Advanced Homework 10

Assigned: Friday, March 18th

Due: Before the first lecture on Friday, April 1

Submission Instructions

To receive credit for this assignment you will need to stop by someone's office hours, demo your running code, and answer all the bolded questions.

From `gprof` to `gcov`

gcov is a more advanced profiler. While gprof gave only function-level information, gcov can give line-by-line statistics, letting you find hot loops or other fine-grained issues.

What extra flags do you have to give the compiler to get gcov working?

The `-pg` flags added calls to `_mprofil` to every function, roughly, what do these new flags do?

One interesting thing that gcov can show you is "taken/not taken" statistics for branches (how often was this if statement true).

Modify the code from the homework to include some branches. Include some that are always taken, some that are never taken, and some that are taken probabilistically with 25, 50, and 75% chance (`man 3 rand` will be helpful). The probability that each branch is taken should be read from a file or command-line argument, i.e. you must be able to change the probability that a branch will be taken *without* recompiling your code. Your branches should run many, many times.

Show the output of gcov's branch metrics and show that your probabilities are working as expected.

Show how changing the probabilities leads to different results.

Gcc supports something called *Profile Guided Optimization*. The idea here is that gcc can use the result of a profiling run to guide it compilation. For example, if a branch is likely to be taken, it can provide hints to the CPU that that branch will be taken. In some cases, it will even emit code that removes the branch, assumes it was taken, and checks at the end whether it was right, undo-ing the code that was run if it was wrong.

Show how to add coverage information to compilation and optimization.

Can you find a program that runs faster with coverage information? Any previous projects that do searches are good candidates. Anything long-running will be better. If you can't find something that runs noticeably faster, can you at least show an example where the compiled output (`objdump`) changes? Can you give a *rough* explanation of what changed (you *do not* need to deeply understand x86 assembly! An educated guess is fine). objdump gives a lot of information, you only need to look at specific functions though – use search.