Part One: a brief explanation describing how my changes ensure mutual exclusion

1, In main() function:
I use nested parallel regions to assign multiple threads to two sections, one is for consumer(), the other is for producer().

```
omp_set_nested(1);
 #pragma omp parallel num_threads(2)
{ if (omp_get_thread_num() == 0){
   #pragma omp parallel num_threads(nconsumers) shared(out, itemCount)
   { // Task 0 for consumer() }
 }else{
   #pragma omp parallel num_threads(nproducers) shared(in, itemCount)
   { // Task 1 for producer() }
  }
}
```

In this part, I also use ***#pragma omp parallel shared(in, itemCount)*** and ***#pragma omp parallel shared(out, itemCount)*** to ensure that in and out, itemCount is global in different threads.

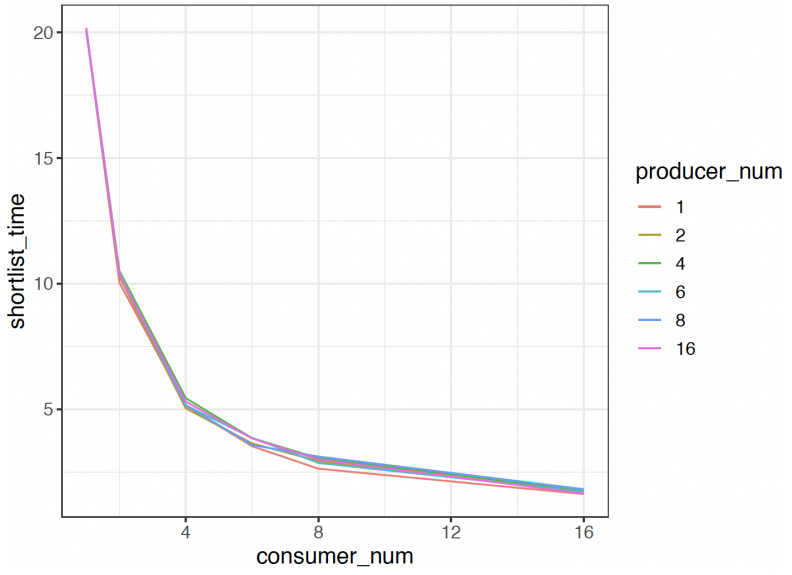
2, In consumer() and producer() function:
I use ***#pragma omp private(number, producerno)*** and ***#pragma omp private(number, consumerno)*** to make sure that the number and producerno, consumerno each has a copy and different specific value in each thread.
I use ***#pragma omp for*** to ensure loop worksharing, openmp will assign the jobs in the loop to each thread equally.

3, In insert_data() and extract_data:

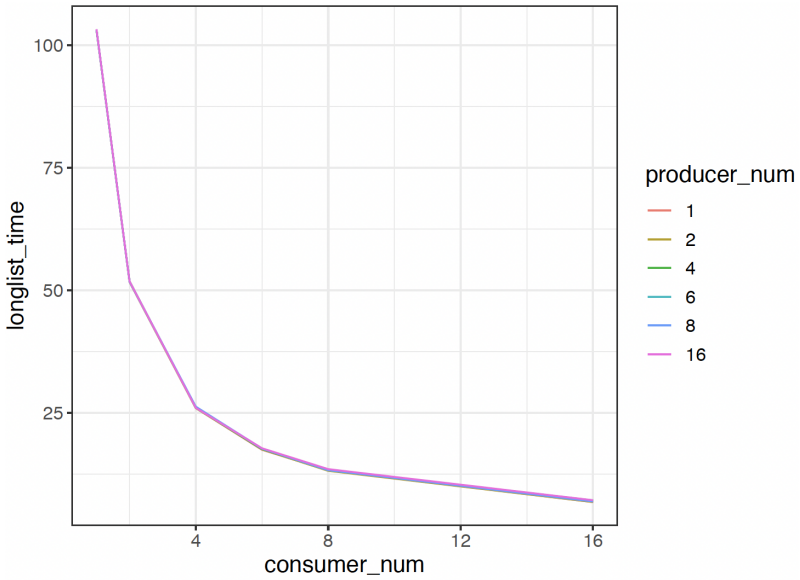
itemCount indicates the number of elements in the buffer, inserting will add 1 to itemCount and extracting will minus 1 from itemCount.
In insert_data function, I use ***while(itemCount == MAX_BUF_SIZE) {;};*** to ensure that it will wait for inserting if the buffer is full.
In extract_data function, I use ***while(itemCount == 0) {;}*** to ensure that it will wait for extracting if the buffer is empty.

I use ***#pragma omp critical*** to ensure mutual exclusion while inserting or extracting data from the buffer because **critical** will make sure that only one thread at a time processes that block.

Part Two: the scalability of my parallel solution



(A)



(B)

I select **1 producer and 8 consumers** as the best.
From the figure we observed that **the increase of producers numbers does not affect the processing time**, because the inserting process costs very little time compared to the consumer process, **so I choose 1 producer with one thread processing the producing process**.
The increase of consumer numbers reduces the processing time. The scalability reduces as the number of consumers increases. I choose **8 consumers** with 8 threads in parallel because this is **a trade off between reduction of processing time and increase of threads**.
The unexpected result is that the increase of producers' numbers does not affect the processing time.