Assignment 1: Introduction to LLVM

EECE 571P — Program Analysis and Optimizations
Winter Term 2015

Due: Feb. 9, 11:59 PM

Goal

For this assignment, you will implement a Branch Instruction Count pass in LLVM. The goal of this pass is to collect the count of conditional branches and unconditional branches in a program and report it.

We will be using the Low Level Virtual Machine (LLVM) compiler infrastructure developed at the University of Illinois Urbana-Champaign for our project work. While you are free to choose the whatever version of the platform/OS that LLVM supports, we will support X86 based machine and Linux OS. Therefore, if you use a different platform/OS, please pay attention to the compatibility. We recommend the LLVM version to be 3.7.1.

Get Started with LLVM

There are 3 big steps to get started with LLVM.

1. Download the LLVM from `http://llvm.org`. Follow the instructions at `http://llvm.org/docs/GettingStarted.html` to install it. *We recommend the build type to be "debug builds with assertions enabled" (see the instructions for details) so that you are able to debug your code in the future.* We recommend you use Clang as the compiler front end. The installation guide of Clang can be found at `http://llvm.org/docs/GettingStarted.html` and `http://clang.llvm.org/get_started.html`. You will install(build from source) LLVM 3.7.1 if you follow the instructions at `http://llvm.org/docs/GettingStarted.html` or TA's tutorial on Piazza.
2. Read through the documentation at `http://llvm.org/docs/` (Design& Overview and User Guides in particular). Writing an LLVM Pass tutorial (`http://llvm.org/docs/WritingAnLLVMPass.html`) is particularly useful.
3. Write some toy examples in C or C++ and play with it. Compile the examples to LLVM IR files with Clang and try some LLVM analysis and transform passes on them.

Branch Instruction Counting

In this assignment, you will write an analysis pass which collects the number of conditional branches and unconditional branches in a program. For conditional branches, you will also collect the number of "equal", "greater than" and "less than" comparisons. The numbers will be reported with 5 metrics:

- *NumCondBranch* = Number of conditional branches in the program
- *NumUncondBranch* = Number of unconditional branches in the program
- *NumEqBranch* = Number of conditional branches whose comparison type is equal test
- *NumGTBranch* = Number of conditional branches whose comparison type is greater than test
- *NumLTBranch* = Number of conditional branches whose comparison type is less than test

These are already declared in the skeleton code, using the LLVM STATISTIC mechanism. What you need to do is to update the variables at the right time.

Implementation Guidelines

1. Download the skeleton code on Piazza and extract it somewhere. Copy the folder BranchInstCount to `$LLVMSRC/lib/Analysis`[1] . Add the folder to CMake list.
2. Add code to the BranchInstCount.cpp file; this is the only file you will be handing in.
3. Go to `$LLVMDST` and run `make` to create a dynamically loadable shared library in `$LLVMDST/lib/`. On Linux, it will be called libAssign1.so. You can try it out with the following command:

   ```
   opt -load $LLVMDST/lib/libAssign1.so -help
   ```

   You should see your pass (`branchinstcount-assn1`) listed along with other optimization passes.
4. Test your pass.

Test Your Pass

You are responsible to test your pass. You can add the argument `-stats` when you run your pass to check the correctness of the metrics statistics. You can start with some small examples and understand the collected numbers.

There is a `test` directory in the handout. You can make changes to the `Makefile` under `test` directory to automate your test procedure.

Handin

When your code is done, just submit the BranchInstCount.cpp as an attachment and email to the TA with "EECE571 Assignment1" in the subject: gpli@ece.ubc.ca.

Evaluation Criteria

The assignment counts 15 points. Coding style counts for 5 points, the left points are evenly distributed among the five metrics, i.e. 2 points each.

[1]$LLVMSRC means the root directory of the LLVM sources.