CS 380C: Advanced Topics in Compilers
Assignment 4: Loop analysis

March 4, 2023

Late submission policy: Submission can be at the most 2 days late. There will be a 10% penalty for each day after the due date (cumulative).

The goal of this assignment is to familiarize you with the LLVM compiler by implementing a simple LLVM pass to analyze loops in programs.

1 Properties of loops

A compiler analyzes and optimizes programs in passes; each pass scans or traverses the intermediate code to analyze or transform it. Analysis passes are typically used to collect information about a program that helps in ensuring that an optimization pass is safe (not alter the meaning of the program) or determining whether an optimization pass could be beneficial. In this assignment, you will implement an analysis pass to learn interesting properties about the loops in a program. Your pass should report the following information about all loops that are used in a program:

1. Function: name of the function containing this loop.
2. Loop depth: 0 if it is not nested in any loop; otherwise, it is 1 more than that of the loop in which it is nested in.
3. Contains nested loops: determine whether it has any loop nested within it.
4. Number of top-level basic blocks: count all basic blocks in it but not in any of its nested loops.
5. Number of instructions: count all instructions in it, including those in its nested loops.
6. Number of atomic operations: count atomic instructions in it, including those in its nested loops.
7. Number of top-level branch instructions: count branch instructions in it but not in any of its nested loops.

For each loop, print a line to standard error in the following format (case and space sensitive):

```
<ID>:  func=<str>, depth=<no>, subLoops=<str>, BBs=<no>, instrs=<no>, atomics=<no>, branches=<no>
```

`<ID>` represents a global counter for each loop encountered, starting at 0. `<str>` represents a string for function name or boolean value - “true” or “false”. `<no>` represents a number.

2 Getting Started with LLVM

Familiarize yourself with the LLVM compiler infrastructure since you will be using it for the coming assignments too. You can find all the necessary documentation here:

http://llvm.org/docs/

You can use the following LLVM manuals to learn more about the compiler:

Downloading and building LLVM	http://llvm.org/docs/GettingStarted.html
LLVM command line tools	http://llvm.org/docs/CommandGuide/
LLVM IR Reference Manual	http://llvm.org/docs/LangRef.html
LLVM Programmer's Manual	http://llvm.org/docs/ProgrammersManual.html
Writing an LLVM Pass	http://llvm.org/docs/WritingAnLLVMPass.html
LLVM testing infrastructure	http://llvm.org/docs/TestingGuide.html

Read these manuals very selectively, or you will spend far too much time on them!

Download and build the source code for release version 11.0.0 (do not get the source code from the SVN head); configure LLVM for a release build.

You can use any machine for development. You will need roughly 30 GB of disk space on your machine. The build may take around half an hour on a laptop.

3 Implementation Guidelines

Please follow these guidelines precisely because the grading scripts will be based on it.

1. Create a new directory `*UTEID*-LoopAnalysis` in the `lib/Transforms/` folder of the LLVM source tree - `$SRC_ROOT` where *UTEID* should be your EID in lower case letters (for example, abc889-LoopAnalysis). All your source files, including your build script (`CMakeLists.txt`), should be within this directory. This is the source directory you will submit.
2. Register your pass as `*UTEID*-loop-props` (in your source code).
3. Create a `cmake` build script to build ~~`LoopAnalysis.so`~~ in the `lib/` folder of the LLVM object tree - `$OBJ_ROOT`. You can use `$SRC_ROOT/lib/Transforms/Hello/CMakeLists.txt` as a template.
4. Append this line to `$SRC_ROOT/lib/Transforms/CMakeLists.txt`:

   ~~`add_subdirectory(LoopAnalysis)`~~
5. Run your pass using this command:

   ```
   opt -load $OBJ_ROOT/lib/LoopAnalysis.so -loop-props <$INPUT >/dev/null 2>$OUTPUT
   ```

   $INPUT will be a LLVM bitcode file (`.bc`).

IMPORTANT: DO NOT READ `$SRC_ROOT/lib/Transforms/Scalar/LICM.cpp`

You are not allowed to copy source code from anywhere, including other LLVM source files. You can include the header files and call those functions if you think it does precisely what you want. If you are not sure whether you can use something or if you think some LLVM source code is making the assignment trivial, then please let us know on Piazza (you can use a private note).
Note: This assignment is intended to be straight-forward.

4 Testing

You are responsible for writing test cases to test your pass. Write test cases such that you can test various aspects of your pass; start off with simple ones and then add more complex ones. Feel free to use LLVM test programs. We can use any program for testing your pass. Your program must be robust (no crashes or undefined behavior) and precise (correct output). So, please test your code thoroughly.

5 Deliverables

Submit (to canvas) an archive (preferably, `.tar.gz`) of the source directory containing your source code, build script (`CMakeLists.txt`) and `README` file. Please do not submit the built binary.
The `README` file should mention all the materials that you have read or used for this assignment, including LLVM documentation and source files; you can only skip mentioning header files that you have explicitly included in your source code. Mention the status of your submission, in case some part of it is incomplete. You can also include feedback, like what was challenging and what was trivial. Please also mention your project partner.
You can also submit the test cases for which your pass worked (in the same directory); if they are publicly available, then just mention them in the README file.
Please feel free to help each other with the build system since that is not focus of this class.

6 Grading

We will do the following to grade your assignment:

1. Unzip the archive you submit into the `$SRC_ROOT/lib/Transforms` folder, which will contain a `CMakeLists.txt` that adds your directory as a sub-directory.
2. Run `make` inside your sub-directory in the `$OBJ_ROOT/lib/Transforms` folder.
3. Use `opt` as mentioned in the implementation guidelines to test your pass on a few input programs. Your program should not crash.
4. Compare (`diff`) the output (to standard error) against the expected output. It should match.

Note that this will be done through scripts. So, please follow the output and implementation guidelines precisely.

Any clarifications and corrections to this assignment will be posted on Piazza.