EECS 583 – Homework 1

Winter 2023
Assigned: Wed, January 11, 2023
Due: Mon, January 23, 2023 (11:59pm Eastern USA)

Late Submission Policy

Submissions are accepted a maximum of two days after the specified deadline. For each day late, a 10% penalty will be deducted from your score.

Statistics Computation Pass

The goal of this homework is to learn to write your first real LLVM pass. As part of this, you will learn to use the profiler which provides dynamic execution frequencies for the compiler to make use of.

Write a statistics computation pass in LLVM that computes several dynamic operation counts for each function. First, the total number of dynamic operations should be computed along with the percentages in the following categories: integer ALU, floating-point ALU, memory, biased-branch, unbiased branch, and all other operations. Use the following rules when categorizing the operations:

- Branch: br, switch, indirectbr
- Integer ALU: add, sub, mul, udiv, sdiv, urem, shl, lshr, ashr, and, or, xor, icmp, srem
- Floating-point ALU: fadd, fsub, fmul, fdiv, frem, fcmp
- Memory: alloca, load, store, getelementptr, fence, cmpxchg, atomicrmw
- Others: others

Every operation should be placed in one of the categories, so all are counted. Print this information out in a text file named benchmark.opcstats in the following format (comma separated, one function in each line): FuncName, DynOpCount, %IALU, %FALU, %MEM, %Biased-Branch, %Unbiased-Branch, %Others
Percentage values (including %Biased-Branch, %Unbiased-Branch) are computed as the targeted instruction type over the entire DynOpCount (e.g. %Biased-Branch = biased_branch_inst_count / total_dyn_inst_count * 100%).

For example, if a function has 50% operations being integer ALU operations, its %IALU should be 0.500. To print the value "val" in LLVM in this format, use the following: #include "llvm/Support/Format.h" then errs() << format("%.3f", val); for more info, see [link]. Print all zeros if a function has never been executed.

The branch bias is: (frequency_of_most_likely_target / total_execution_frequency). For branches with only a single target, the branch bias is 100%. For classification, branches are considered biased if the bias > 80% and unbiased otherwise.

To write a pass in LLVM, refer to https://llvm.org/docs/WritingAnLLVMPass.html. You should always keep your code separate from the core compiler code to ease integrating future releases. To get dynamic statistics, you will need to learn how to run the LLVM profiler.

To get started, download 583_W23_hw1.tgz from the course website. This file contains 3 benchmark applications (explained below) as well as skeleton code for your pass and a run script for running LLVM. To unpack the tgz file, run tar zxvf 583_W23_hw1.tgz

Benchmarks to Run

There are 3 benchmark applications that you should profile and collect statistics for: 583simple, 583wc, and 583compress. Each is progressively larger and more complex. Each benchmark contains directories for source code (src with 1 .c file), sample input file (input1), sample output file (output1), and a text file with information on how to run the benchmark, exec_info_input1. It is recommended to use the run shell script (run.sh) that we have provided to execute your pass on the benchmarks. Instructions for using the run script are given in comments at the beginning of the run script.

If you prefer to run things manually and not use the run script, here is some additional information about running the benchmarks. Execution parameters are given in exec_info_input1, you can use those to setup, run, and check the output of the file. Since you are executing an analysis pass, the output of the code should not change. If running it manually remember to copy the input file to the same directory as your binary executable to ensure that your profile counts match ours. You can compile each benchmark with gcc to make sure they work before running LLVM. Some benchmarks may check if an output with the same name already exists (for 583compress, compress.in.Z) and ask users whether to overwrite the file if it already exists. This will make a difference in program execution, thus leading to different statistics. Therefore, please make sure to delete output from the previous run before collecting your statistics.

Submission

To submit your homework, put a single .tgz (gzipped tar file) into the directory /hw1_submissions/ on eecs583a.eecs.umich.edu via scp (later versions will automatically overwrite the earlier versions if you submit multiple times):

```
$ tar cvzf ${uniquename}_hw1.tgz ${uniquename}_hw1
$ scp ${uniquename}_hw1.tgz ${uniquename}@eecs583a.eecs.umich.edu:/hw1_
submissions/
```

Please name your tar file ${uniquename}_hw1.tgz, and organize the directory using the following structure:

```
${uniquename}_hw1/
        src/
                hw1_pass.cpp (source code for your pass)
        results/
                583simple.opcstats
                583wc.opcstats
                583compress.opcstats
```