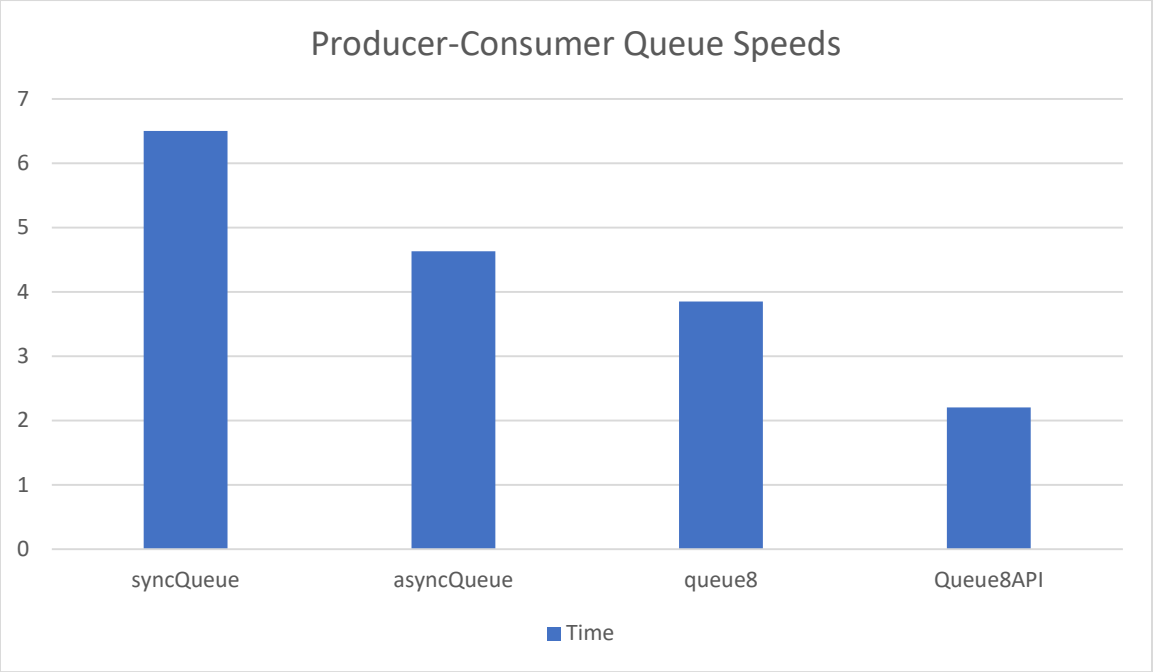
Homework 3 Write-Up

Galen Turoci

CSE 113

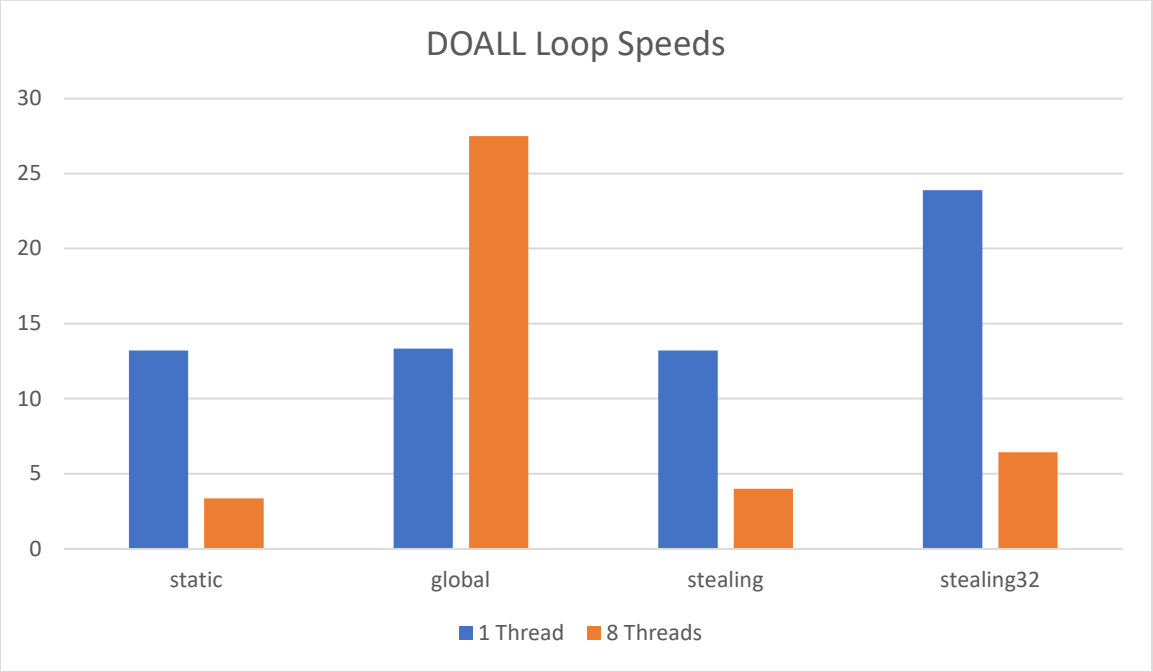
Part 1



Presented here are the timing results of 4 different producer-consumer queues. For reference, I am running this on an 8-core 3700MHz processor and 16GB of RAM. With the exception of syncQueue, all of these results are based on the averages of running each implementation 10 times (syncQueue was run thrice). It is worth noting that asyncQueue and queue8 varied wildly in their timings, sometimes running in 1.5 seconds and sometimes running close to 9; additionally, both of these implementations deadlocked about ¼ of the time, though I cannot be certain as to why. Initially I thought perhaps it had something to do with my CQueue.h implementation, but if that were the case, queue8API also would have deadlocked, and it did not (unless I was simply lucky). This leads me to believe it is something about the way the threads are communicating in main.cpp & main3.cpp, however I was provided main.cpp and have not modified the functions wherein both threads are operated on; as for main3.cpp, it is possible I implemented it incorrectly, despite my best efforts.

Most of my implementation largely relied on closely following Professor Sorenson's May 6th lecture, though I did have to get a bit creative to come up with a solution for how to compare the head of the queue to the tail when it reaches CQUEUE_SIZE; the solution I came up with is a bit ugly but it gets the job done as far as I can tell. For enq_8 and deq_8, I originally had them doing the enqueueing and dequeueing in-house before I realized I can just pass the work off to enq and deq respectively, and it would make the solutions easier to both implement and read.

Part 2



Presented here are the timing results of 4 different DOALL parallel loops. For reference, I am running this on an 8-core 3700MHz processor and 16GB of RAM. My results are… surprising to me. I would not expect the static loop to be the fastest of them all, and I would not expect the global loop to be by far the slowest, especially with multiple threads. While I would expect that stealing would be faster than both static and global, I am not surprised that it is also faster than stealing32, as when looking at the amount of work done by each algorithm, stealing32 is the same as stealing, but with much more unnecessary work to do that just adds overhead (unless my implementation is somehow very wrong, but if that is the case I cannot see how that is). I am also not surprised that, excepting stealing32, the single-threaded implementations of these loops all take basically the same amount of time, since they're all performing the same task, this time without parallelization.

Like with Part 1, much of my implementations closely followed Professor Sorenson's May 11th lecture. Initially, I was unaware that the IOQueue was not "circular" in nature like the queues in Part 1, and that did cause me some trouble, but I'm afraid that aside from that there isn't much that's interesting about my implementations. My deq_32, like my enq_8/deq_8 from part 1, simply calls deq() 32 times rather than doing the dequeueing in-house, making implementing it a trivial matter. My only concern with my implementations is again that I have somehow implemented deq_32 incorrectly, however if I have I cannot see how.