Alan Zheng
301189136

Assignment 3 Report

This report analyses my implementation of three locks: a simple spinlock, an exponential backoff, and a queue lock. Each implementation is recursive so I have included tests to support both the iterative and recursive usage of each of the locks. For instance, for the spinlock, there is a test for mySpinLockTAS (iterative test method using recursive lock) and myRecursiveSpinLockTAS (recursive test method using recursive lock). All tests are performed in an environment with multicore processors.

My main priority was ensuring the results are correct. For each of the tests, the iterative and recursive test methods' expected total count is equal to numItterations * numThreads. For the rest of this report, when I refer to parameters, I am talking about the variables under test, which includes threads, number of iterations, operations outside critical section and operations inside critical section. The numItterations variable is fixed at 130000 initially to avoid segmentation fault when using the recursive test methods. This is due to the stack overlapping with the heap.

The measurements are taken based on a few considerations. I want to understand how all locks compare relatively to each other given fixed parameters. For each of the parameters, I will increase it one at a time while leaving all other parameters fixed at the default value and perform an analysis on the behavior. The default values are set to (4, 130000, 0, 1) for number of threads, number of iterations, work outside critical section and work inside critical section respectively.

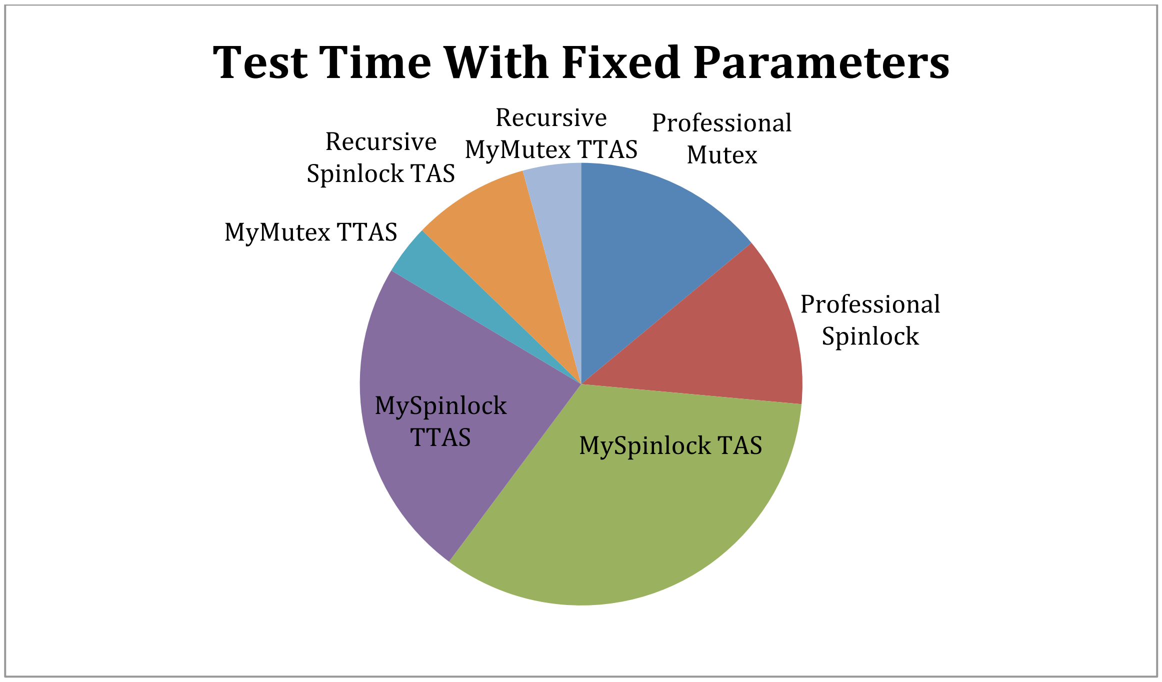
A comparison is made between all test methods when parameters are fixed at a certain default value. Figure 1 shows that regardless of which lock is used (spinlock/mutex locks), a recursive approach take much less time to increment a counter than an iterative one. This is due to caching of instructions for that function. MySpinlock TTAS performs better than MySpinlock TAS as a value does not need to be fetched from main memory every time. Instead, the value can be read directly from the cache. However, MySpinlock TTAS does not perform as well as MyMutex TTAS because there is still contention for the lock. The problem is that after the lock is released, there is an invalidation storm. Upon all of those threads' next access to their cache, they will encounter a cache miss. This results in higher traffic on the bus as each of the threads attempt to acquire the new value from main memory. MyMutex TTAS implements a backoff strategy, which attempts to avoid the prior situation of bus contention from happening, as can be seen from the results. The threads are backed off for a random period of time before trying to acquire the lock again. Thus, MyMutex TTAS has a much smaller test time than the other locks.

When comparing lock performances after a 10x increase in the number of threads, I noticed a drastic difference in the percent increase. For the sake of avoiding a segmentation fault, I chose to leave out the recursive tests for this analysis. One would expect all the locks to increase by the same fixed percent, but Figure 2 shows that this is not the case and the spinlocks increase by more than 10x. In fact, for MySpinlock TAS and MySpinlock TTAS, they both have roughly a 62x increase in test time. This is not surprising given the exorbitant bus contention increase with the number of threads after the lock becomes free.

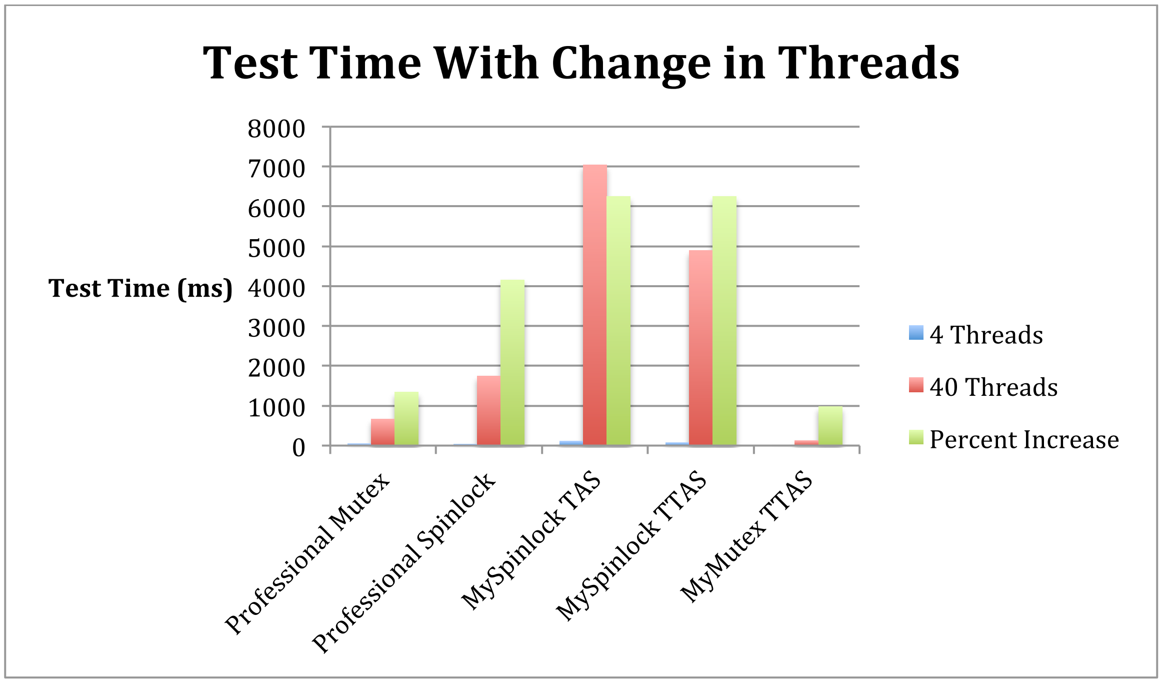
From Figure 3, by increasing the number of operations outside the critical section, I noticed that regardless of which lock is used, the test time increase are relatively the same rate. The difference is that the Professional Mutex, Professional Spinlock, MySpinlock TAS and MySpinlock TTAS perform roughly the same time in terms of performance while MyMutex TTAS has a slightly better performance than all of them. This allows me to conclude that that the professional mutex and professional spinlock is not implemented using the backoff strategy since MySpinlock TAS and MySpinlock TTAS does not use that strategy either.

Figure 4 and 5 shows that increasing the amount of work inside the critical section will take drastically longer to run the tests in comparison to increasing the same amount of work outside the critical section. For this test, I set the work outside the critical section to 3000 with all other parameters to defaults and comparing the results to the same amount of work inside critical section with parameters set to defaults. This result is consistent regardless of which locking mechanism is used. In terms of differences, MySpinlock TAS has the largest spike in performance (approximately 647% increase from time of operations outside critical section). These behaviors can be explained by the fact that when the CPU tries to write a value to main memory, there will be a huge delay, which is caused by the high bus contention (as explained previously). I also wanted to see whether a work increase from 3000 to 6000 will change the rate of increase, but as can be seen from Figure 5, this is not the case. The shape of the graph for Figure 4 and 5 is about the same, which means that an increase in the amount of work does not change the rate of increase for the test time with respect to all the locking mechanisms.

Alan Zheng
301189136

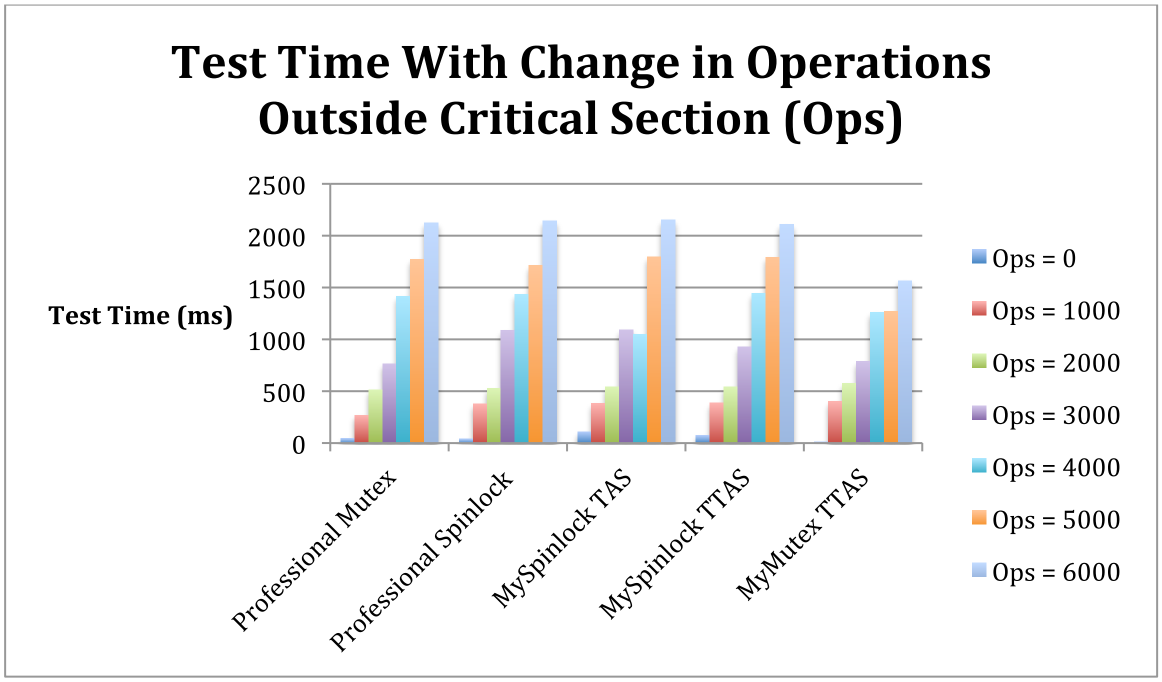


[Figure 1]

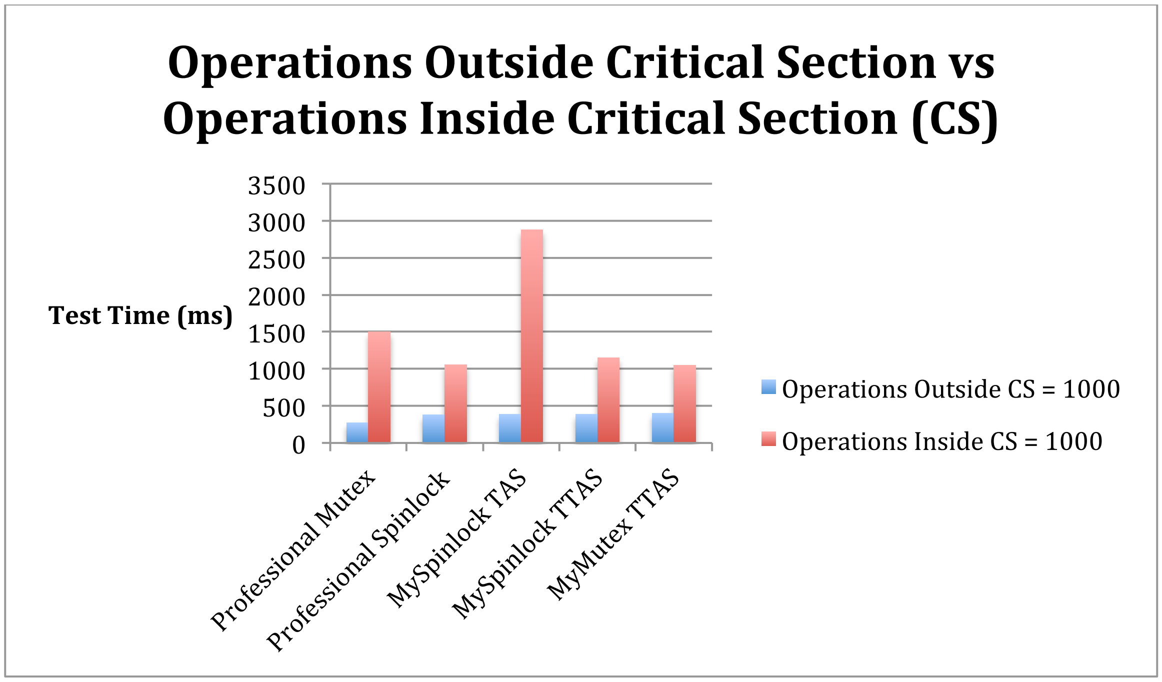


[Figure 2]

Alan Zheng
301189136

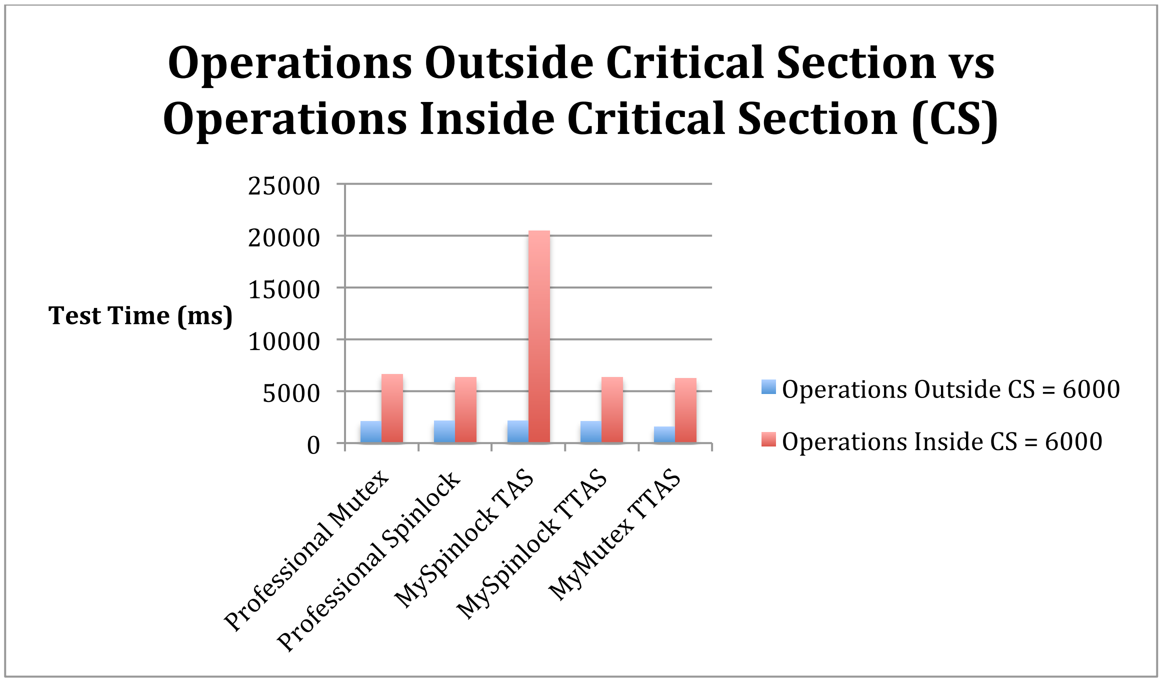


[Figure 3]



[Figure 4]

Alan Zheng
301189136



[Figure 5]