CS 4240 Project 1: Code Optimization

Chengming Gu, Paul Gibert

February 17, 2020

1 Architecture

This project has the following components: IR Instruction class, parser methods, IR generation methods, control flow graph class, optimizer methods, and a main script for running. Program execution starts by parsing all the instructions, passing it to the optimizer, and then outputting the final IR.

1.1 Intermediate Representation

We have decided to encode every possible line in tiger IR code as an IRInstruction object, this includes all the obvious suspects, but also:

- Function boilerplate (start and end signature, declaration, variable lists)
- Labels
- Function calls

The only "line" that isn't considered to be an IR Instruction is an empty line. Each instruction encodes information that includes but is not limited to:

- type/opcode
- a list of arguments
- some properties about the instruction, such as:
  - whether it is a def/use
  - the variables it writes/reads
  - whether it is a branch

1.2 Control Flow Graph

For a given program, there are exactly as many Control Flow Graphs generated as there are functions. This choice is because optimization does not occur across functions, thus there is a separate Control Flow Graph for each function. A Control Flow Graph encapsulates the following information/behavior:

- A list of instructions
- An adjacency list
- Helper methods for tasks such as:
  - Getting the kill set of a def
  - Getting the predecessor/successors of a def
  - Removing an instruction from the CFG

Each basic block in our codebase is exactly one IR Instruction, and this includes the forementioned types that are not normally consider to be instructions. These atypical instructions have empty gen and kill sets and control is passed from the previous instruction to the next instruction in order. Originally, the design attempted to use a maximal basic block, but this presented challenges when computing reaching definitions during for dead code elimination. A simpler single-instruction basic block reduced complexity and was opted for instead. This required no unique class for basic blocks.

1.3 Optimize

The optimizer contains functions for running dead code elimination using the fixed point algorithm as well as copy propagation. However, as noted in a later section, copy propagation is incomplete. Undiscovered bugs cause the algorithm to fail in finding replacement candidates which is likely due to its constraints being too strong.

Regardless, the optimizer optimizes the code (assuming both dead code and copy propagation are to be performed) by performing a first pass for dead code, a pass of copy propagation, and another pass for dead code. Note that in our implementation, after a single round of optimization, we actually output an IR file, and start the next round by parsing that IR file. The reason for doing this is twofold:

- This helps debug as we can see what each phase is doing individually, to see where any bugs might come from
- This also makes the line number correction easier (or not necessary at all) in our code, since there are some places which use line numbers and other places that use the index of an instruction in the CFG. This is a potential area for refactoring.

2 Implementation Language

Our final language of choice is python 3.8. We initially decided to use C++ as an opportunity to learn the language, but eventually decided to switch to python since there were some compiler discrepancies between members of the team (in particular how clang and g++ optimize loops differently).

In any event, there are some unique advantages that python provides that makes it easy to do this project:

- Builtin functions that trim whitespace away from strings
- Builtin set data structures that permit common set operations
- OOP capabilities
- Array slices

3 Challenges

As described before, we ran into a lot of issues with c++. This is partially due to lack of familiarity with the language, and also because of the more verbose nature of c++. While it can still do the same things that python does, having builtin functions and list comprehension and array slices make things much easier.

Other than the language difficulties, another thing that made it difficult at first was the fact that we used maximial basic block. After switching to a single-instruction block, the algorithm was much easier to implement and debug.

4 Known bugs

As mentioned earlier, in its current state, copy propagation does not work.

5 Potential Improvements

- Fix copy propagation
- Refactor the code so that there is consistent usage of line number vs index in the CFG across the entire codebase

6 Usage instructions

6.1 Version requirements

Any installalation of python 3 should work. We implemented and tested our code using python 3.8, but since we didn't use any 3.8 features, 3.7 should work as well. There shouldn't be any package that needs be installed, but if python reports the absence of any package, install with pip.

6.2 Running the optimizer

To run the optimizer, simply using the *main.py* file. When ran with the -h flag or without providing necessary flags, a help menu will show up that details the command line flags that need to be passed in.

As an example, one might run the code as follow:

```
python main.py --input="public_test_cases/sqrt/sqrt.ir" --output=
"public_test_cases/output/sqrt.ir" --dead
```

7 Test Results

We will only include results of dead code here, since copy does not work.

Test Case	Original	A	B	C	Ours
0	482	473	434	407	434
1	1124	1105	1026	970	1026
2	1806	1777	1658	1554	1658
3	2520	2481	2322	2172	2322
4	3126	3077	2878	2677	2878
5	4180	4121	3882	3638	3882
6	4654	4585	4306	3958	4306
7	5544	5465	5146	4746	5146
8	6274	6185	5826	5449	5826
9	7012	6913	6514	6018	6514

Test Case	Original	A	B	C	Ours
0	94	87	83	76	83
1	70	65	61	55	61
2	16	15	11	10	11
3	101	93	89	82	89
4	133	123	119	109	119
5	155	143	139	127	139
6	168	155	151	138	151
7	272	251	247	226	247
8	155	143	139	127	139
9	16	15	11	10	11

As can be seen, our output matches the compiler B result exactly. The criteria that we've constrained in code for copy propagation seems to be too tight, so that in most cases it doesn't actually replace any copies.