Project Proposal

Abhiroop Kodandapursanjeeva

My project involves investigating the use of multithreading in mutation testing tools. The reason for this is to simply make the mutation testing process more time-efficient by utilizing more computing power. Initially, my idea was to improve the performance of existing open-source tools. However, after further research it seems all the popular open-source tools already implement this functionality. Therefore, it would be more worthwhile to instead build my own mutation testing tool, implement both sequential and parallel functionality, and measure the differences in performance. I believe it would be a great learning experience to build the tool, and would allow me to understand the underlying processes more in depth, giving me a better idea of which processes can be optimized using multithreading.

To make building the mutation testing tool simpler, I plan on using some existing open-source tools to help. Primarily, I plan on using JavaParser, which is capable of taking Java source code and producing an Abstract Syntax Tree. This would help greatly with the process of parsing code and generating mutants. This stage may be optimized by allowing threads to create mutants of different files simultaneously, or by allowing threads to create different mutants of the same file simultaneously. In the next stage, we have to run the generated mutants on the test cases. This can be optimized by allowing threads to split the test cases and share all mutants, or split the mutants and share all tests. An interesting question regarding multithreading for mutation testing is whether it is better to assign each thread an equal amount of work, or to allow threads to "pull" work from a common pool when they are ready for more work. The former prioritizes minimal communication between threads, but is bottlenecked by the slowest thread. The latter allows faster threads to perform more work, but may be prone to the inefficiencies of threads needing to perform atomic operations on the common pool of work.

To measure the performance of this tool, it would be sufficient to simply measure the time taken for the tool to execute. The measurements can be made separately for the first stage (mutation generation) and the second stage (running tests).