1 Introduction

In this homework, we prepare some of the tools we need for this and subsequent homeworks, and get some initial practice using profilers to identify performance bugs in a program.

For this class, you will need a computer under your control, capable of running Ubuntu 18.04 either natively, or in a virtual machine. You will be provided with free VMWare licenses, but if you have a preferred hypervisor, feel free to use that instead. If you do not have a laptop or desktop computer capable of running these softwares, you can request a loaner laptop for the semester from the Computer Science department. Contact Phil Beltran ⟨pbeltr1@uic.edu⟩ for more information about this option.

2 Prerequisites

- Optionally create a virtual machine with sufficient storage for Ubuntu 18.04 and some additional programs. 5 GB should be ample.
- Install Ubuntu 18.04 Desktop[1].
- Install the build-essentials package `apt-get install build-essentials`
- Install linux-tools package, for the `perf` utility.
- Install bcc and all dependencies, the INSTALL.md file here has instructions for installing using apt: `https://github.com/iovisor/bcc`: the snap package `apt-get install snapd; snap install bcc` fails to resolve symbols for some reason.
- Install the valgrind package.

3 Performance Debugging

The program `raytrace`, available at `http://github.com/uicperformance/sillybugs`, contains a number of severe performance bugs. First install the m4 macro preprocessor `apt-get install m4`, then clone the repository, edit `codes/Makefile.config` to have the correct BASEDIR, then go to the folder codes/apps/raytrace and build the binary using `make`.

To run the program use this command line `./RAYTRACE -a1 inputs/teapot.env` in the raytrace folder. It takes a few seconds to run, due to the bugs.

In addition to the source code, the git repository contains the binary `RAYTRACE_original`. This is the original program, without the introduced performance bugs, compiled with the same Makefile. You will find that the original binary runs much

[1] Ubuntu Desktop is a great choice, with lots of good free software for it. However if you, like me, prefer to work on a Mac desktop, you can install Ubuntu 18.04 Server instead, and ssh to the VM from your mac. I would strongly advise against using a Windows desktop for this class.

faster, and your job is to figure out why. TIP: you can use `-a10` instead of `-a1` to make the programs take longer to run, potentially giving you more detailed profiling data.

The introduced bugs are well hidden, and the program is relatively large, making it extremely difficult to debug its poor performance without tools. You'll want to use the tools discussed in class, and any other profiling tools you like, to track down these bugs. Recommended tools include `time`, `perf record`, `valgrind --tool=callgrind` and `offcputime`. Since it is quite a bit of code, you may also want to use `grep` to search for keywords in text files.

3.1 The bugs

Several bugs have been introduced, and hidden with varying degree of sophistication inside the program. In this assignment, all bugs are exclusively added as new lines to the code, thus they can be fixed by simply deleting or commenting out the offending lines.

As a good first step, use `time` to note differences in time spent between the buggy and the bug-free binaries, then decide what tools to use to track down the bugs.

3.2 What to turn in

The task is to identify and fix all of the performance bugs in the program. You will know that you have succeeded when your compiled binary on average runs as fast as the original program, and produces the exact same output. Turn-in will be done via github classroom (instructions will follow). Please turn in your updated sillybugs source folder (sources only, not your executable or object files). To turn in your submission, first add and commit files you have added or modified in your repository. Then push the repository to your private GitHub classroom repository (after adding the turn-in repository as a new remote). The turn-in repo is accessible using the invitation link shared on Piazza.

Also, directly under the sillybugs folder, include a file named report.pdf, containing the following bug report for each bug you identified. You can use any program you want to produce the PDF file, but if you use LaTeX, and include the .tex sources next to the PDF file, you will receive a small bonus.

- A very short description of the bug, in terms of what unnecessary thing it does (e.g. *Uses the google search engine, via HTTP request, to calculate sums of integers.*).
- A very short description of the proposed fix (e.g. *replace HTTP request with + operator*).
- A brief description of how you found the bug.
- A screen shot of the tool you used, showing the evidence you used to track down this bug.

3.3 Validating your fixes

Before turning in your homework, verify that your performance fixes did not introduce correctness bugs. RAYTRACE produces an output file (inputs/teapot.rl). Use any tool you want to check that the output from RAYTRACE_original and your RAYTRACE is the same.

3.4 Double-check your submission

To make sure that you submitted everything you think you submitted (`git` can be a little tricky until you get used to it), *git clone* your turn-in repository into a fresh folder, check that your `report.pdf` is in the sillybugs folder, and that RAYTRACE compiles as you would expect.