Thesis Proposal: **Optimizing Execution Time of Microbenchmark Suites using Benchmark Stability Measures**

Anonymous Student, Matr.-Nr. XXXXXX

As software systems increase in complexity, it becomes more important to measure performance changes in the application software using techniques such as application benchmarking and microbenchmarking. While application benchmarks treat the application as a black box and apply stress from the outside in order to evaluate application performance under realistic load, microbenchmarks operate on a more fine-granular level by repeatedly calling single functions on code level.

Both benchmarking types are expensive to execute, as application benchmarks need a (production-near) testbed of the software to run on, while microbenchmarks often vary in their execution time and need to be run multiple times in order to get repeatable, stable results. Existing approaches optimize microbenchmark suites by, e.g., reducing the number of microbenchmarks in a suite without affecting function coverage or practical relevance,[1] and trying to predict application performance changes using the execution of microbenchmark suites.[2] There is, however, still potential in optimizing execution times of microbenchmark suites by considering the stability of individual microbenchmarks. The stability of microbenchmarks often depends on factors, such as IO/filesystem access or the number of loops inside the function. Therefore, some functions deliver stable benchmarking results faster than others, and can potentially be executed fewer times without sacrificing result accuracy. This, however, needs to be tuned individually for each microbenchmark as their stability can vary.

In this thesis, we want to, (i) determine the best configuration for a given microbenchmark, i.e., its execution time, number of repetitions, number of iterations, etc., to obtain stable/reliable results, and (ii) find out how we can use that information to lower the total execution time of the whole microbenchmark suite without affecting their statistical results. In order to achieve this, we plan to start with the full microbenchmark suite and collect stability metrics during each iteration. These metrics can then be used to determine stable benchmarks in subsequent iterations and stop them early. The metrics can then also be used to further optimize repetitions and individual execution time, when the full suite is run again.

In order to evaluate these approaches, we will apply the implemented optimizations to existing open source projects with microbenchmark suites following current best practices. This allows us to compare the optimization to the current state of the art. In particular, we want to verify that (i) execution time of the entire suite is reduced due to our optimization, and (ii) results show no meaningful differences compared to the state of the art.

[1] Grambow et al., Using application benchmark call graphs to quantify and improve the practical relevance of microbenchmark suites

[2] Grambow et al., Using Microbenchmark Suites to Detect Application Performance Changes