COMP 40 Lab: Introduction to profiling tools

November 12, 2010

Pairs

If you have not yet submitted a working Universal Machine, you *must not* partner with someone who does have a working Universal Machine they can measure.

If you have a working Universal Machine that can run `midmark.um` in less than three minutes, *please start this command in a Terminal window* right away:

```
  valgrind --tool=callgrind --dump-instr=yes ./um midmark.um
```

If you do not have a working Universal Machine, please continue to the introduction below.

Introduction and instructions

If 80% of a program's time is spent in input and output, and only 20% of the program's time is spent in computation, you'll get a bigger performance improvement by speeding up the I/O, where the most time is spent. How do you know where the most time is spent? You could guess, but even programmers with many years of experience are notoriously bad guessers. This lab will introduce you to tools you can use so you don't have to guess.

The plan of the lab is

1. Get code from the `git` repository for the lab.
2. Use `valgrind` and `kcachegrind` to answer some simple questions about image rotation.
3. Compare profiles of two versions of `unblackedges`
4. Submit the answers to those questions by using `submit40-lab-profile`
5. If time permits, and you have a working UM, use `valgrind` and `kcachegrind` to answer some simple questions about the Universal Machine

Getting the code

If you run

```
git clone -o comp40 /comp/40/git/lab-profile
```

then you should get a file `QUESTIONS` as well as some source code and a `compile` script. Running `sh compile` should get you an `unblackedges` binary.

- If you look at the link command in the compile script, you will notice that `unblackedges` is linked against an *optimized* version of CII, using the linker argument `-lcii-O2`.

You should also be able to link against `-lcii-O1`.

Creating an execution profile using `valgrind`

As a sanity check, run

```
pngtopnm /comp/40/images/bitonal/hyphen.png | ./unblackedges | display
```

If all is well, you should see a page of text with black edges removed. Next, to know how long the program takes, run it with the `/usr/bin/time` command:

```
pngtopnm /comp/40/images/bitonal/hyphen.png | /usr/bin/time ./unblackedges > /dev/null
```

Finally, to get a profile, use `valgrind`:

```
pngtopnm /comp/40/images/bitonal/hyphen.png |
valgrind --tool=callgrind --dump-instr=yes ./unblackedges > /dev/null
```

The first option tells `valgrind` to create a *call-graph profile*. The second option tells `valgrind` to record the number of times each machine instruction is executed. The overhead of profiling is very high—expect this step to take a few minutes.

Viewing the profile with `kcachegrind`

Look in your directory for the profile data. It will be in a file called `callgrind.out.`*pid*, where *pid* (process ID) is a number that identifies the Unix process used to run your program. Supposing that your *pid* is 2268, run

```
kcachegrind callgrind.out.2268
```

and view the results. Here are some things to try with the user interface:

1. Click on `main` in the left panel. This enables you to view the whole program.
2. In the sequence of tabs near the top of the screen (in the middle), click on *Callee Map*. This tool helps you visualize a tree as a set of *nested rectangles*. The total area inside a rectangle is the time spent in a given function and all its callees, and so on transitively. So the full rectangle is the whole program, and `Bit_put` is a function that takes a significant amount of time by itself, without much time spent calling other functions.[1]

   In the upper right, you can also see that a lot of time is spent in `putc`.
3. In the sequence of tabs near the bottom of the screen (in the middle), click on *Call Graph*. This shows
   - Each procedure
   - The number of instructions executed in that procedure
   - Arrows to other procedures it called.
4. Click on the large % sign in the row of icons in the upper left, just below the menu. Displays switch from counting instruction fetches to percentages. Return the call graph and figure out what fraction of time is spent in I/O versus actually removing black edges. Add this to the `QUESTIONS` file for the lab.

Answering simple questions about image rotation

At this point you should be ready to profile and view image-rotation code. Please do so, answer the questions in the lab's `QUESTIONS` file, and run `submit40-lab-profile`.

Advanced exercise: improving `unblackedges`

1. Copy file `unblackedges-lab.c` to file `unblackedges-queue.c`.
2. On lines 23 and 24 of `unblackedges-queue.c`, please change `Seq_addlo` to `Seq_addhi`. This changes switches the graph search from depth-first search to breadth-first search.
3. Add a case to the linking code in the `compile` script so that you can link a new executable binary `unblackedges-bfs` to do black-edge remove via breadth-first search.
4. Time it and profile it.
5. What happens to the total time?
6. What happens to the fraction of time spent in `Bit_put`?
7. Since the implementation of `Bit_put` has not been improved, what has changed?

Improving Universal Machine

If you do not finish this exercise during lab, you do not have to finish it outside of lab. But if you have time, it will be helpful preparation for the profiling assignment. At this point you should be ready to profile and view your Universal Machine.

[1]Alert! You may see different behavior on the lab machines than I saw on the machine on which this lab was prepared.

Submitting

Please run `submit40-lab-profile` before leaving the lab, whether or not you have profiled the `Universal Machine`.