Hardware Configuration

- RPi Version: 4

- RAM Quantity: 8 GB

- SD Card Class: SanDisk 32GB Class 10 MicroSD Card

- Cooling: None

- Case: Yes

- Desktop

Source Code

Located in Appendix A

Maximum Update Rate

I didn't implement blitting for this project. I understand that blitting can save time when updating the animation, but the overhead associated with splitting the entire figure update animation between different subplots wouldn't have been any more efficient.

The function would need to be split because of the scatter plots. Those plots were dependent on new values for their x-axes.

The maximum **dependable** update rate I was able to achieve was 430 milliseconds. I could run the plots temporarily at lower rates, but they would eventually hang.

A side note on this rate: I measured the amount of time that it took to update the plot, and I found that the average time taken was around 430 milliseconds. I also know that there were times when updating the plot took 100 milliseconds longer than that. There were times that the time taken to run the animation exceeded the interval between function calls. I had assumed that the reason the animation failed was because the interval was less than the time needed to update the animation, but that doesn't seem to be the case.

Images

Figure 1: Baseline

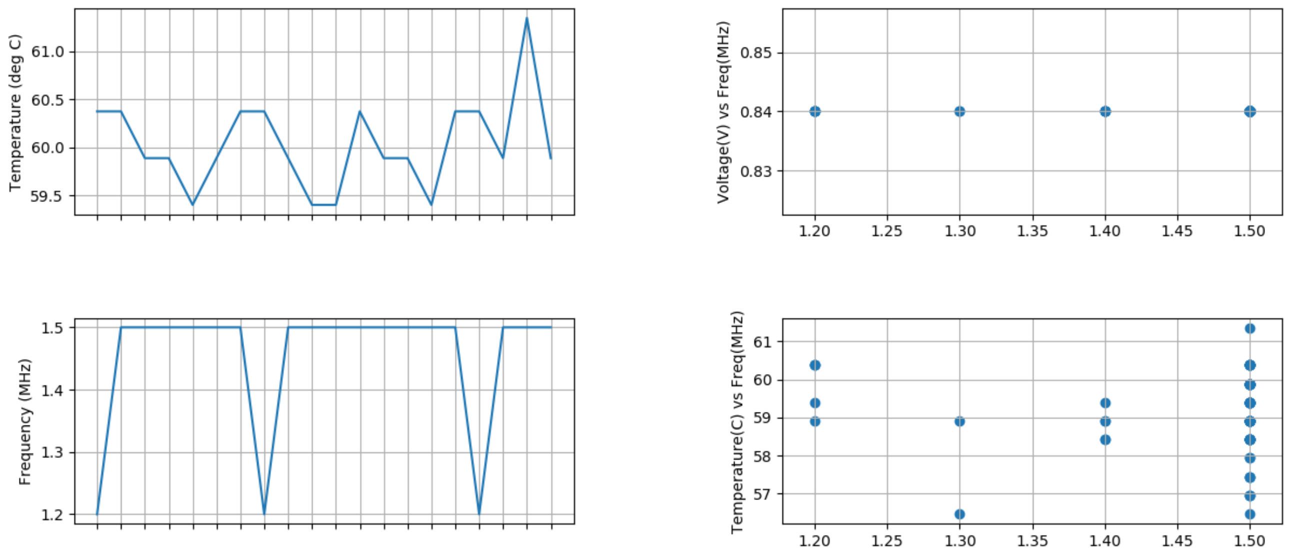


Figure 2: OnDemand

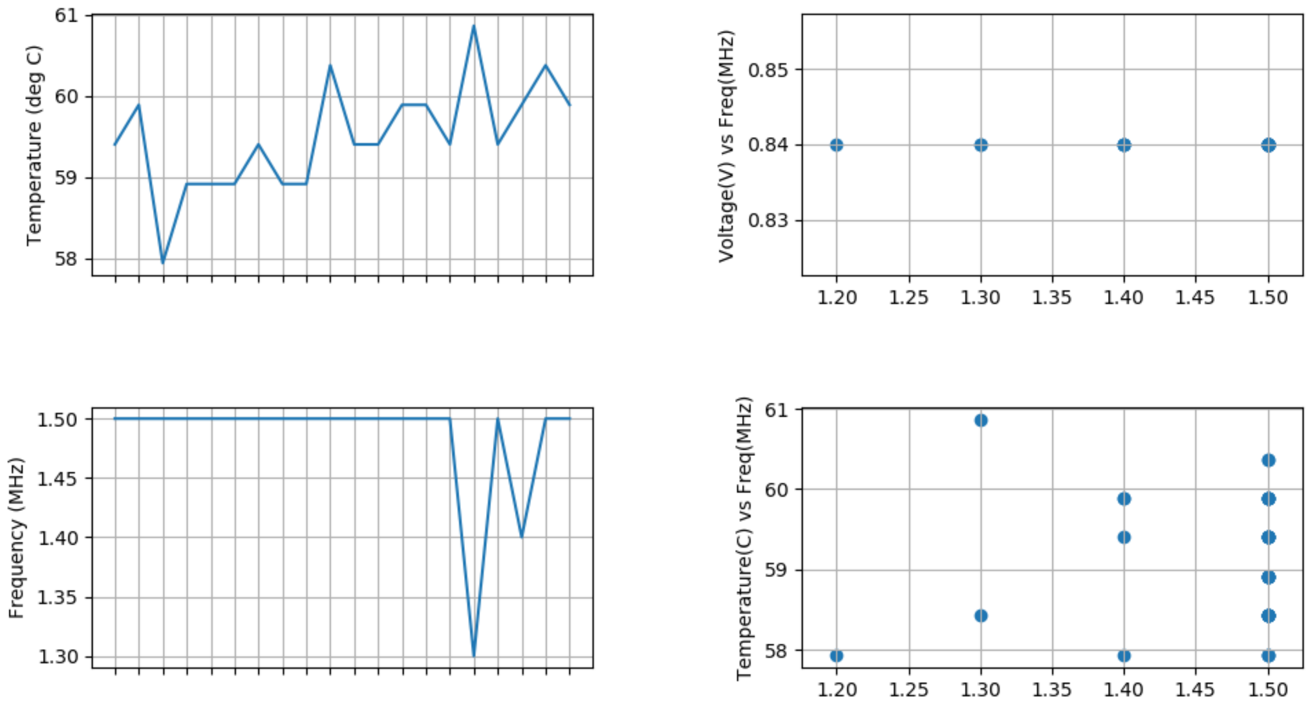


Figure 3: Performance

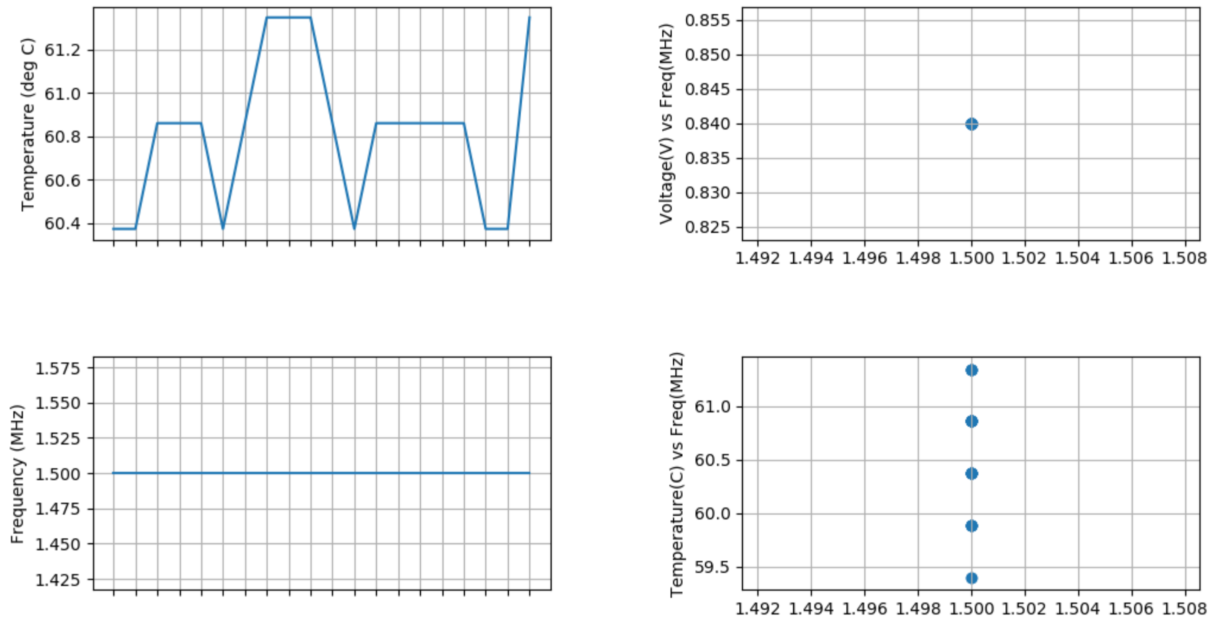
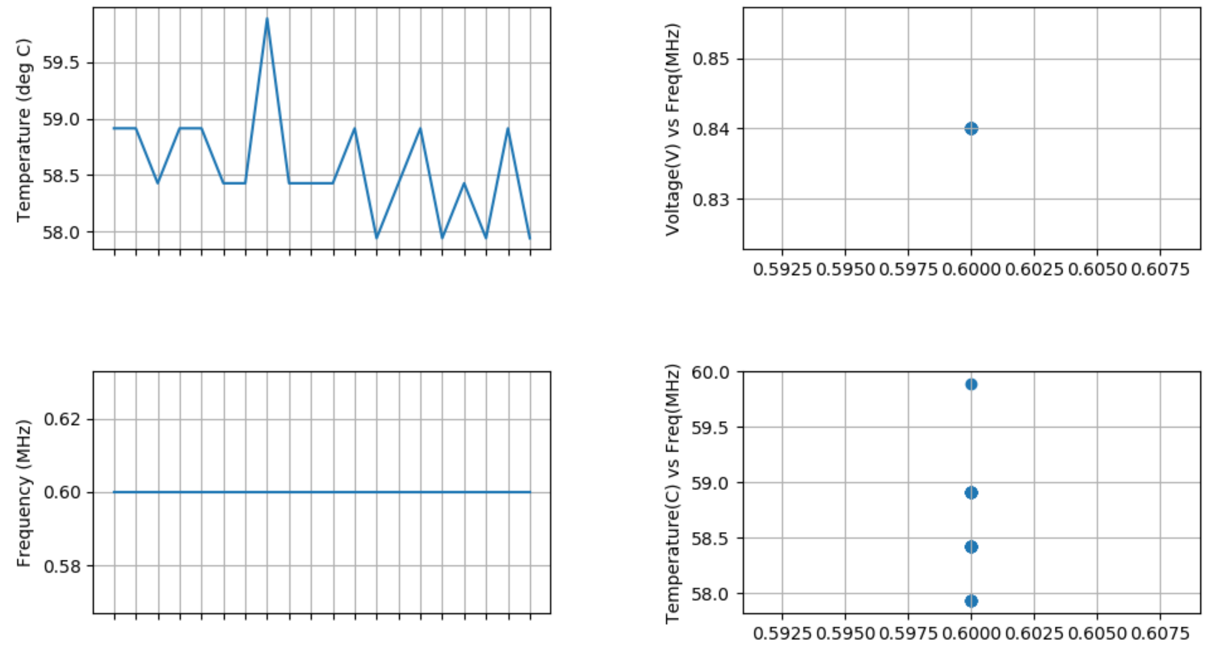


Figure 4: Powersave



Analysis

I think this question is asking if CPU frequency dropped during any of the benchmarking tests. I did not observe this during my tests. The frequency either remained constant or increased.

Instrumentation Overhead

Interval (ms)	Benchmark Time (ms)
430	1772
1000	1781

As the data indicates, there was no obvious correlation between the interval rate and benchmark result time. Intuitively, one would expect for the benchmark to take less time if the animation was happening less frequently. The animation must add some load to the CPU, so the benchmarking would take place with less resources. Apparently, this is not the case, as the benchmark time was higher when the animation was happening less often. Perhaps this is an anomalous occurrence, and the occurrence of the animation update and the benchmarking somehow had no overlap. Or, maybe the update is being handled by a graphics processing routine. It is difficult to pinpoint exactly what is happening on the pi at any given time.

Data

Language	On-demand	Performance	Powersave	Ratio (pws/perf)
C/C++	1.81	1.76	4.40	2.5
C++11	1.34	1.34	3.34	2.49
Haskell	2.65	2.65	6.62	2.49
Java	4.91	4.85	11.56	2.38
Python	158.68	162.03	406.01	2.51

The data above indicates that the run time ratio is very closely linked to the frequency ratio. The java bench mark is the farthest away in terms of the ratio deviating from 2.5. I am not sure why that is. I suppose if the benchmark was **purely** CPU bound, each of these values would be exactly 2.5. I believe the test was generated in an attempt to be CPU bound, so my guess is that any difference in the run time ratio is due to some external process running on the pi. Maybe the load on the CPU changed while the benchmark was running at one of the frequencies, or perhaps the load on the CPU changed between runs.

Appendix A

```
import os
import subprocess
import shlex
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from gpiozero import CPUTemperature
from time import sleep, strftime, time
import datetime as dt
plotInterval = 1000
fig, ax = plt.subplots(2, 2)
fig.tight_layout(w_pad=4, h_pad=4)
temperatureAx = ax[0, 0]  # fig.add_subplot(2,2,1)
freqAx = ax[1, 0]  # fig.add_subplot(2,2,3, sharex=temperatureAx)
frequencyCommand = "cpufreq-info -f"
voltageFreqAx = ax[0, 1]  # fig.add_subplot(2,2,2)
temperatureFreqAx = ax[1, 1]  # fig.add_subplot(2,2,2)

voltageCommand = "vcgencmd measure_volts core"
frequencyCommandArgs = shlex.split(frequencyCommand)
voltageCommandArgs = shlex.split(voltageCommand)
timeValues = []
temperatureValues = []
frequencyValues = []
voltageValues = []
animateRunTime = []
animateTemperatureRuntime = []


def getFrequency():
    data, temp = os.pipe()
    os.close(temp)
    s = subprocess.check_output(frequencyCommandArgs, stdin=data)
    freqValue = float(s.decode("utf-8"))
    return freqValue


def getVoltage():
    data, temp = os.pipe()
    os.close(temp)
    s = subprocess.check_output(voltageCommandArgs, stdin=data)
    commandOutput = s.decode("utf-8")
    commandOutput = commandOutput[5:-2]
    voltage = float(commandOutput)
    return voltage


def animateTemperaturePlot(i, timeAxis, freq, voltage, temp):
    global animateTemperatureRuntime
    animateTemperaturePlotRuntimeStart = time()
    currentTemp = CPUTemperature().temperature
    timeAxis.append(dt.datetime.now().strftime('%H:%M:%S'))
    temp.append(currentTemp)
    timeAxis = timeAxis[-20:]
    temp = temp[-20:]
    temperatureAx.clear()
    temperatureAx.plot(timeAxis, temp)
    temperatureAx.set_ylabel('Temperature (deg C)')
    # temperatureAx.get_xaxis().set_visible(False)
    plt.setp(temperatureAx.get_xticklabels(), visible=False)
    temperatureAx.grid(True)

    currentFreq = getFrequency()
    freq.append(currentFreq/1000000)
    freq = freq[-20:]
    freqAx.clear()
    freqAx.plot(timeAxis, freq)
    freqAx.set_ylabel('Frequency (MHz)')
    plt.setp(freqAx.get_xticklabels(), visible=False)
    freqAx.grid(True)

    currentVolt = getVoltage()
    voltage.append(currentVolt)
    voltage = voltage[-20:]
    voltageFreqAx.clear()
    voltageFreqAx.scatter(frequencyValues, voltageValues)
    voltageFreqAx.set_ylabel('Voltage(V) vs Freq(MHz)')
    voltageFreqAx.grid(True)

    temperatureFreqAx.clear()
    temperatureFreqAx.scatter(frequencyValues, temperatureValues)
    temperatureFreqAx.set_ylabel('Temperature(C) vs Freq(MHz)')
    temperatureFreqAx.grid(True)
    # animateTemperatureRuntime.append(time()-
animateTemperaturePlotRuntimeStart)
    # if len(animateTemperatureRuntime) == 10:
    #     sum = 0
    #     for entry in animateTemperatureRuntime:
    #        sum += entry
    #     avg = sum/10
    #     print("Average runtime for temperature plot:
{}".format(avg))
    #     animateTemperatureRuntime.clear()

aniTemp = animation.FuncAnimation(fig, animateTemperaturePlot,
fargs=(timeValues, frequencyValues, voltageValues,
temperatureValues), interval=plotInterval)

plt.show()
```