Ran Pang

Sandhya Dwarkadas

February 18, 2015

Parallel and Distributed Systems Project 2 Report

locks and barriers

Please run "make" to generate "locks_test" and "barriers_test".

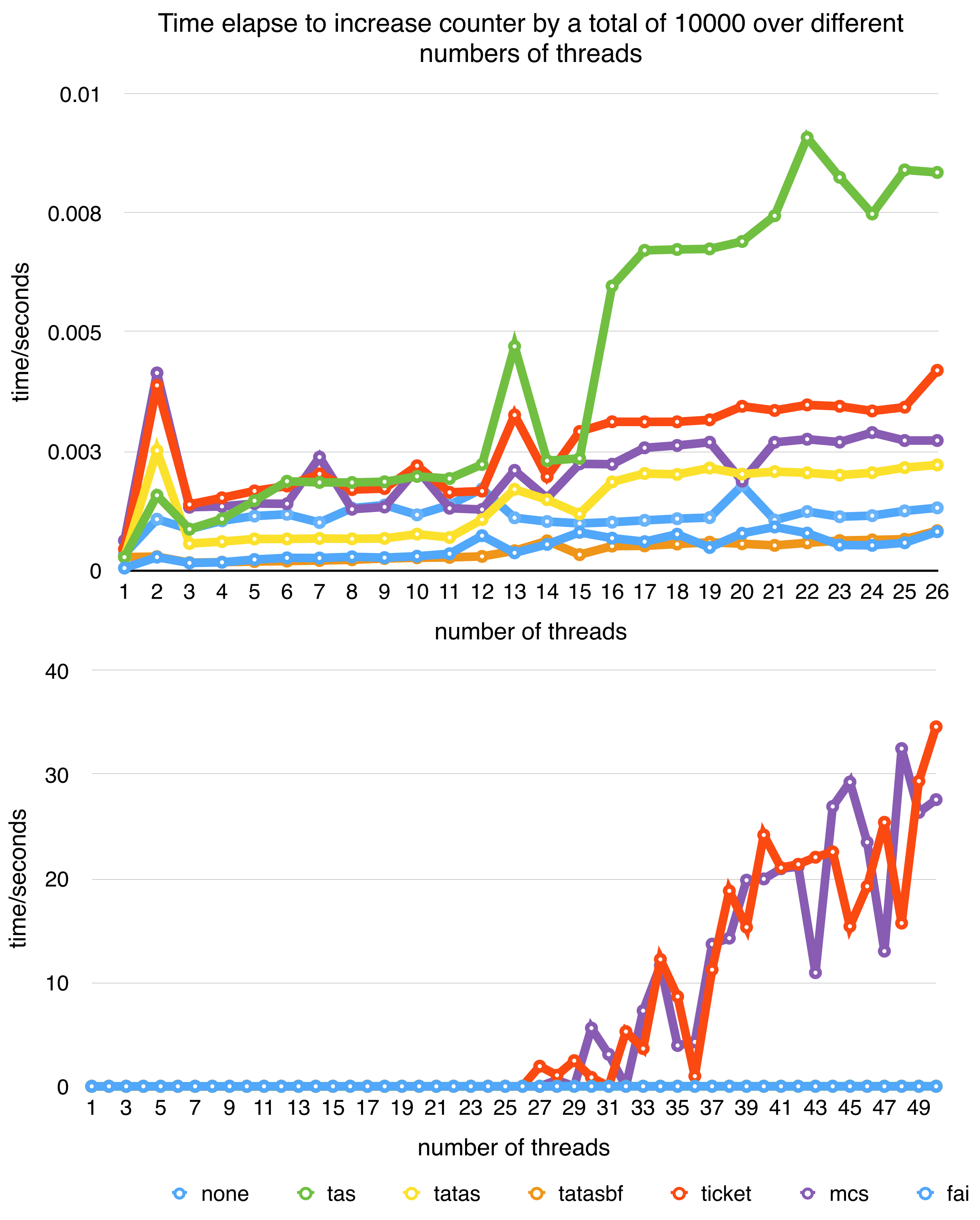
Please run "./locks_test -h" "./barriers_test -h" to see how to use the programs.

This project is meant to implement and test the performance of several types of locks and barriers and can cooperate with pthread. Locks include no lock, test and set lock, test and test and set lock, test and test and set lock with exponential back-off, ticket lock, MCS lock and fetch and increase without lock. Barriers include pthread-based barrier, centralized sense-reverse barrier and tournament barrier. The algorithms of these locks and barriers are provided by the assignment description thus not shown here. This report focus on the comparative performance of these locks and barriers with some argument on the possible reasons that make the difference.

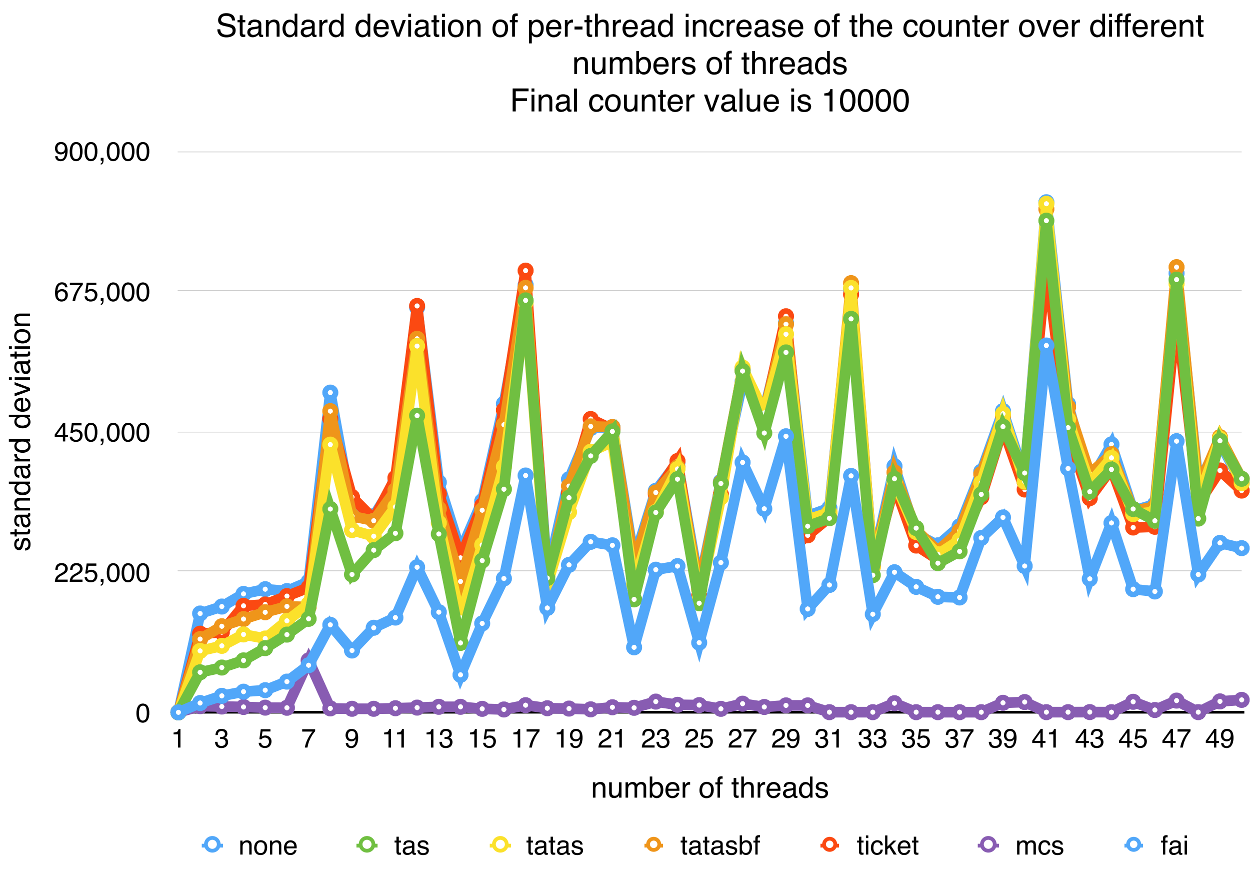
The environment specifications for the experiment is shown in the table below:

machine	cycle3.cs.rochester.edu
OS	Linux 3.17.7 x86_64
processor model name	Intel Xeon CPU E5-2430 2.20GHz
number of processors	24
number of cores per processor	6
cache size	15360 KB
memory available	63127064 KB

Performance of the locks over different number of threads is shown in the two figures below:



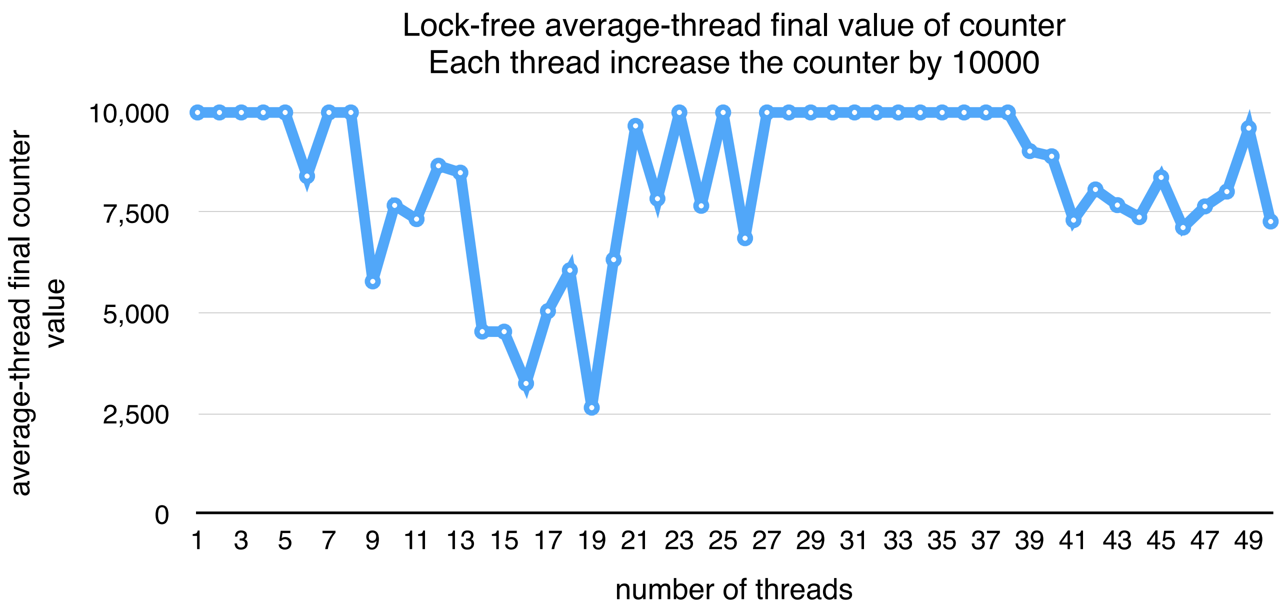
These results suggest that when the number of threads is less than the number of processors(24), test and set lock scales the worst while test and test and set locks with exponential back-off scales as good as no-lock. As the number of threads exceeds the number of processors, ticket lock and mcs lock explode in terms of time consumption while the other locks remain relatively stable.



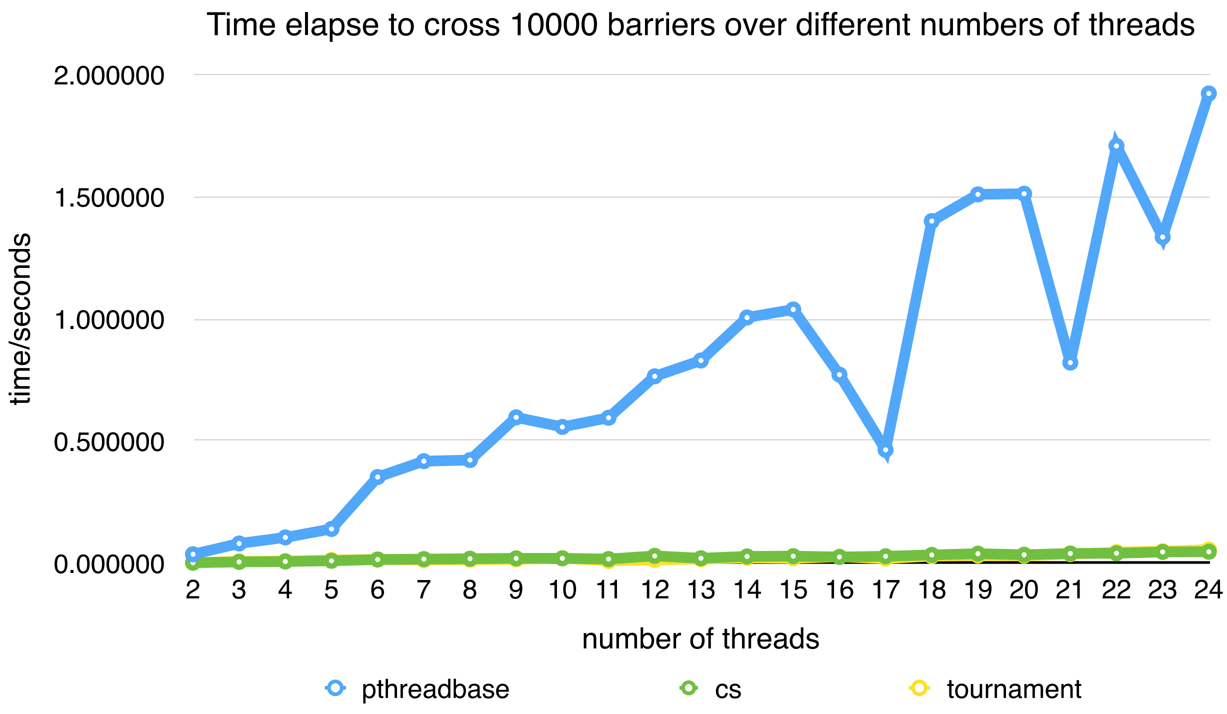
The fairness of the locks over different number of threads is shown in the figure below:

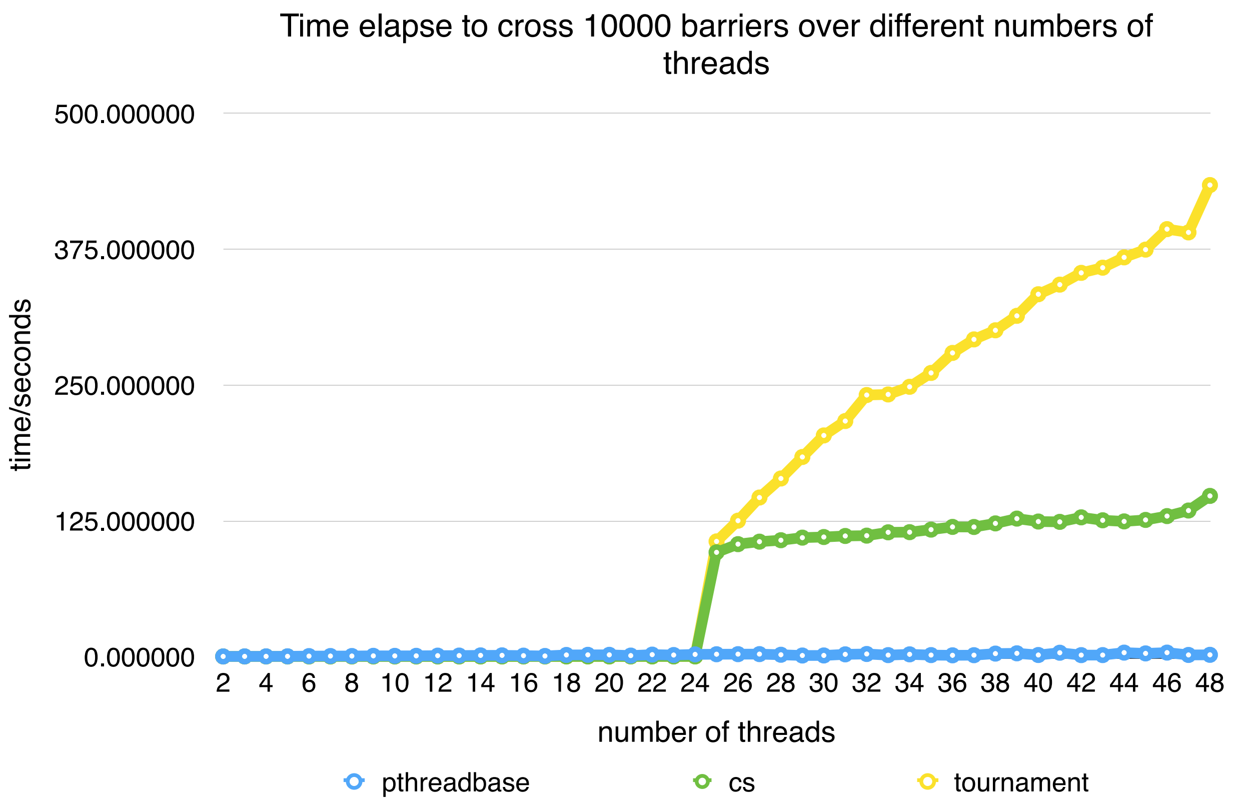
As we can see, mcs lock, among others, enjoy the best and most stable fairness throughout different number of threads. Although the fairness is not so stable for other locks other different number of threads, it indicate no significant general influence of the number of thread on fairness of all locks.

When each thread is let to increase the counter for a given number of times, all locks except for no-lock generate the correct final counter value. The final counter value of no-lock over different number of threads is shown below:



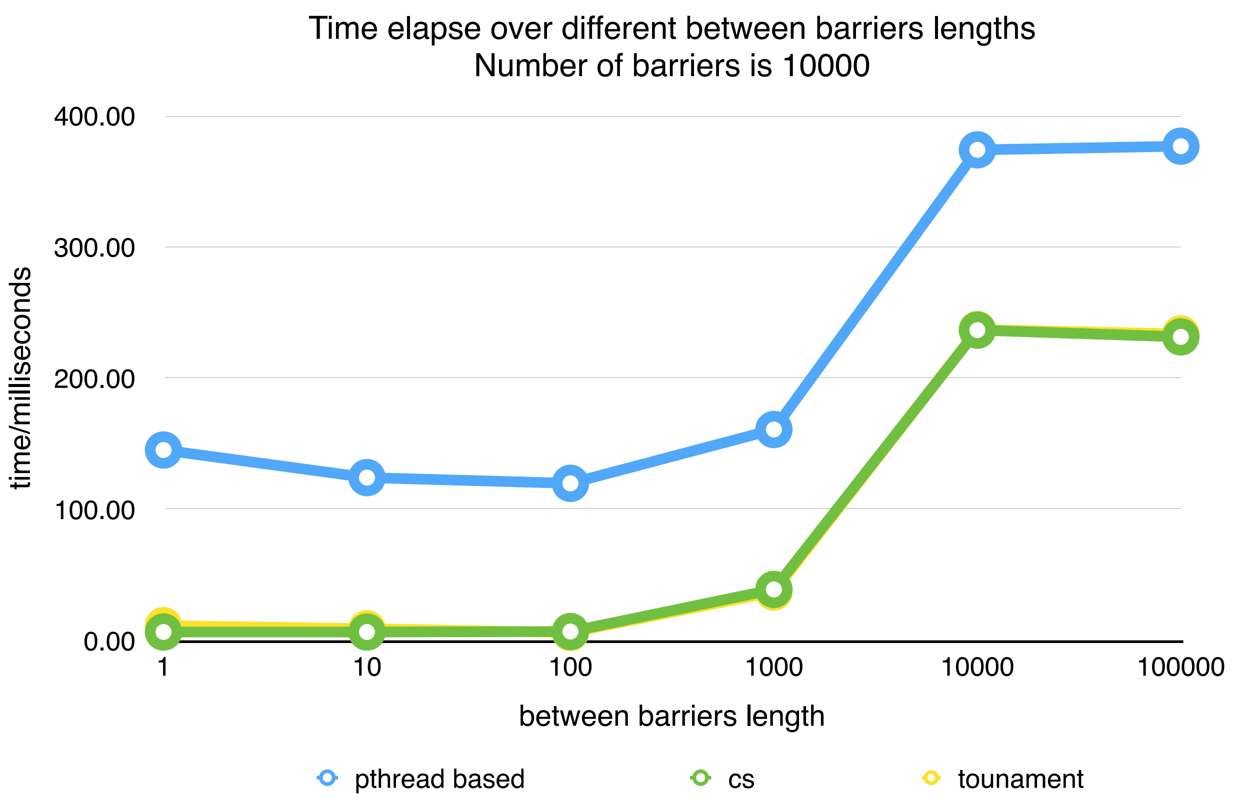
Performance of the barriers over different number of threads is shown in the figure below:





The results show that when the number of threads is less than the number of processors, pthread-based barrier is much worse in scalability than the centralized sense-reverse barrier and the tournament barrier where the latter two barriers exhibits quite a good scalability. However as the number of threads exceeds the number of processors, pthread-based barrier keeps a relatively stable time consumption while centralized sense-reverse barrier makes a rigged jump to a higher level of time consumption and tournament barrier keeps increasing its time consumption at a relatively high speed.

In the end, the influence of between-barrier execution length on the performance of the barriers is shown here:



As is shown in the figure, pthread-based barrier sees a slight increase in efficiency as between-barrier length rises within a low level, while the other two barriers keeps stable in performance. Such bonus disappear as between-barrier length continues growing for all three barriers.