Overview of sequential solution and project structure.

In my sequential program, I basically use for loop to iterate every branch, every commit, every parent of the commit, getting the revisions. After that, I still use for loop to iterate every revision in repoA comparing it to every revision in repoB for computing similarity.

So, I wrote an interface, GitRepo, and two classes, GitRepoParallel and GitRepoSequential implements it. Also, class DiffComposiiton is composed by parent and child SHA1 hash from a specific revision. Lastly, I wrote sequential and parallel similarity computation methods in similarityUtil.

Strategies to improve using parallelism

By inspecting the sequential program, it's not hard to notice that there are several tasks can be parallelized. As far as I know, the most complicated computation is revision comparison. As a result, I first revised this part. I mainly used producer-consumer design pattern introduced in lecture and recitation. I basically created other produce threads to parallelized handle every similarity computation between repos, and then use another consumer thread to collect results generated by producers. In terms of experiment, I initially used small repos, repo2 and repo50 to test if I got speedup. Surprisingly, parallel program was in average 5% slower than sequential one. I wondered maybe it's due to the overhead from parallelism. Then, I used bigger repos, two repo185 for test. I found the performance got improved by 16%(37s -> 31s). Thus, I came to a conclusion that in small size, using parallelism could get worse performance because creating and managing threads may be overhead. On the contrary, when it comes to larger size, it usually performs better using parallelism because computing took more time than overhead. Again, I used larger size of repo to test, namely repo350 and repo550. Using sequential program took 417 seconds, but parallel programs only tokk 52 seconds. What an improvement! It got speedup by about 85%. All in all, my experiments say that over the size grows, using parallelism got better performance.

I later did another tweak trying to improve performance even though I got speedup from parallelizing similarity computation. I made getting revisions from repo parallel. Again, by producer-consumer pattern, instead of sequentially iterating every branch getting every commit, I created a thread for each branch. In a such, I am able to deal with multiple branches getting revisions in a parallel manner. In terms of test, I used two of repos, which are repo185 and repo350. I respectively used both repos fetching revisions in different way. In repo185, I got 17.8 seconds in sequential program but 11.2 seconds from parallel's, about 37% speedup. As for repo350, sequential program got 2.4 seconds, but parallel one got 1.8, about 25% speedup. Obviously, this tweak works.

By combining the above parallel revision, I experimented and measured it again. Using repo50 and repo185, sequential program took 27 seconds, and parallel program took 21 seconds, about 22% speedup. Using repo185 and repo350, sequential program took 81 seconds, and parallel program only took 30 seconds, 62% speedup.

Lastly, to help with similarity computation, my program will compute a diffsMap composed of DiffEntry with its composition, namely parent and child SHA1 hash. Since parallelization works well, I'd like to revise my program again and make generating diffsMap parallel. After tests with repos, I found making this part parallel got no obvious improvement and sometimes even worse by around 7%. I speculated that it's also about parallelism overhead. As a result, I chose to delete this part, and keep similarity computation and getting revisions from repo parallel.

Archive concurrency safety

In this homework, for similarity computation, I usually used map to store some data calculated in advance. To achieve concurrency safety, I used ConcurrentHashMap to deal with concurrency. Also, when using producer-consumer pattern to parallelize tasks, I created an ArrayBlockQueue to perform asynchronization computation. Although arrayBlockQueue will be accessed by both producer and consumer, its characteristic can guarantee concurrency safety. In addition, when passing object to thread, I would use defensive copy to guarantee program correctness.

Why I chose my test repositories

In general, the reason I chose those test repositories is the amount of commits. I chose different commit number so that I can fully experiment my program in different data set. I've tried to test larger target(over 1000 commits) locally. However, it took too long to finish test. As a result, picking repositories in a reasonable size is important. Lastly, in order to fully control the repository and test correctness of my program, I created repo2 by myself. Thus, I can use it to test basic functionality. The rest of repositories are either existing repo at my computer or cloned from GitHub. I tried to find repositories in reasonable size and in different language. By searching keyword "java" and "javascript", I found a ~350 commits javascript repository and a ~550 commits java one.