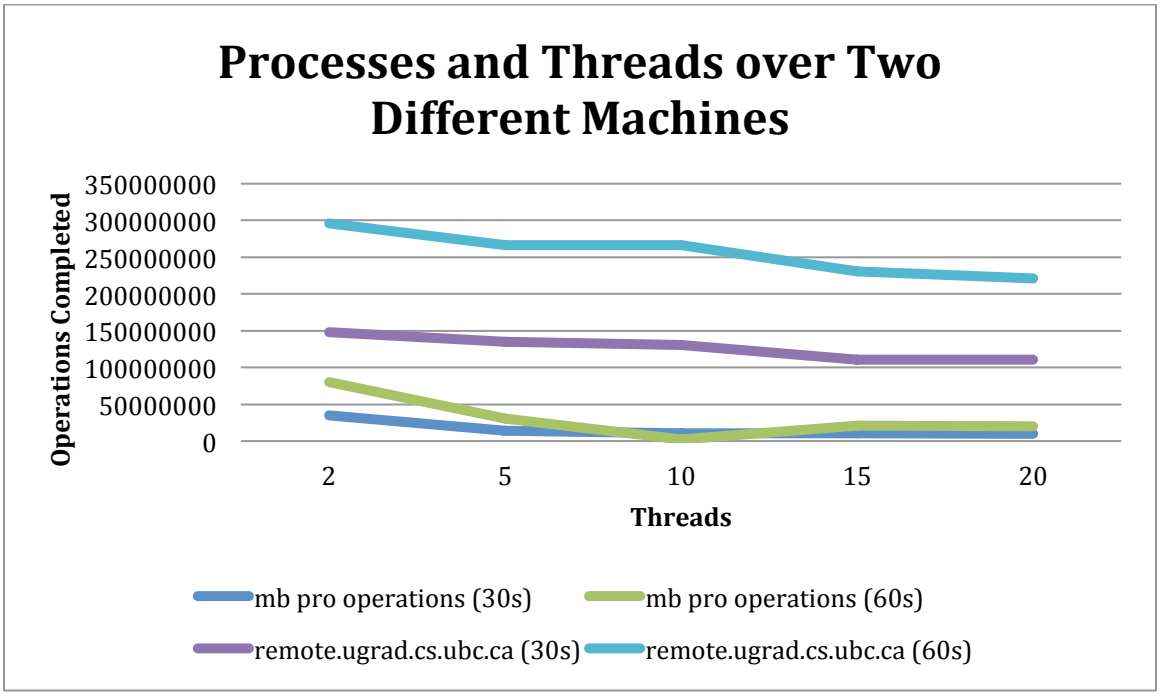
CPSC416 Assignment 2
Michael Wu 71163091



1. The thread-safety mechanism used in this benchmark implementation included two mutex locks from the C library pthreads. One lock was used to guarantee that any given time, a single thread could at most be in the motion of queuing an item, or could be in the motion of dequing an item from the shared queue. This meant that threads could spend quite a bit of time waiting for a lock to be retrieved to allow it to perform actions on the queue. Because there was only one lock used for the queue, this also meant that a thread could be waiting to simply dequeue an item, even though a thread which currently owned the lock was simply in the process of queuing an item. Independent actions, but this could lead to tricky edge cases without careful handling on a shared structure. Another lock was used to ensure that reading and writing of the operation count variable was also thread safe and non-conflicting with others.
2. In theory, the more threads a problem receives, the better performance can be achieved since operations can run in parallel. However, there is an inherent bottleneck within this shared queue – both operations of a queue and dequeue require a thread to exclusive access to the lock to modify it. Therefore, by adding more threads in an attempt to execute workloads in parallel, the threads, as mentioned above, may suffer in performance since every other thread will be waiting for the thread holding the lock to finish the queue or dequeue operation, even if they are independent. The threads will actually be bound in performance by the speed in which context switching can occur between each process. There may be modest gains leaping from one thread to two, but overall, threads will suffer from the blocking that occurs as they each attempt to access and acquire the lock.