# Locks Report

**Methodology:**

Both my spin lock and mutex has the following:

1) volatile unsigned long “locked”; this was used as the lock variable.

2) long “owner”; this is used to keep track of which thread is currently holding the lock. This was necessary to make both locks recursive.

In all the implementations, the general concept was the same. Call TAS or CAS until lock has been acquired, *then and only then* set the owner to self’s pid. This ensures that there is ever only one thread setting the owner variable and thus it does not need to be atomic. As well, the unlock method *must* reset the owner variable *before* the lock variable to prevent deadlocks.

The mutex function required a usleep() function for the thread to back off for a period of time. This period of time was generated using the thread ID to ensure uniqueness.

**All** data was averaged from 10 tests, excluding any test sample that was more than 30% deviated from the average of the other 9.

The iteration count was kept at 1,000,000 because the trial counts were large enough to eliminate a good amount of margin of error.

Thread count per trial set:

1 thread (no contest); 2 threads; 4 threads (CPU physical core count); 8 threads (CPU logical core count); 12 threads; 16 threads; 20 threads; 32 threads; 64 threads (but half the iteration, to compare with 32 thread test set)

**Sources of Error:**

CPU temperature may cause later tests to slow down. Thus, a sleep function was added between each test to make sure the CPU had enough time to rest. This function was removed later because it is not worth waiting for any test less than the most extreme, e.g. 32 threads with n = 1,000,000.

**Interpretation and Insights**

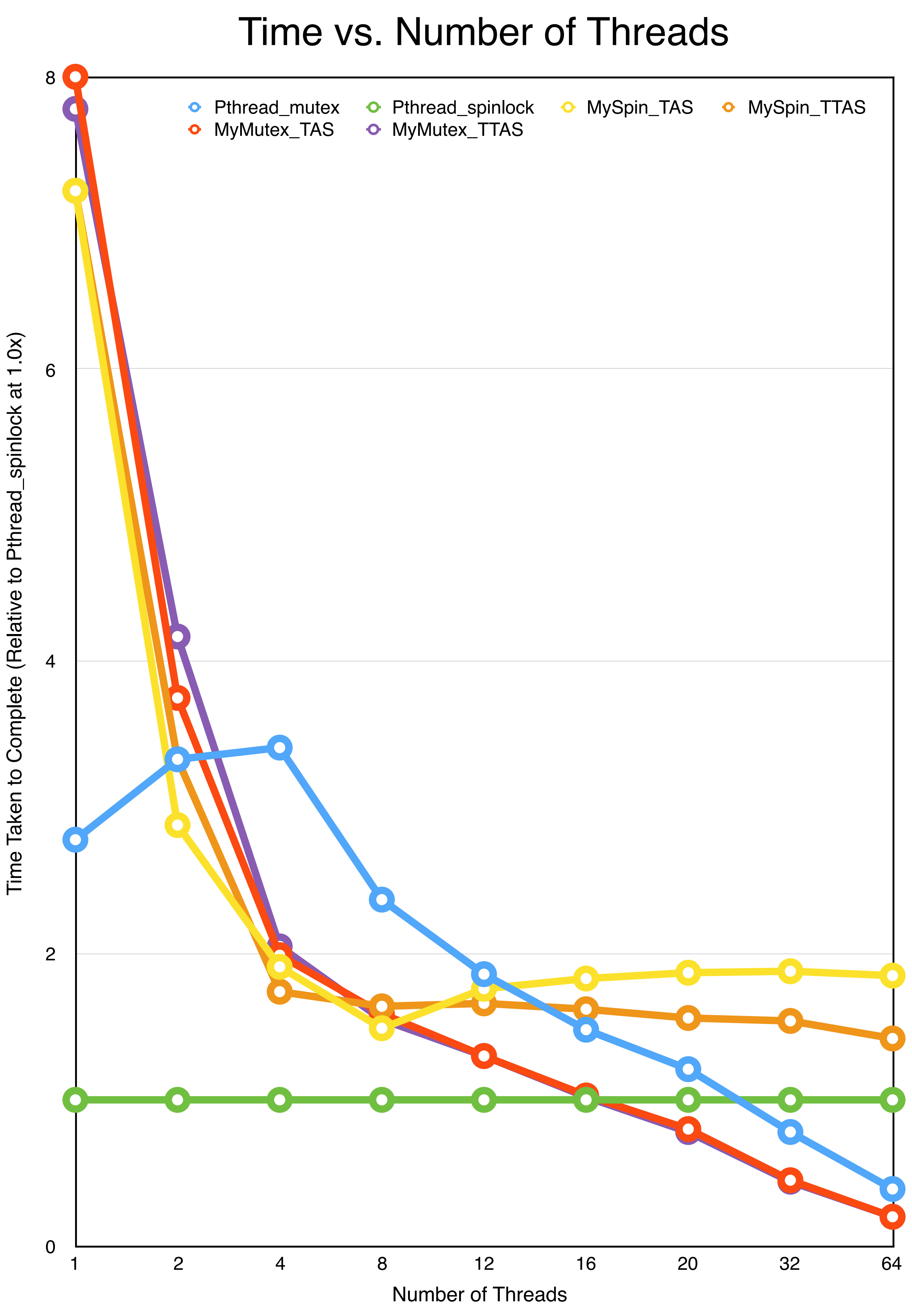
It is notable that the pthread implementations, especially the spin lock, is very consistent across all tests, while the my mutex one varies up to 40% between trials in the lower thread count tests.

However, once the thread count was high enough, i.e. from about 12 threads, the mutex trials became very very stable. I suspect this is due to the nature of exponential back-off so that even if 20 threads are running, the CPU’s 8 logical cores can still handle everything because it no longer needs to spin all 20 threads.

The data is below, and there is a graph on the next page. Each test is compared to the pthread spin lock, and (0.5x) means **twice as fast as spin lock** while (2.0x) means twice as slow (or twice as long as spin lock).

| **Iterations = 1,000,000** | **Threads = 1** | **Threads = 2** | **Threads = 4** |
|---|---|---|---|
| **Pthread_mutex** | (2.78x longer) 25 ms | (3.33x) 160 | (3.41x) 870 |
| **Pthread_spinlock** | (1.00x base) 9 ms | (1.00x) 48 | (1.00x) 255 |
| **MySpinlock TAS** | (7.22x longer) 65 ms | (2.88x) 138 | (1.91x) 487 |
| **MySpinlock TTAS** | (7.22x longer) 65 ms | (3.33x) 160 | (1.74x) 443 |
| **MyMutex TAS** | (8.00x longer) 72 ms | (3.75x) 180 | (1.99x) 507 |
| **MyMutex TTAS** | (7.78x longer) 70 ms | (4.17x) 200 | (2.05x) 524 |
| | | | |
| **Iterations = 1,000,000** | Threads = 8 | Threads = 12 | Threads = 16 |
| **Pthread_mutex** | (2.37x) 1753 | (1.86x) 2608 | (1.48x) 3507 |
| **Pthread_spinlock** | (1.00x) 740 | (1.00x) 1404 | (1.00x) 2368 |
| **MySpinlock TAS** | (1.49x) 1100 | (1.76x) 2465 | (1.83x) 4336 |
| **MySpinlock TTAS** | (1.64x) 1212 | (1.66x) 2335 | (1.62x) 3829 |
| **MyMutex TAS** | (1.59x) 1179 | (1.30x) 1820 | (1.03x) 2433 |
| **MyMutex TTAS** | (1.55x) 1150 | (1.30x) 1831 | (1.02x) 2419 |
| | | | |
| **Iterations = 1,000,000** | Threads = 20 | Threads = 32 | Threads = 64 **Iteration = 500,000** |
| **Pthread_mutex** | (1.21x) 4402 | (0.78x) 6984 | (0.39x) 6783 |
| **Pthread_spinlock** | (1.00x) 3647 | (1.00x) 8980 | (1.00x) 17374 |
| **MySpinlock TAS** | (1.87x) 6820 | (1.88x) 16887 | (1.85x) 32186 |
| **MySpinlock TTAS** | (1.56x) 5707 | (1.54x) 13849 | (1.42x) 24656 |
| **MyMutex TAS** | (0.80x) 2920 | (0.45x) 4041 | (0.20x) 3551 |
| **MyMutex TTAS** | (0.78x) 2827 | (0.44x) 3951 | (0.20x) 3525 |

The graphed version is next page with colours. Notice that TAS mutex (red) and TTAS mutex (purple) overlap each other as thread count increases. This is because they are relatively the same. Both are excellent because they have exponential back-off. On the other hand, spin lock TAS and TTAS (yellow and orange) are advancing at the same rate as the pthread spin lock, becoming very very inefficient as thread count increases.


Time vs. Number of Threads
Pthread_mutex
Pthread_spinlock
MySpin_TAS
MySpin_TTAS
MyMutex_TAS
MyMutex_TTAS
Time Taken to Complete (Relative to Pthread_spinlock at 1.0x)
0
2
4
6
8
Number of Threads
1
2
4
8
12
16
20
32
64