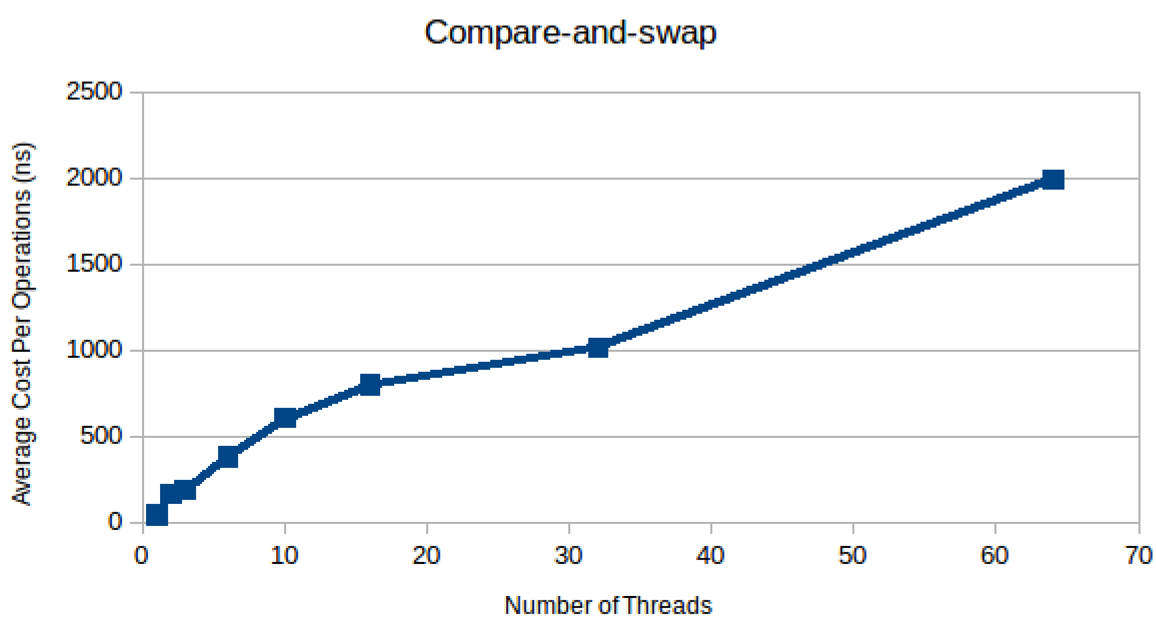
Spinlock

Number of Threads	Average Cost Per Operations (ns)
1	34
2	109
3	114
6	260
10	396
16	611
32	635
64	1080



Compare and swap

Number of Threads	Average Cost Per Operations (ns)
1	46
2	168
3	188
6	382
10	607
16	799
32	1016
64	1991



QUESTIONS 1.3

1. All options perform similarly for low numbers of threads because there are only so many threads to grab locks or make comparisons. With fewer threads, there is less waiting for each thread to do so.

2. The three protected operations slow down as the number of threads increases because more threads wait to grab the lock, or go through more comparisons.

3. Spin locks are expensive for large number of threads as there is more contention for the spinlock; the threads do busy-waiting. Whenever a thread has its turn, it waits in a loop, taking CPU time to continuously check the lock.

QUESTION 2.1

Variation in time per operation vs number of iterations:

Correcting this effect:

QUESTION 2.2

Variation in time per protected operation vs number of threads

QUESTION 2.3

The change in performance of the synchronized methods is a function of the number of threads per list

Threads per list is a more interesting number than threads

QUESTION 3.1

1. **The mutex must be held when pthread_cond_wait is called because pthread_cond_wait unlocks the lock and block on a condition variable. The mutex can be held, unlocked by the call, and acquired by the calling thread thread, so that the original thread gets it after.**
2. **The mutex must be released when the waiting thread is blocked else it will result in deadlock. There may be more threads attempting to acquire the lock.**
3. **The mutex must be reacquired when the calling thread resumes so that the condition variable will not be touched by other threads.**
4. **This must be done inside of pthread_cond_wait so to prevent race conditions. Another thread could change the data before the thread can block. Pthread_cond_wait could also be stuck in an infinite loop for that reason.**

5. **This can only be done outside of a user-mode implementation of pthread_cond_wait. A system call is used to release the mutex and access the condition variable, which requires privileges from the kernel.**