CS6004 Assignment 4

Govind Saju
200070020

Virendra Kabra
200050157

April 2024

1 Introduction

The objective of this assignment was to perform an analysis via the soot framework on an input program, and utilise the analysis results to transform the class files. The modified class files are to then be bench-marked using the Eclipse OpenJ9 VM.

We have chosen to perform the null check analysis and subsequently remove redundant null checks from the class files. Our code for the same are in the following github repositories:

- Soot Analysis and Transformation - `https://github.com/virendrakabra14/CS6004_A4`. Branch `main`
- OpenJ9 Instrumentation - `https://github.com/govindsaju/openj9`. Branch `v0.26.0-release`

2 Methodology

In order to perform the analysis, we utilise the Nullness analysis provided by the soot framework. This is an intra-procedural analysis, hence not very precise. We perform the transformation by checking all null checks in the intermediate code, and if we encounter a null check which the analysis tells us is always null or non null, then the redundant if statement is replaced directly with a goto/nop statement.

In order to evaluate the effectiveness of this, we modified the openj9 VM's bytecode interpreter. The main way we counted the number of nulls and non nulls was by intercepting the ifnull and ifnonnull handlers inside the bytecode. We added a print (j9tty_printf) statement inside this to log the function (context) in which this statement was called, and utilised this to log the total number of calls to these functions and the number that was actually null / non null.

A point to note here is that a large number of JVM functionalities use these functions, and adding mere counters inside them were returning extra null / non null checks in even simple programs. This prompted us to log the function context inside the ifnull / ifnonnull statements and utilise that for our experiments.

3 Evaluation and Results

- The evaluation was done by the measures outlined in the methodology section.
- We evaluated these with the commands `openj9-javac File.java` followed by `openj9-java -Xint File` where openj9-javac and openj9-java are aliases to the built versions of javac and java from our modified variant of the openj9 VM. We have bundled all steps into the script `run.sh`.
- The primary test case we evaluated on is `Test4.java`. We created this by going through Soot's Nullness Analysis and identifying cases where we can definitely know if the object in condition is null/non-null.
- The results are as follows: The original bytecode consisted of 9 ifnull / ifnonnull checks, whereas bytecode from our transformed class consisted of only 1 such check.
- We also tried a timing-benchmark with `System.nanotime` but it was not always better for `Test4.java` with `main` looping 10^6 times - this was probably due to efficient CPU-branch-prediction.
- To make this harder, we randomized test function calls in `Test5.java`. Over 10^6 calls and averaged over 5 runs, there was an improvement of 4.47% from 40.237ms to 38.437ms. Interestingly, in this case, we got much greater gains without interpreter mode: 27.92%, arising mostly from 2 runs out of the 5 we averaged over - this is probably due to the profile-based manner in which the runtime works.
- On a more involved `Test6.java`, we saw an improvement from 11600 to 5890 nullchecks.

All files mentioned here are inside the `pa4` folder in the soot-analysis repository mentioned in section 1. Instructions to run the bash script are present in the README of the `pa4` folder.

4 Conclusion

We performed a transformation of class files to remove redundant null checks. Nullness Analysis from Soot was used to identify variables of known nullness. Further, we modified the OpenJ9 VM to evaluate effectiveness of our transformation. This was done by additional logging at certain locations within the bytecode interpreter. Finally, we tested the same by comparing number of nullchecks in original and transformed bytecode, along with a timing benchmark.