COMP207P
Compiler Coursework2 Report

Group members: XIAOFENG FU, HAO CHEN, WENTAO WEI

Introduction

This coursework gives us a chance to be familiar with the bytecode and we must figure out three tasks: simple folding, constant variable folding and dynamic variable folding. To solving these problems, we need to write an optimisation algorithm. And the recognition & replacement of the bytecode are the most important part in our algorithm. In this report, we will show our algorithm structure and explaining it in detail.

Algorithm Structure

The main method of our algorithm is *optimizedMethod().* At the beginning, we get the instruction list from the Method instance and then we put the instruction list in a MethodGen instance. Then *optimizedMethod()* first traverses the instruction list for one time and replacing all IINC bytecode by:

- BIPUSH
- ILOAD
- IADD
- ISTORE

and then this method traverses the instruction list again and it will operate bytecodes differently depends on different types of bytecodes: *ArithemeticInstruction*, *IfInstruction*, *ConversionInstruction* and *StoreInstruction.*

Arithmetic Instruction

In the case of *ArithmeticInstruction*, we will first get related values on the stack by *getValue* method. For example, if IADD is found, our algorithm will find two values that are needed to be added. Then we will insert the calculated value to the instruction list and delete the IADD instruction and related value instructions.

- LDC
- LDC2_W

If the value is an integer or a float, we will insert it by LDC, else if the value is a double or a long, we will insert it by LDC2_W.

If Instruction

For the *IfInstruction*, our algorithm includes:

- IFNE
- IFLE
- IFGE
- IF_ICMPLE

- IF_ICMPGE
- IF_ICMPNE

And we will get related values that are needed to be compared by *getPrev().getValue()* method. Then we will delete all related instructions expect the needed boolean value. Moreover, if the if instruction is a start of a for loop, our algorithm will skip the instruction.

Conversion Instruction

We will convert a number to specific type: integer, float, double and long if a *ConversionInstruction* is detected. It includes:

- I2F
- I2D
- I2L
- F2I
- F2D
- F2L
- D2I
- D2F
- D2L
- L2I
- L2D
- L2F

We get the value by *getPrev().getValue()* method and then cast this value to specific type and then insert it to the instruction list by LDC or LDC2_W.

Store Instruction

For the store instruction, we includes:

- ISTORE
- DSTORE
- FSTORE
- LSTORE

Once we detect a store instruction, we will continue traverse the instruction list and find all related load instructions with same index and replacing them by the value that stored in the store instruction by:

- BIPUSH
- SIPUSH
- LDC
- LDC2_W

And then we will delete the store instruction and the store.prev() instruction and also the related load instruction. Yet, if the method detect second store instruction with the same index, it will stop the load instruction replacement. Also, our algorithm will detect whether the

store instruction is a part of a for loop. Therefore, this algorithm will not change the for loop structure if the store instruction is a part of the loop.

Check

Since some instructions may be deleted during the second traversal. At the end of the algorithm, it will traverse the instruction list again and it will change the target of the if instruction of for loops to the correct target by *setTarget()* method.

Method list:

otpimizedMethod(ClassGen cgen, ConstantPoolGen cpgen, Method m);
//main structure
getValue(InstructionList list, InstructionHandle handle, ConstantPoolGen cpgen);
//get value of a instruction from the stack
convert(InstructionHandle handle, Number toChange);
//cast the type of numbers.
deleteInstruction(InstructionList list, InstructionHandle handle);
//delete a instruction