```
Yinghuas-MacBook-Pro:hw3 yinghuamo$ ./p9
pi_serial runtime:      0.202601        result :        3.458627
pi_atomic runtime:      0.706888        result :        3.456649
pi_critical runtime:    0.673379        result :        3.456649
pi_manualA runtime:     0.165563        result :        3.451014
pi_manualB runtime:     0.134797        result :        3.455625
pi_auto runtime:        0.134454        result :        3.456426
Yinghuas-MacBook-Pro:hw3 yinghuamo$ ./p9
pi_serial runtime:      0.198471        result :        3.458563
pi_atomic runtime:      0.701349        result :        3.457613
pi_critical runtime:    0.715201        result :        3.457613
pi_manualA runtime:     0.152212        result :        3.452054
pi_manualB runtime:     0.137310        result :        3.457126
pi_auto runtime:        0.137403        result :        3.455914
```

Here is the result of the experiment;
The ranking will be:
Automatic Reduction < Manual Reduction B < Manual Reduction A < Serial < Critical < Atomic

1.
The automatic reduction and manual Reduction B are fastest for they won't be blocked in order to wait for other thread to update the 'count' nor wait all threads to finish task and add up to the final result;

2.
Atomic and Critical are slowest for they have to wait and compete about the resource of 'count' variable;

3. I think there is a discrepancy between them. The major difference between these two method is the Manual Reduction A will wait all the threads done with their tasks and add up the partial result in a serial adding. That takes time.

4. Well, actually, after several times of experiment, I did not notice a big discrepancy between Atomic and Critical. Sometimes Atomic is faster, sometimes Critical is faster. But theoretically, atomic should be faster than critical.

About Shared Pi with pthreads:
Well, according to the experiments above, I would like to use the Manual Reduction A generally, but with a vector called 'ready[nthreads]' to see whether the corresponding thread has finished or not (it works like conditional variables). After creating all the pthreads and I will have a 'while' loop to check through the 'ready' vector, if 'ready[id]' is positive, then I would like to add up the corresponding partial count immediately and reset it to false. This 'while' loop until all partial counts have been added to the final count.