The two repository I used is my repository in 17214 course names “kge” with the team repository in homework 05. The reason why I used these two repositories is they have several similar files, and both have hundreds of commits, which would significant difference in time consuming when the performance is improved.

The first strategy I used is paralyze the load process. During the loading process of each repository, there are hundreds of commits, and after storing all the commits and the parent & child commit pairs, I need to set the Similarity of each commit pair, in which case, I set 4 threads to paralyze this process. The results showed slightly improvement. The spent time decrease from 51.18 secs to 42.32 secs.

Then I used several threads to paralyze the process of building the queue of RevisionPair. During this process, we need to take all the CommitPair in both repositories to compare with each other, which is really time consuming. Then I used several threads to realize this process. In each thread, I wrote a producer class and a consumer class, which both implements the Runnable interface to realize the running process during building and polling the PriorityQueue of the RevisionPairs. To guarantee the safety of this process, the Queue and the running processes are all synchronized. However, this method did not show good advantages over the sequential method, which took about 40.17 secs.

Then I tried to use ExecutorService pool to paralyze the process of building the queue of RevisionPair. One problem is when I use a PriorityBlockingQueue to store all the RevisionPairs, the consumer must start to work after all the pairs have been inserted into the queue. To guarantee the completion of the ExecutorService pool, each thread should be shutdown and wait termination, which is really time consuming. It took about 150 seconds to finish this process, so it does not work. Then I changed the element in BlockingQueue from RevisionPair to Future<RevisionPair>, and take to get the result after loading all of the RevisionPair. It took 43.76 secs to finish this process. It did show great improvement comparing to the last one. But it still did not show advantages to the common thread in the second method, the results shown almost the same with the second method. It did make sense because the principle of these two methods are almost the same.

The speedup results:

$$S_{ParallelComparator} = \frac{L_{Sequential}}{L_{ParallelComparator}} = \frac{51.18}{40.17} = 1.28$$

$$S_{PoolComparator} = \frac{L_{Sequential}}{L_{PoolComparator}} = \frac{51.18}{43.76} = 1.17$$