OJXPerf: Pinpointing Object-level Redundancies in Java Programs

Anonymous Author(s)

ABSTRACT

This document provides the supplementary materials for the PLDI 2020 submission. It describes the optimizations under the guidance of OJXPerf for JGFSerialBench [2].

1 CASE STUDIES

1.1 JGFSerialBench

JGFSerialBench is a benchmark of Java Grande benchmark suite, which are designed to be representative of scientific and other numerically intensive computation [2]. We run JGFSerialBench using its default input size. After profiling JGFSerialBench, one object, `arraybase`, which is allocated at line 371 in method `JGFrun` of class `JGFSerialBench` stood out, as shown in Listing 1. Listing 1 shows the allocation site of this `arraybase` is in a while loop. In each loop, the program creates a new `arraybase` and puts some contents info this `arraybase`. As this `arraybase` is a two-dimensional array, `arraybase` uses writeObject method (line 375) to put its one-dimensional elements (can be seen as an object) into an object `arrayout`. OJXPerf reports that the redundancy factor θ of the created `arraybase` is 68.8%, and its lower bound factor ω indicates that the largest identical `arraybase` group size is larger than 15.2%. To avoid such redundant object copy operations, we redesign the code as we first check whether the program has a different `arraybase` compared with the one from the previous iteration. If the `arraybase` keeps the same, then copy of `arraybase`'s contents is unnecessary. This optimization yields a $(1.1 \pm 0.03)\times$ speedup to the entire program.

```
368 public void JGFrun(){
369   ...
370   while  (time < TARGETTIME && size < MAXSIZE){
371     ▶arraybase = new arrayitem [size][LENGTH];
372     arrayout = new ObjectOutputStream(arrayfout);
373     ...  //put contents in array base
374     for (j=0; j<size; j++) {
375       ▶arrayout.writeObject( arraybase[j]);
376     }
377   }
378   ...
379 }
```

Listing 1: OJXPerf pinpoints the `arraybase` array suffering from object-level redundancy in JGFSerialBench

REFERENCES

[1] RoaringBitmap authors. 2019. RoaringBitmap. https://github.com/RoaringBitmap/RoaringBitmap.

[2] Mark Bull, Lorna Smith, Martin Westhead, David Henty, and Robert Davey. 2001. Java Grande benchmark suite. https://www.epcc.ed.ac.uk/research/computing/performance-characterisation-and-benchmarking/java-grande-benchmark-suite.