CS515 Parallel Programming: Assignment #1

Due on April 20, 2016 at 11:59pm

Jingke Li Spring 2016

Konstantin Macarenco

Quick sort summary
The main complication of Quick Sort over Producer Consumer is that Producer consumer has only one Producer and many Consumers, whereas this Quick Sort implementation requires many Producers to many Consumers.

Termination condition:
Task queue being empty as termination condition, this is the most obvious solution, however it has a major drawback (as mentioned in the assignment):
Race condition with the quicksort method (Producer). It is possible to have a situation when queue emptiness is checked when quicksort is started but not yet added any tasks in the queue, in this case workers exit prematurely.
Another termination condition is needed. We have two options,

1. Global variable to count producers
2. Special task that indicates that no more tasks will be produced, for example a task with negatives values in place high and low.

Later won't work since qucksort, unlike simple producer-consumer, builds binary tree, which size will vary depending on the size of the array. Each node will create "final" task, that only indicates that this particular node is finished but not necessarily the entire sort, and each node has no knowledge about the tree size.

On the other hand, a global variable (numProducers) that holds number of active quicksorts solves this problem. Each time before quicksort call the variable increased by 1, and decreased when branch dies, so we always know if sort is over $numProducers + queueIsEmpty$ is enough for workers to terminate correctly. There is one drawback in using shared variable, it creates additional bottleneck, since we have M producers updating it, and N workers reading - every update and read must be lock protected. This is not as noticeable, since the queue itself is a bottleneck.

Blocking vs non Blocking queue
Another important decision is whether to make workers sleep, or spin wait if there is no tasks available. This problem doesn't apply to producers, since we make an assumption that queue has "unlimited" size.
Sleep has one potential pitfall - last qsort is still in session but it already queued it's task, which was consumed before qsort returned, hence this will put all threads that are in the middle of this operation into infinite sleep. Another disadvantage is that since qsort never sleeps there is almost no wait on the task queue, performing context switch on a worker is more expansive than spin wait in this case.

Other Problems
I had hard time synchronizing threads, since at least one termination condition must be unsatisfied at the start. I ended up setting $numProducers$ to 1 in main() so all workers will spin wait while queue is empty.

Another problem is debugging, it is becoming extremely to hard as number of threads is increasing.

Producer consumer

In this problem I used standard producer-consumer solution with blocking queue, with a special task as a termination condition, that producer adds to the queue as soon as all task are produced. In this case it is a task with different high and low values. As soon as the task is in the queue producer signals one of the consumers and quits.
Each consumer will get the last task, add it back to the queue, and signal the next consumer.
One advantage of using the queue over a global variable as a termination condition is that it doesn't add any additional penalties since each consumer already locks the queue to get next task.