ECE 785: Project 1 (v1.0)
Basic Performance Analysis and Display

Concepts

In this project you'll modify code to measure the SoC temperature and CPU frequency periodically, log them and display them in a live graph with matplotlib. Page numbers refer to the textbook (Molloy's **Exploring Raspberry Pi**).

Starting Point

Start with the temp.py program on the course GitHub repository at AES-2021/Project1. This is based on the Raspberry Pi Organization's Temperature Log example project. (The instructions refer to using the Mu programming environment, which is already installed and is accessible through the desktop interface.) The original RPi project code has some bugs, and the GitHub version doesn't have the graph plotting code.

- Information on matplotlib.pyplot is here.
- Many Python tutorials: https://wiki.python.org/moin/BeginnersGuide/Programmers

Monitor Temperature, CPU Frequency and Voltage

Modify the Python program so it also monitors/logs/displays CPU frequency and voltage.

- Restructure temp.py to use matplotlib.animation as described here. Reduce the interval argument to animation.FuncAnimation to determine the maximum update rate with no benchmarks running and include this value in your report.
- Measure the CPU frequency and voltage by running separate programs.
    - Use a subprocess to call a program from Python with **os.popen** or a similar mechanism, as described here.
    - You have two programs which can read the frequency:
        - **cpufreq-get**. See page 165 for how to install and use cpufrequtils.
        - **vcgencmd measure_clock arm.** Type vcgencmd commands to see the available commands, and RPi 4 additions are discussed here.
    - Read the voltage by running the program **vcgencmd measure_volts core.** It is not clear if the core voltage is the same as the CPU voltage. *Extra credit: confirm or disprove this with web searches. Include your findings (with citations) in your report.*
- Use matplotlib.pyplot to create four labeled subplots:
    - One line plot called **Temperature** for SoC temperature over time.
    - One line plot called **Arm Frequency** for the Arm CPU frequency over time. All four CPU cores are clocked at the same frequency, so only one frequency needs to be plotted. See this tutorial and related pages for details.
    - One scatter plot called **Temperature vs. Frequency** showing temperature (Y axis) vs. core frequency (X axis).
    - One scatter plot called **Voltage vs. Frequency** showing voltage (Y axis) vs. core frequency (X axis).

Consider closing web browsers or extra applications on the RPi as they may increase the compute load and the temperature. In addition, running the python program from the command line (rather than through Mu) will reduce the compute load. To do this, use **python3 temp.py** rather than **python temp.py** (which failed when it couldn't import matplotlib).

Run Times and CPU Frequencies

Chapter 5 (pp. 160-165) starts with a comparison of the performance of an N-body gravitational simulator program (5,000,000 iterations) when implemented in different programming languages. The code is in the directory chp05/performance of the exploringrpi GitHub repository. If you haven't already done so, clone that repository using the instructions on page 110 (*Code for This Book*). You will run the code in several ways and evaluate the program performance (run time), voltage, frequency and temperature.

1. Baseline configuration: on-demand governor
2. Performance governor
3. Powersave governor

Report Contents

Configuration

- Hardware configuration: RPi version, RAM quantity, SD card class, what cooling (if any) is used on the SoC (heatsink? fan?), whether using an RPi case, whether in headless or desktop mode

Temperature, CPU Frequency and Voltage

Monitor Program

- Source code listing of temperature/frequency/voltage monitor program
- What is the maximum update rate of your animation/plotting code?

Data: Four Screenshots of Python Plot Window (With all Four Subplots)

- Baseline (idle): 30 seconds of data with no browser or benchmark running.
- Tests: select specified governor (listed below), start recording temperature and core frequency, wait for at least 5 seconds (idle), then run **C/C++ benchmark**, then wait long enough (idle) for SoC temperature to fall to near idle level. You may automate this if you like.
    - On-Demand
    - Performance
    - Powersave

Analysis

- If CPU speed was throttled in any tests, explain when it started (e.g. 3 seconds after start). If not, state it was not throttled.

Benchmark Run-Time Performance

Instrumentation Overhead

- Compare the C/C++ benchmark with Performance governor with minimal interval and interval=1000. How much does the FuncAnimation interval affect the run-time performance of the benchmark programs?

Data

	cpufreq governor		
Language of Program	**On-demand (seconds)**	**Performance (seconds)**	**Powersave (seconds)**
C/C++			
C++11			
Haskell			
Java			
Python			

Analysis

- The CPU clock frequency ratio for performance vs. powersave is 1500 MHz/600 MHz = 2.5. For a purely CPU-bound benchmark, the runtime should fall by that factor. By what factor did each benchmark runtime fall? What does this imply?