Report

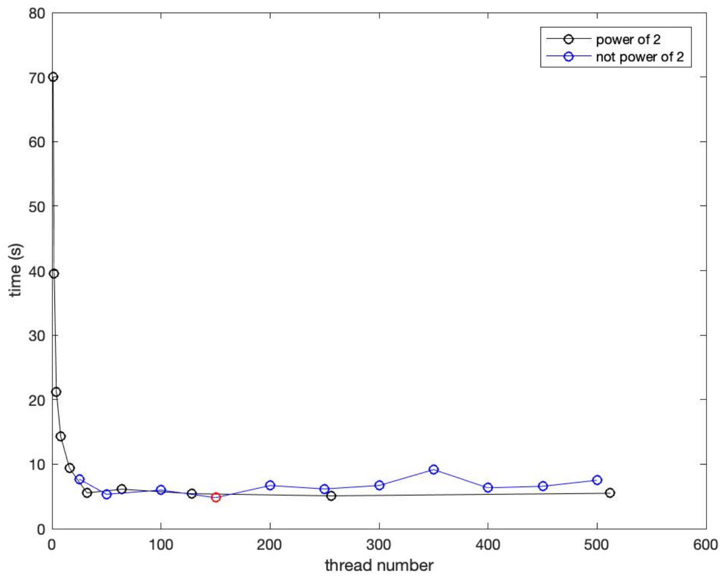
b04202008　韋彥丞

A. As codes and Makefile

B. I use threads on two parts.

1. When building random forest, I use threads to build different decision trees simultaneously. In each run, I first assign each thread a root, which is for one tree. And then I wait for its result when I need to assign new tasks to this thread.
2. When I want to use random forest to predict result of test data, I use thread to assign different test data to different thread at the same time. In order to pursue better efficiency, I want to avoid creating and joining threads too often. Thus, I maintain a read buffer (Capacity 4096 test data) and divide these data to different threads equally. For example, if there are 256 threads, each thread gets 16 test data and calculate them. After reclaiming all data from all thread, I re-assign all threads again.

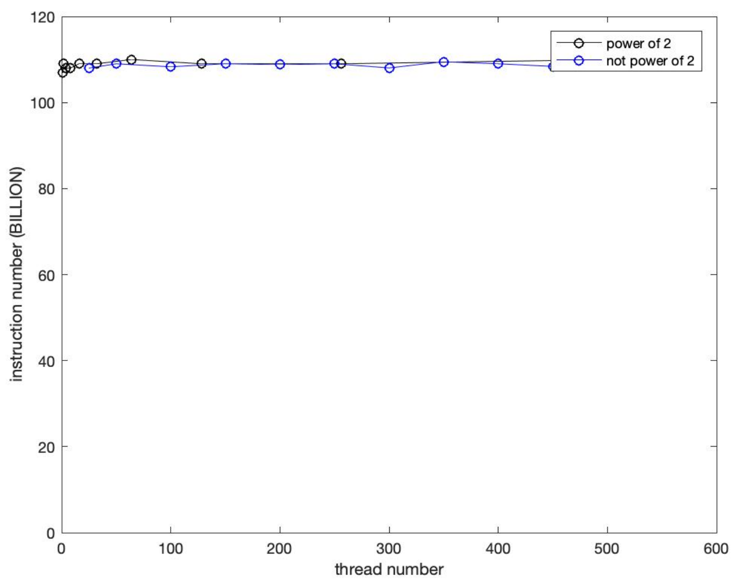
C.



The graph shows the experiment result, the relationship between real time and thread number. The black dots show the number that is power of 2, while the blue ones represent the number that is not the power of 2. The red dot is (thread num=150, time = 4.789), which is the fastest one.

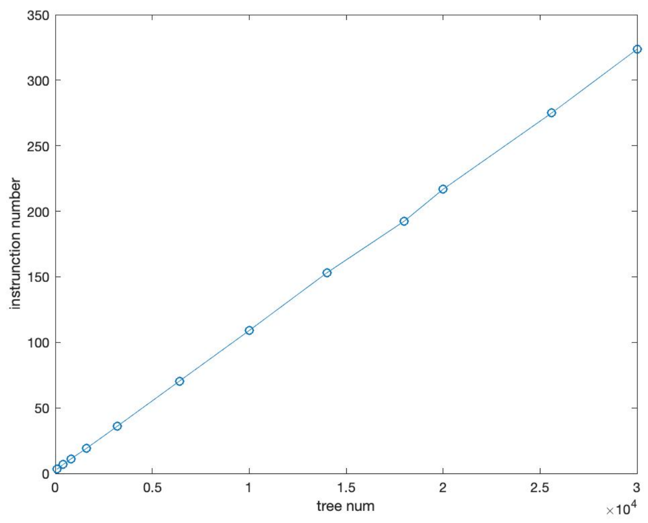
The time goes exponentially down in the beginning, which shows that multithreads indeed save times. However, when the number grows up, it exists a limit, which might be the bottleneck for the task that not the threads deal with. When the thread number keeps growing up, it seems that time goes up slightly, which might result from too many works on creating and joining threads.

D.



The graph shows the relationship between thread number and instruction number. The total instructions number almost remains the same. Even though opening numerous threads, the total workloads are the same. Thus, even it saves time, the total instructions cannot be saved.

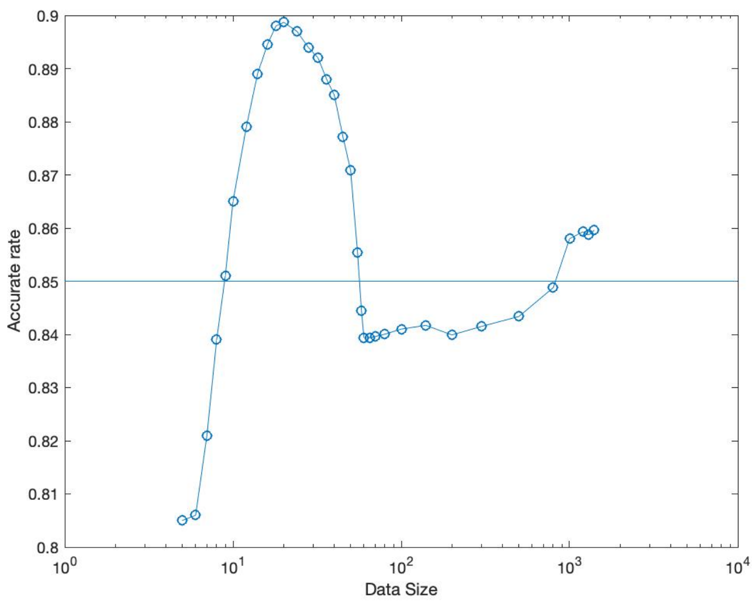
E.



The graph shows the relationship between tree numbers and instructions number. The results indicate that it is almost proportional. It is reasonable since more trees require more instructions to complete, and every tree requires almost the same tasks to finish it. Thus, it is linear, which is predictable.

F.

1. I record the accurate rate for different data size per tree. The result shows below.



The pattern is odd. In the beginning, it is reasonable that rate is low when data per tree is low. However, the accurate rate reaches maximum at around 30~40. After that, it goes down rapidly, and then rate rises slowly in the following. There is a limitation for testing data, since I cannot open too large data size to avoid compiling error.

2. I also try another method to implement multithread on testing data, but it fails in the end. I try to assign each reading test data to different threads one by one, but it turns out that too many operations must be taken for creating threads, which results in failure once reading too many test data.