Project 2 - Multi-thread programming on Raspberry Pi

Orion Peter Fernandes - 65047854

1. Project Description (2 pts)

The purpose of this project is to get familiar with multi-threading programming on embedded systems (Raspberry Pi) Two multi-thread programming methods are employed in this project –

i) OpenMP

ii) pthreads

2. Experimental Setup (4 pts)

i) Download the cany_local_omp.c from the assignment page on Canvas

ii) Transfer test.pgm file from Assignment 1

iii) Compare differences between original canny_local.c and canny_local_omp.c from this assignment

iv) Compile canny_local_omp.c with -fopenmp flag

gcc canny_local_omp.c -lm -fopenmp -o canny_omp

v) Compare performance difference between canny_local.c and canny_local_omp.c

time ./canny_omp test.pgm 1.0 0.2 0.6

vi) Modify Blur in y-direction similarly to Blur in x-direction

vii) Compile and test performance after modification of Blur in y – direction.

viii) Download canny_local_pthread.c from Canvas page

ix) Compile canny_local_pthread.c with -lpthread flag for execution

gcc canny_local_pthread.c -lm -lpthread -o canny_pthread

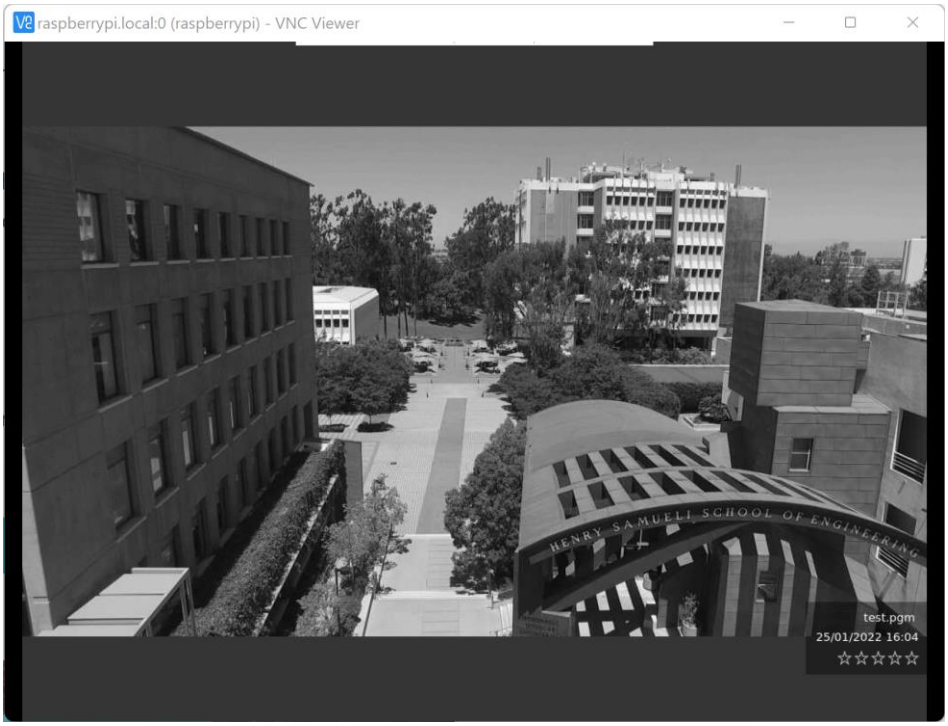
x) Compare performance differences between canny_local_pthread.c and original canny_local.c

time ./canny_local_pthread test.pgm 1.0 0.2 0.6

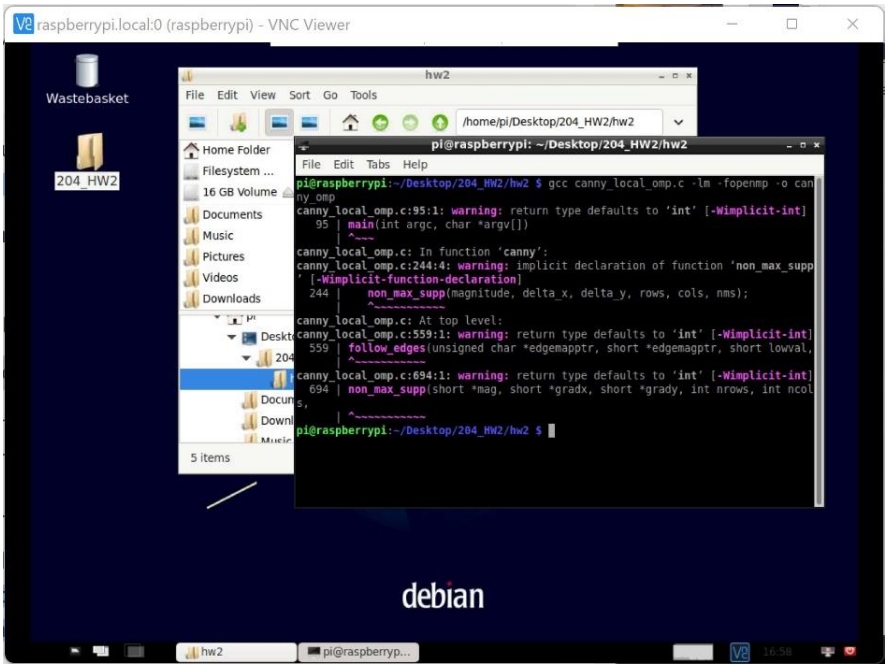
xi) Modify Blur in y-direction similarly to Blur in x-direction.

xii) Compare performance without multi-threading, with OpenMP, and with pthreads

3. Results (6 pts)

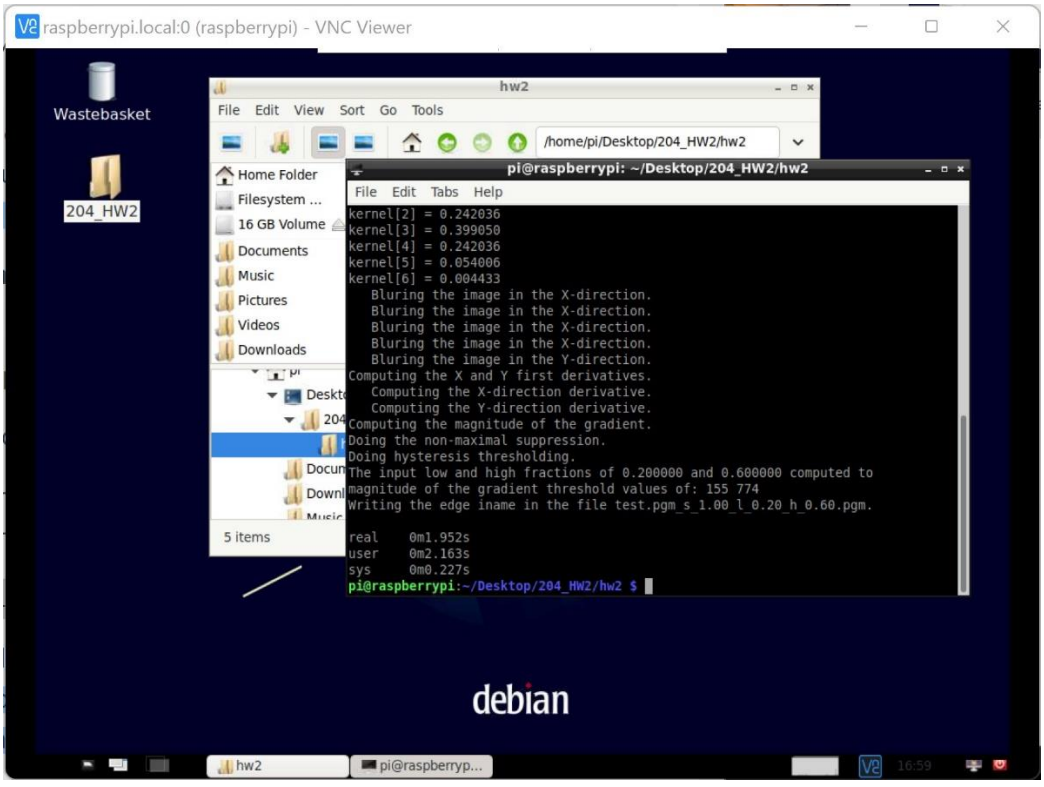


test.pgm

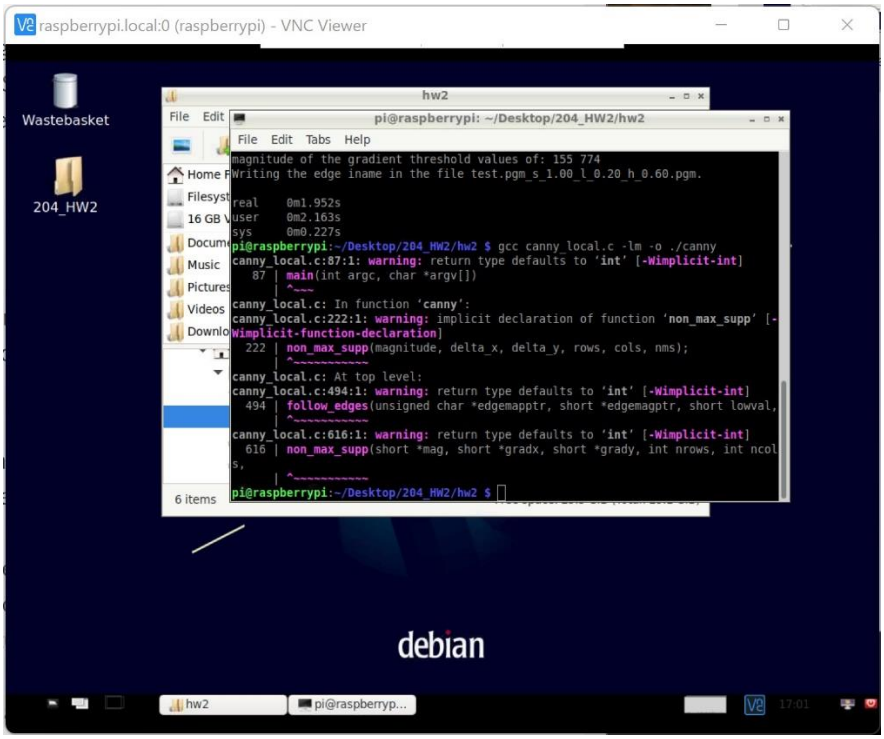


Running and compiling canny_local_omp.c

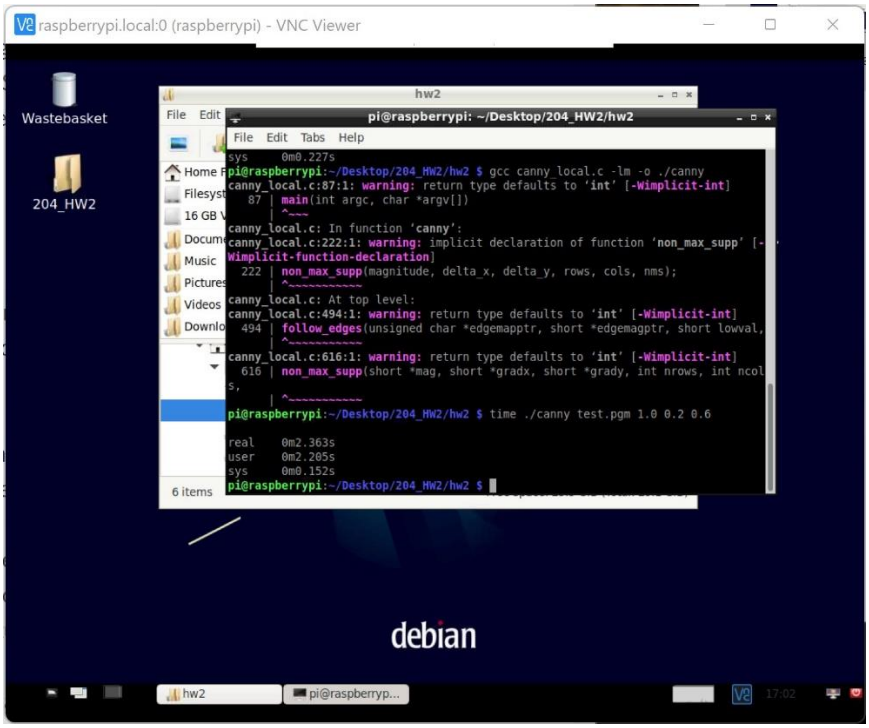
test.pgm after compilation of canny_local_omp.c



Timing measured for canny_local_omp.c



Running and compiling canny_local.c



Timing measured for canny_local.c

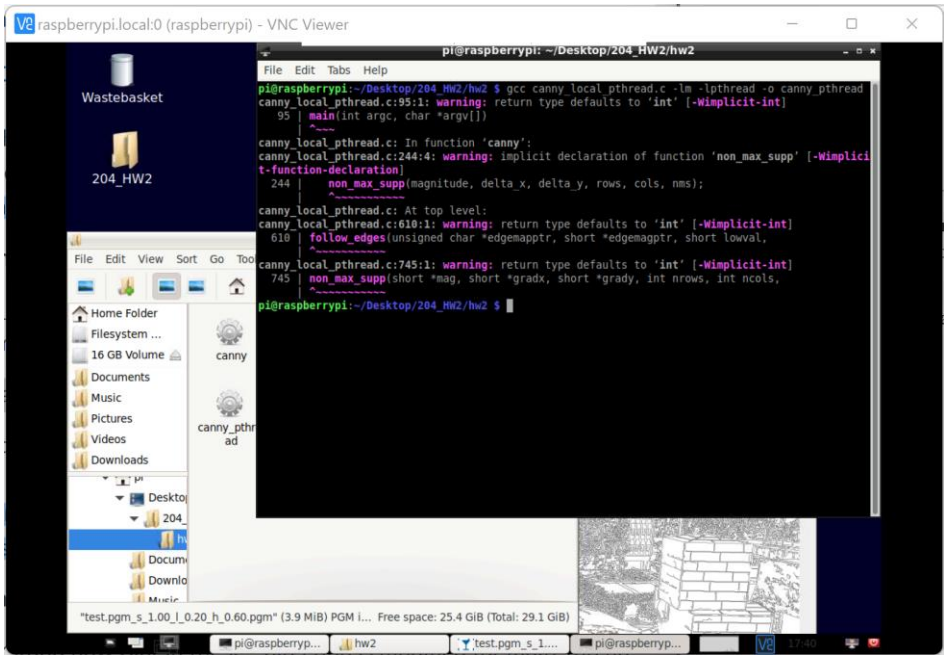
One of the benefits of OpenMP is that it coexists with compilers who don’t understand OpenMP. OpenMP supplies many types of synchronizations to help with many different situations.

The code structure for blur in x-direction under the gaussian_smooth function of canny_local.c has been modified in a certain structure to accommodate for OpenMP. The following blur in y-direction was modified to match the previous function for OpenMP.

Timing Measurements using time command

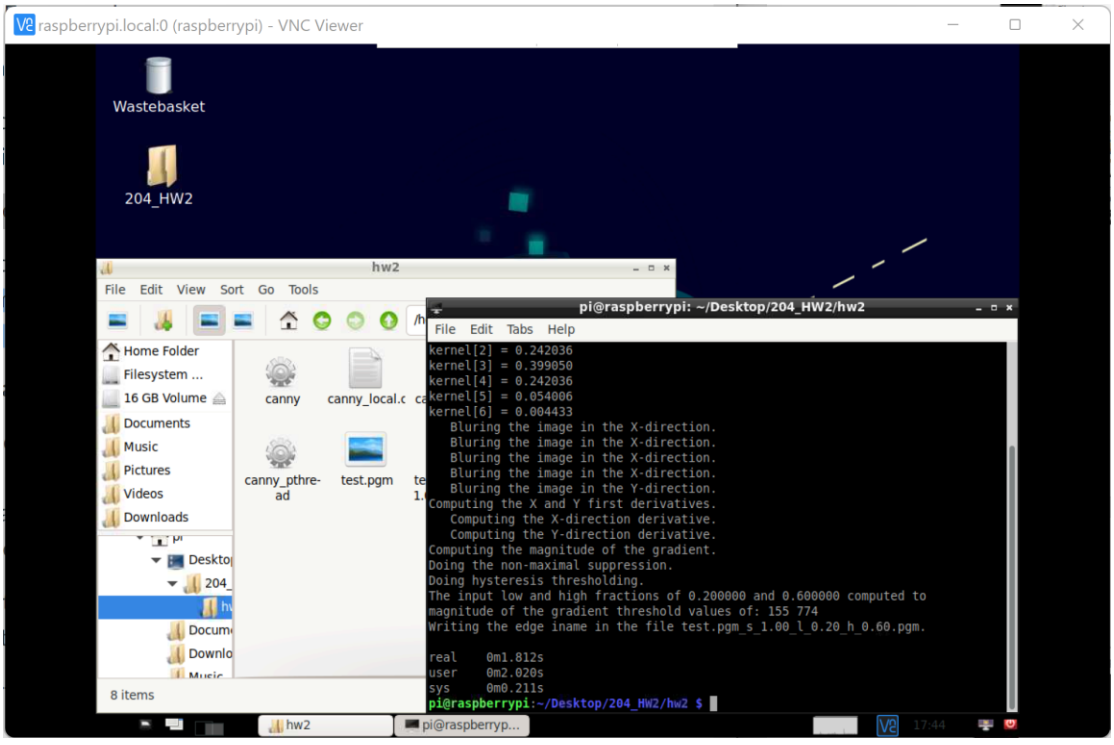
	canny_local.c	canny_local_omp.c
real	**2.363 s**	**1.952 s**
user	**2.205 s**	**2.163 s**
sys	**0.152 s**	**2.227 s**

From the above table, we can infer that canny_local_omp.c executed faster than canny_local.c except during the system execution.



Compiling canny_local_pthread.c

test.pgm produced from canny_local_pthread.c



Timing measured for canny_local_pthread.c

Pthreads are implemented as kernel threads through the kernel. Thread library may still be extensively involved in synchronizing threads, but the kernel handles thread creation and scheduling. They have the following advantages:

- Can take full advantage of multiprocessor architecture within a single process.
- No latency during system service blocking (any more than between processes.)

The code structure for blur in x-direction under the gaussian_smooth function of canny_local_pthread.c has been modified to introduce pthreads for multithreading. The following blur in y-direction function was modified to match the previous function for pthread functionality.

Timing Measurements using time command

	canny_local.c	**canny_local_pthread.c**
real	**2.363 s**	**1.812 s**
user	**2.205 s**	**2.020 s**
sys	**0.152 s**	**0.211 s**

From the timing table above, we can determine that the inclusion of pthreads for multithreading within the canny code has resulted in faster execution times except for the system similar to the measurements of OpenMP.

4. Problems and Discussion (6 pts)

- The execution of the canny_local.c, canny_local_omp.c and canny_local_pthread.c all executed with warnings which aren't always a good thing during execution.
- Yes, the results agree with my expectation after inferring the observations from the time command. Multi-threading helps execute code faster except for 'sys'.
- The results agree with my first expectations since multithreading methods employed for the canny code were expected to make code execution faster which have been observed after compilation.
- The methodology of this project could be improved by executing each modified canny file right after the other to note down timing results and note observations.

- **Exteral source used for OpenMP -** https://docs.microsoft.com/en-us/archive/msdn-magazine/2005/october/openmp-and-c-reap-the-benefits-of-multithreading-without-all-the-work#:~:text=One%20of%20the%20benefits%20of,that%20don't%20understand%20OpenMP.&text=With%20multiple%20threads%20running%20concurrently,help%20in%20many%20different%20situations.
- **External source for pthreads -** http://www.yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html

5. Conclusion (2 pts)

From this project, one can conclude that implementation of pthreads and OpenMP greatly speed up execution time through multithreading since the resources are allocated much more efficiently and also executes processes concurrently rather than sequentially which makes efficient use of space (threads) to complete the same process within a shorter interval of time resulting in better throughput.