Tutorial 5: Profiling C++ Applications

Justin Nguyen 10/25/2019

This tutorial assumes basic knowledge of C++, Makefiles, and Linux.

For this tutorial we will be using `gprof` to profile our sobel program and `perf`. A profiler is an application that analyzes (or "profiles") an application while it is running. It can analyze the space, complexity, timings, and frequency of various aspects of the program. The `perf` application is similar to the Performan Monitor Application on Windows or `top` on Linux. This wrapper program will provide some simpler performance information about the program after it's run.

1. Install `perf` by running the following command (note these commands are for Ubuntu):

```
$ sudo apt update
$ sudo apt install linux-tools-common gawk binutils
```

2. Make sure your sobel program can take a video file and that when it completes the program closes successfully,

3. To use `perf` you simply run the `perf` program with your program as an argument like so:

```
$ perf                          # Shows all options
$ perf stat sobelExample        # Prints stats of sobelExample upon exit
```

4. We want to make sure the sobel program completes on its own so we can use one standard video file to compare all the measurements against the same input. Your output shoud look like the following:

```
 Performance counter stats for './sobel example.mp4':

         48,358.49 msec task-clock:u              #    2.716 CPUs utilized
                 0      context-switches:u        #    0.000 K/sec
                 0      cpu-migrations:u          #    0.000 K/sec
            33,675      page-faults:u             #    0.696 K/sec
   154,072,102,212      cycles:u                  #    3.186 GHz
   235,644,831,484      instructions:u            #    1.53  insn per cycle
     7,692,738,964      branches:u                #  159.077 M/sec
       117,990,406      branch-misses:u           #    1.53% of all branches

      17.804835857 seconds time elapsed

      47.369421000 seconds user
       0.873441000 seconds sys
```

Take some time to analyze the performance of your program and understand what each field means.

5. To use `gprof` to profile our applicaiton, add the `-pg` switch to your compiler flags in your Makefile.

6. Compile your program and run it with the example video file. When it finished you should see your program created a file called `gmon.out`.

7. You can now view the profiled statistics by running `$ gprof exampleProgram`. You should see that the output consists of two parts, the flat profile and the call graph. Read the output and understand what the statistics it returns means.

8. You may find that `gprof` may not correctly profile your application. This is because gprof does not profile multi-threaded programs well. You can try to install `valgrind` to profile your application instead:

```
$ sudo apt update
$ sudo apt-get install valgrind kcachegrind graphviz
```

9. You can use the following command to use `valgrind` to profile your application: `$ valgrind --tool=callgrind programName [program_options]`. Note that if you do not pass the `--tool=callgrind` switch, `valgrind` will only run the memory leak performance tool.

10. Run your program with `valgrind` and you will see that it creates a `callgrind.out.XXX` file. This file can be inspected with a text editor, but it is very cryptic. We will use KCacheGrind to view the results instead. You should be able to launch this application through your OS's launcher or on the command line. Read the results and understand the statistics of your profiled applicaiton.