CPU Performance Before And After Multithreading

Mr. Sankar Dasiga[1], Sagar G V[2], Pavan Kumar B C[3], Prabodh B P[4], Ravi Kumar[5]
[1, 2, 3, 4, 5] Dept of ECE
[1, 2, 3, 4, 5] Nitte Meenakshi Institute of Technology

Abstract- *this paper is focused on developing a thread based application to stream video feed from multiple camera simultaneously. We are performing image processing on both frames and measuring the parameters such as resolution, FPS (frames per second) and CPU load, comparing these data in different conditions with and without using threading.*

This project work is performed on raspberry pi 3B development platform. Image processing is performed using opencv and python. The results are obtained by running script that can run in parallel with main script and it is stored in a file with timestamp attached with corresponding readings.
The overall CPU consumption remains the same and FPS is increased by 200% while using threading.

Keywords- FPS, Multithreading.

I. INTRODUCTION

In these Modern days, raspberry pi is being one of the sheets to chip away at. This board can be utilized as independent board which can go about as scaled down CPU. Numerous interfacing should be possible on this board effortlessly which utilizes an easy to understand programming dialect to be specific Python. As the board has 1.2 GHz processor and 1 GB RAM [1]. At the point when Image handling applications are utilized on this board, CPU tends to utilize the greater part of its RAM.

Keeping in mind the end goal to maintain a strategic distance from this working framework will give us a choice of multi-threading. Multi-threading is the capacity of a working framework with a specific end goal to execute different process or strings in parallel. It is meant to expand the usage of a solitary center by utilizing string level and guideline level parallelism. It likewise prompts quick general execution. Strings are lighter than process and offer same memory space. Multi-threading can be of three sorts in particular, Coarse-grained multi-threading, Interleaved multi-threading, Simultaneous multi-threading.

Coarse-grained multi-threading is the less complex multi-threading, which happens when one string will keep running until the point when its blocked and make a long-inactivity slow down. Multi-threading equipment bolster is utilized to permit brisk exchanging between a blocked and an unblocked string which is prepared to run.

Interleaved Multi-threading is to expel all information reliance slows down from the execution pipeline since strings are being executed simultaneously, shared assets should be bigger to abstain from whipping between various strings.

Synchronous multi-threading, this kind of multi-threading applies to CPU that executes a type of parallelism inside a solitary processor. This write is utilized to diminish the waste related with unused issue and spaces.

In Python, threading is an I/O bound assignment. The processor can switch between strings when any of the string is prepared to work. For CPU bound errands threading modules will keep running in a moderate execution time.

In our Project we are interfacing two camera modules, one is PiCam and WebCam. At the point when CPU bound errands are gotten to utilizing PiCam module the CPU nearly utilizes 70% of the RAM and we can't interface another camera until the point that multi-threading is finished. Subsequent to multi-threading the Camera modules are interfaced on a solitary processor and can work proficiently and the utilization of CPU is same as one preceding multi-threading.

II. DESIGN CONSIDERATIONS

When designing this system the important parameters that we took into consideration are resolution, frames rate of individual camera, CPU load and efficiency [2].

WebCam and PiCam that we are using have still picture resolution of 1280 x 960 and 2592 x 1944 and video supports 720p at frame rate of 30fps and 30fps respectively.

While taking the readings we are resizing the frames to 320 x 240 in order to increase the frame rate and reduce cpu load.

A. Equations

i. CPU usage(CPU load)[3][4][5]

$$CPU\ Load = \frac{(Total - PrevTotal) - (Idle - PrevIdle)}{(Total - PrevTotal)} \quad \ldots. (1)$$

Where Total is the sum of idle and Non-idle execution.

PrevTotal is previous value of Total,

$$Total = Idle + Nonidle \quad \ldots. (2)$$

$$Idle = Idle + iowait \quad \ldots. (3)$$

$$Nonidle = user + nice + system + irq + softirq + steal \quad \ldots. (4)$$

The above values are extracted from the CPU stats file in linux, the meaning of the parameters are as follows,
Cupid – number of CPU.

User – normal process execution in user mode.
Nice – niced process execution in user mode.
System – processes executing in kernel mode.
Idle – twiddling thumbs.
Iowait – waiting for I/O to complete.
Irq – handling interrupts.
Softirq – handling software interrupts.

Percentage of CPU usage can be calculated using the equation
CPU usage = (CPU Load x 100) %

The maximum available CPU is equal 1.2GHz.

ii. Frames per second(FPS)

$$FPS = \frac{Total\ Frames}{Elapsed\ Time} \quad \ldots. (5)$$

Total Frames – total number of frames captured in given time T.
Elapsed Time = T.

III. SIMULATION AND MEASURED RESULT OF EXPERIMENT

The Result is measured and validated using python scripts on raspberry pi. We are using pi camera and WebCam as load, the test is conducted with and without threading process.

The values are calculated by conducting loading test. The loading cycles are scheduled as shown in the table below.

Table 1: loading cycles

Cycles	Time(second)	Load
1	0-75	PiCam(T)+WebCam(T)
2	90-120	PiCam(NT)
3	150-180	WebCam(NT)
4	190- 250	PiCam(NT)+WebCam(NT)

NT – No Threading
T –Threaded

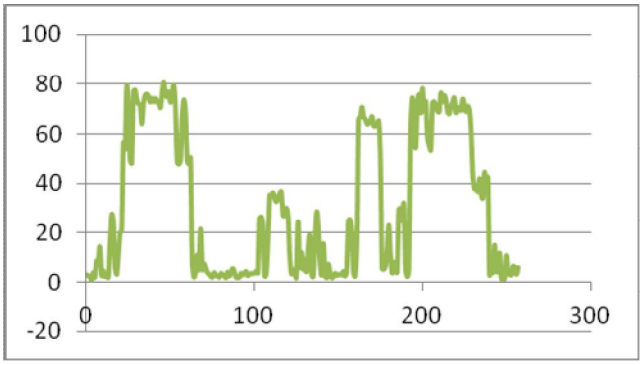


Figure 1: Plot of Time vs. load (loading cycle)

From the graph we can tell that maximum CPU usage of all the loads.

While CSI based camera(PiCam) is using 35% the USB camera(WebCam) is taking up 65% .when combined we are getting upto 80% usage. It is worth mentioning that threading has effect on CPU usage, it does not seen to reduce the CPU load.

This is because the process take same amount of resouces when running with and without threading ,but when using thread it can perform more than it does while not using thread.

The whole point in using the threading method is to increse the FPS of video stream.

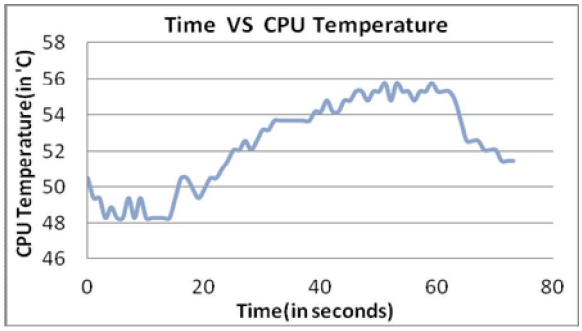


Figure 2 This plot shows the variation in temperature with time when using multiple cameras. (cycle 1)

` The temperature is increasing with CPU consumption and decreases gradually as the resources are released.

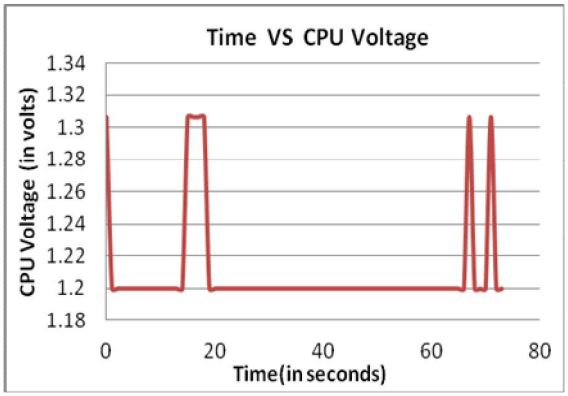


Figure 3 variation of CPU core voltage with time when a load (two camera) is applied and released (cycle 1)

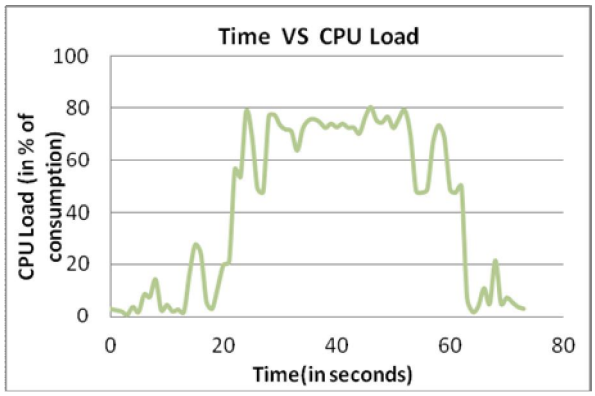


Figure 4 : CPU consumption plot for cycle 1

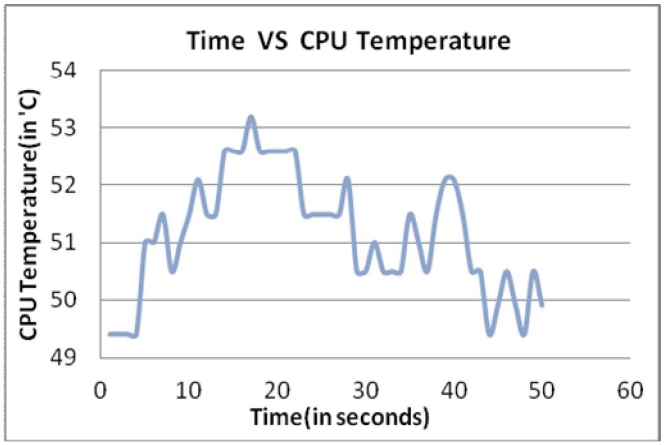


Figure 5 This plot shows the variation in temperature with time when using pi camera. (cycle 2)

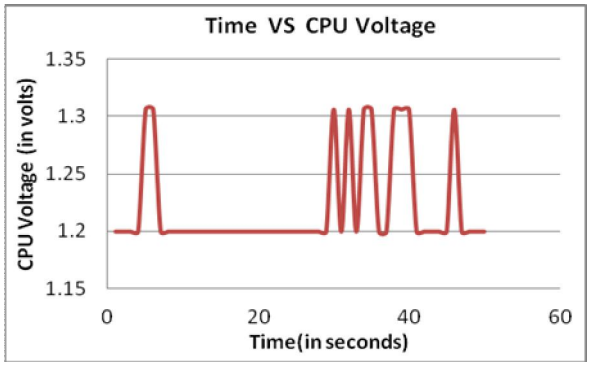


Figure 6 : variation of CPU core voltage with time when a load (pi camera) is applied and released (cycle 2)

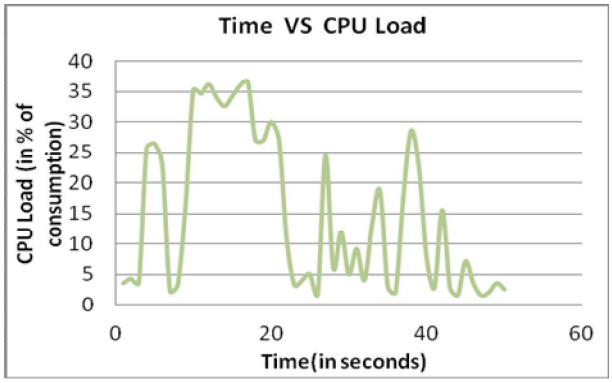
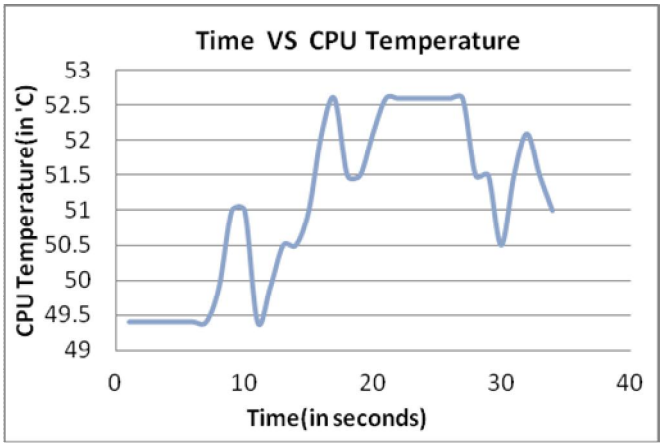


Figure 7: CPU consumption plot for cycle 2



iii. **Figure 8 This plot shows the variation in temperature with time when using web camera. (cycle 3)**

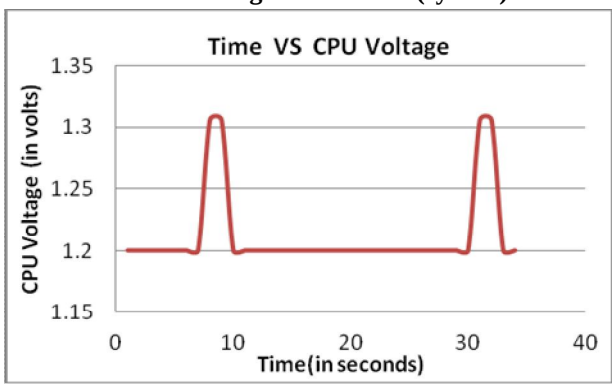


Figure 9 : variation of CPU core voltage with time when a load (web camera) is applied and released (cycle 3)

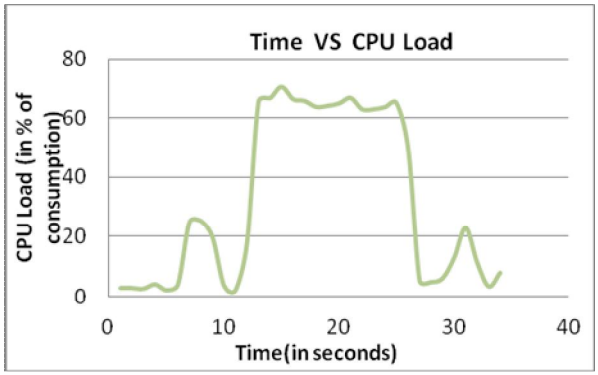


Figure 10: CPU consumption plot for cycle 3

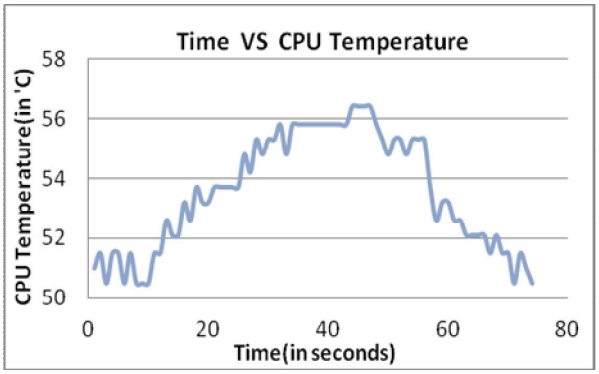


Figure 11 : This plot shows the variation in temperature with time when using multiple cameras. (cycle 4)

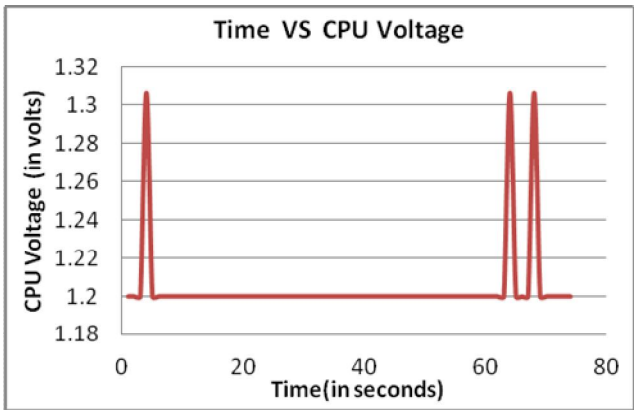


Figure 12: variation of CPU core voltage with time when a load (two camera) is applied and released (cycle 4)

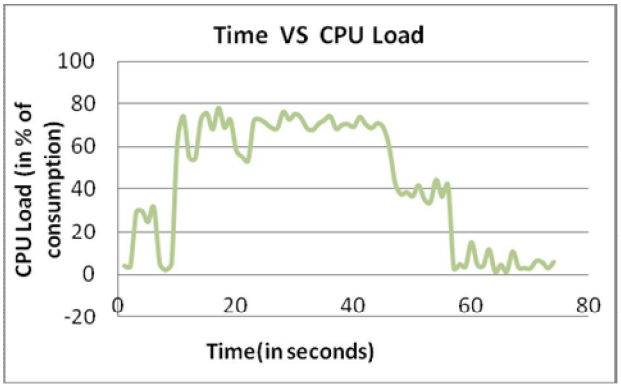


Figure 13: CPU consumption plot for cycle 4

The above plots show the variation in different parameters measured during the experiment.

The change observed in all the cycles is similar and temperature is between 50°C to 58°C.

IV. CONCLUSION

On a whole we found that it is a good practice to use threading when developing an application on any platform to get most out of it.

The performance after using separate threads for each process is high compared to running everything on main thread.

CPU cooling also makes a lot of difference in performance.

From this experiment we observed that the frame rate(FPS) calculated using equation (5) before and after the use of multithreading is 32fps and 96fps respectively for pi camera. There is about 300% increase in FPS.

REFERENCES

[1] https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2837/README.md.
[2] https://www.pyimagesearch.com/2016/01/18/multiple-cameras-with-the-raspberry-pi-and-opencv/
[3] https://www.kernel.org/doc/Documentation/filesystems/proc.txt
[4] https://github.com/pcolby/scripts/blob/master/cpu.sh
[5] http://stackoverflow.com/questions/23367857/accurate-calculation-of-cpu-usage-given-in-percentage-in-linux
[6] http://serverfault.com/questions/648704/how-are-cpu-time-and-cpu-usage-the-same
[7] http://www.webopedia.com/TERM/C/clock_tick.html
[8] http://www.pcworld.com/article/221559/cpu.html
[9] http://stackoverflow.com/questions/16726779/how-do-i-get-the-total-cpu-usage-of-an-application-from-proc-pid-stat
[10] http://www.ask.com/technology/many-times-system-clock-tick-per-second-1-ghz-processor-b9028ab0b0de7883
[11] https://github.com/torvalds/linux/blob/master/fs/proc/stat.c