REPORT

LULAMILE PLATI

PLTLUL001

CSC2002S Assignment PCP1 2022

University of Cape Town • iDyunivesithi yaseKapa • Universiteit van Kaapstad •

SPES BONA

Introduction

The aim of this Assignment is to implement two filters, a mean and median filter both Parallel and Serial for smoothing RGB colour images. A mean filter sets pixels in the images to the average of the surrounding pixels, the median filter sets the pixels in the images to the median of the surrounding pixels. In this Assignment we will be timing the execution of both parallel and serial programs in both filters to determine if the parallel programs are faster than serial programs in both filters.

Parallel Programming uses multiple processors to solve a problem, it breaks the problem into parts and executes it parallel. In this assignment we will use java Fork/Join frameworks.

Classes:

All the four classes contain the following methods, and they are the same in all four of them.

- setImage (Type - void) – This is method uses the input Image Name to read the image using BufferedImage class and prints “Image not Found” if the input image name is not found.
- meanFilter (Type - 2D ArrayList of an integer Array)– This method uses the 2D ArrayList of an integer Array and stores the pixels of the image that will be filtered in a 2D ArrayList of an integer Array and return this 2D ArrayList.

MeanFilterSerial Class

This serial class performs the mean filter it has a 3-parameter constructor which takes the name of the input image, the name of the output image and a window Width (inputImage, outPutImage and window Width) respectively. This class has 3 Methods, setImage, meanFilter and applyMeanFilter.

- applyMeanFilter (Type - 2D ArrayList of an integer Array)- This method takes a 2D ArrayList of an integer Array as its parameter, this will be the 2D ArrayList of the input image pixels. This method uses 4 integer arrays for the 4-pixel components alpha, red, blue and green all of size (window Width)^2 . The first 2 for loops are for iterating through the 2D ArrayList of the input image pixels the second for loops are for the windowWidth*windowWidth (e.g., 3x3) window, the average of the RGB components of each pixel in this window are calculated and then stored to a 2D ArrayList of the new Pixels that will be used to create a new smoothed image.

MedianFilterSerial Class

This serial class performs the median filter, and it also has a 3-parameter constructor which takes the name of the input image, the name of the output image and a window Width (inputImage, outPutImage and window Width) respectively. This class has 3 Methods, setImage, medianFilter and applyMedianFilter.

- applyMedianFilter (Type - 2D ArryList of an integer Array)- This method takes a 2D ArryList of an integer Array as its parameter, this will be the 2D ArryList of the input image pixels. This method uses 4 integer arrays for the 4-pixel components alpha, red, blue and green all of size (windowWidth)^2 . The first 2 for loops are for iterating through the 2D ArrayList of the input image pixels the second for loops are for the windowWidth*windowWidth (e.g., 3x3) window, then finds the median of the ARGB components of each pixel in this window by sorting the 4 integer Arrays and finding the median of each Array then converting them to new pixels and then store the new pixels to a 2D ArrayList of the new Pixels that will be used to create a new smoothed image.

MeanFilterParallel and MedianFilterParallel Classes

These classes perform the mean and median filters parallel using the Fork/Join frameworks they both have a 3-parameter constructor which takes the name of the input image, the name of the output image and a window Width (inputImage, outPutImage and window Width) respectively. These classes both have 2 Methods setImage, meanFilter, an Inner Class MeanFilterThread for the MeanFilterParallel class and MedianFilterThread inner class for the MedianFilterParallel class.

- MeanFilterThread and MeanFilterThread Inner Classes
    - These classes extend RecursiveTask with a return type of a 2D ArrayList of an integer Array which will be the new Pixels to write a new smoothed image. These classes both have a 5-parameter constructor, a 2D Array of an input image pixels, output image name, a window Width, a start value (lo) and an end value (hi).
    - The compute method is similar to the applyMeanFilter and applyMedianFilter method in the MeanFilterSerial and MedianFilterSerial classes it follows the same format to filter an image, but this method does not have a parameter it uses the 2D ArrayList from the constructor. This method first compares the difference of hi and lo to the SEQUENTIAL CUTOFF and if the SEQUENTIAL CUTOFF is greater than (hi-lo) then the program will execute sequentially but if the SEQUENTIAL CUTOFF is less than (hi-lo) then the program will execute using divide-and-conquer Parallel algorithm using the Fork/Join frameworks to speed up the execution.

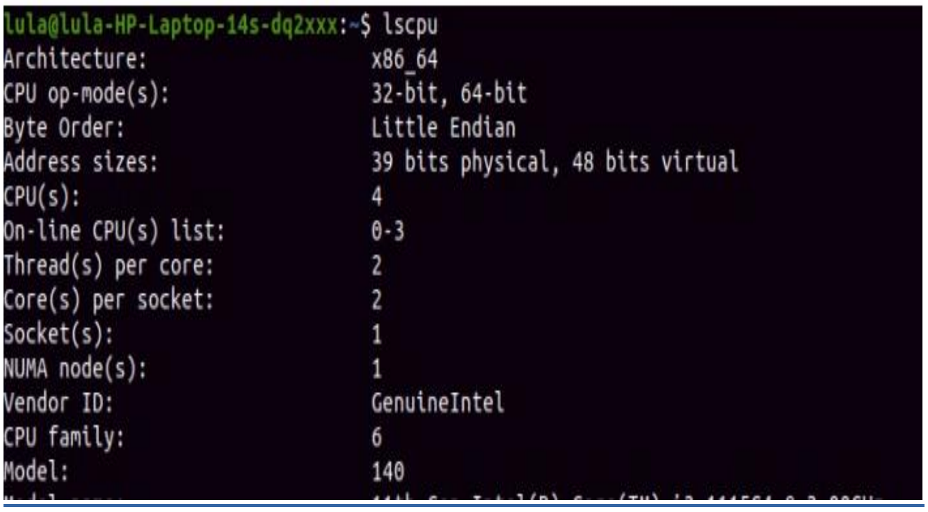
VALIDATION and TIMING

To show if my results are correct, I went online to see if there are programs similar my serial programs, I compare my code to the ones I could find. After correcting my code, I then used an image similar to the one in the Assignment example to check if the image was smoothed after applying the filters and both filters worked correctly.

I then compared the time the serial filters took to execute to the time the Parallel filters took, I used *currentTimeMillis ()* to time my programs. To make sure that I timed them correctly, I timed them after reading the input image and stopped the timer before writing the new smoothed pixels to the output image, I did this to limit the time of execution by not including the time for creating and calling the BufferedImage objects.

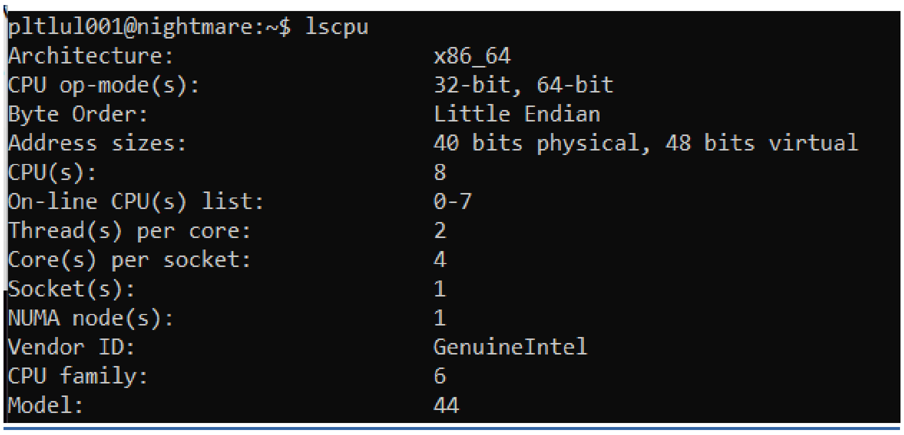
MACHINE ARCHITECTURES

The laptop I was using



```
lula@lula-HP-Laptop-14s-dq2xxx:~$ lscpu
Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
Address sizes:                   39 bits physical, 48 bits virtual
CPU(s):                          4
On-line CPU(s) list:             0-3
Thread(s) per core:              2
Core(s) per socket:              2
Socket(s):                       1
NUMA node(s):                    1
Vendor ID:                       GenuineIntel
CPU family:                      6
Model:                           140
```

Nightmare



```
pltlul001@nightmare:~$ lscpu
Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
Address sizes:                   40 bits physical, 48 bits virtual
CPU(s):                          8
On-line CPU(s) list:             0-7
Thread(s) per core:              2
Core(s) per socket:              4
Socket(s):                       1
NUMA node(s):                    1
Vendor ID:                       GenuineIntel
CPU family:                      6
Model:                           44
```

OPTIMAL SERIAL THRESHOLD

I chose my SEQUENTIAL CUTOFF for both MedianFilterParallel and MeanFilterParallel classes by using different sequential cut-offs and different window Widths, I found out that when using greater window Widths my programs took longer to run and the time decreased when I used 600 SEQUENTIAL CUTOFFS for the MeanFilterParallel and 1000 SEQUENTIAL CUTOFF for the MedianFilterParallel

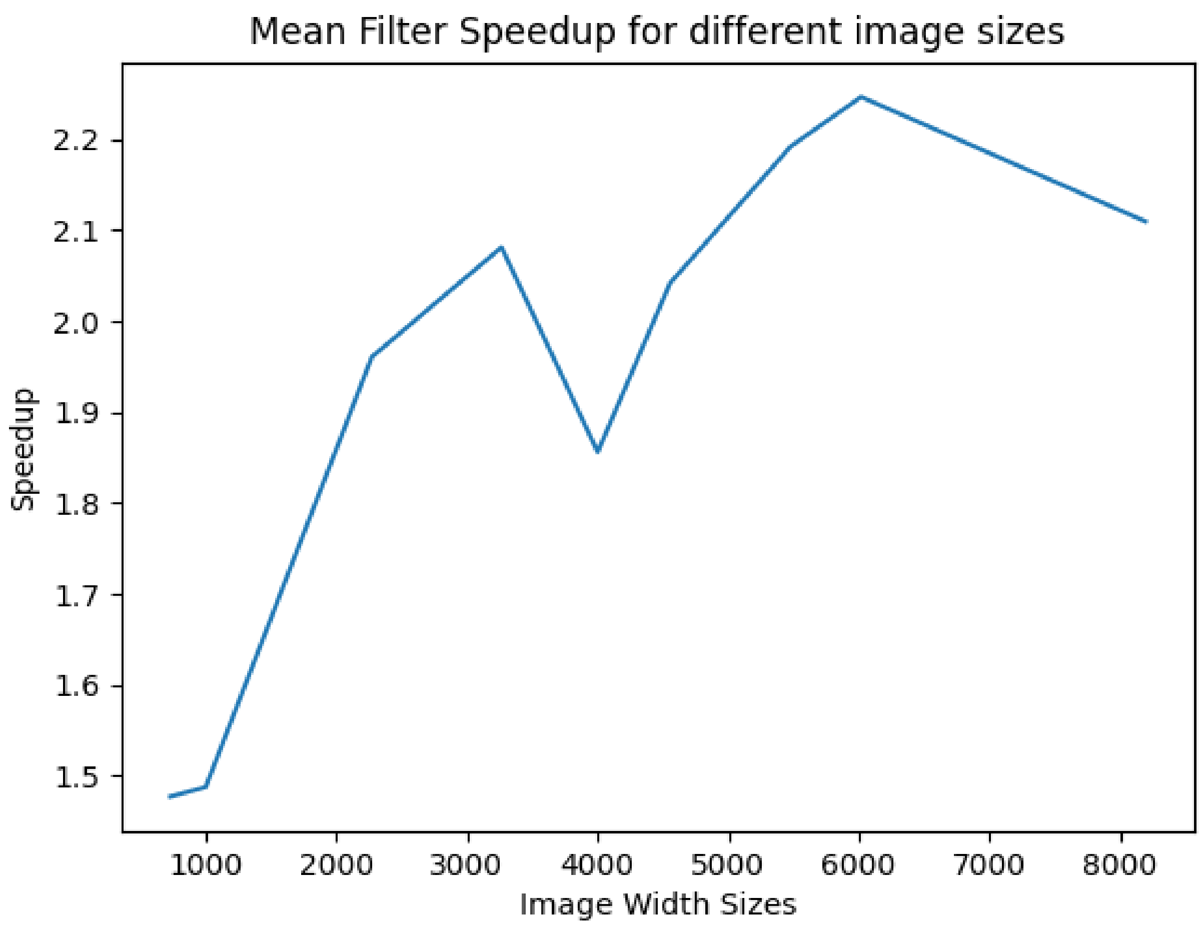
PROBLEMS/DIFFICULTIES

- I had problems when running my serial programs with a window Width that is greater than eighty-one the programs would take more than 5 minutes to execute, and my laptop would overheat.
- I had a problem with implementing the parallel programs with RecursiveAction I ended up converting my method from void to have return types and I used RecursiveTask instead.
- I did not use images with sizes greater than 8000 in the MedianFilter because the medianFilter took longer to smooth these pictures I decided to end at 6000 image size width for the medianFilter.

RESULTS

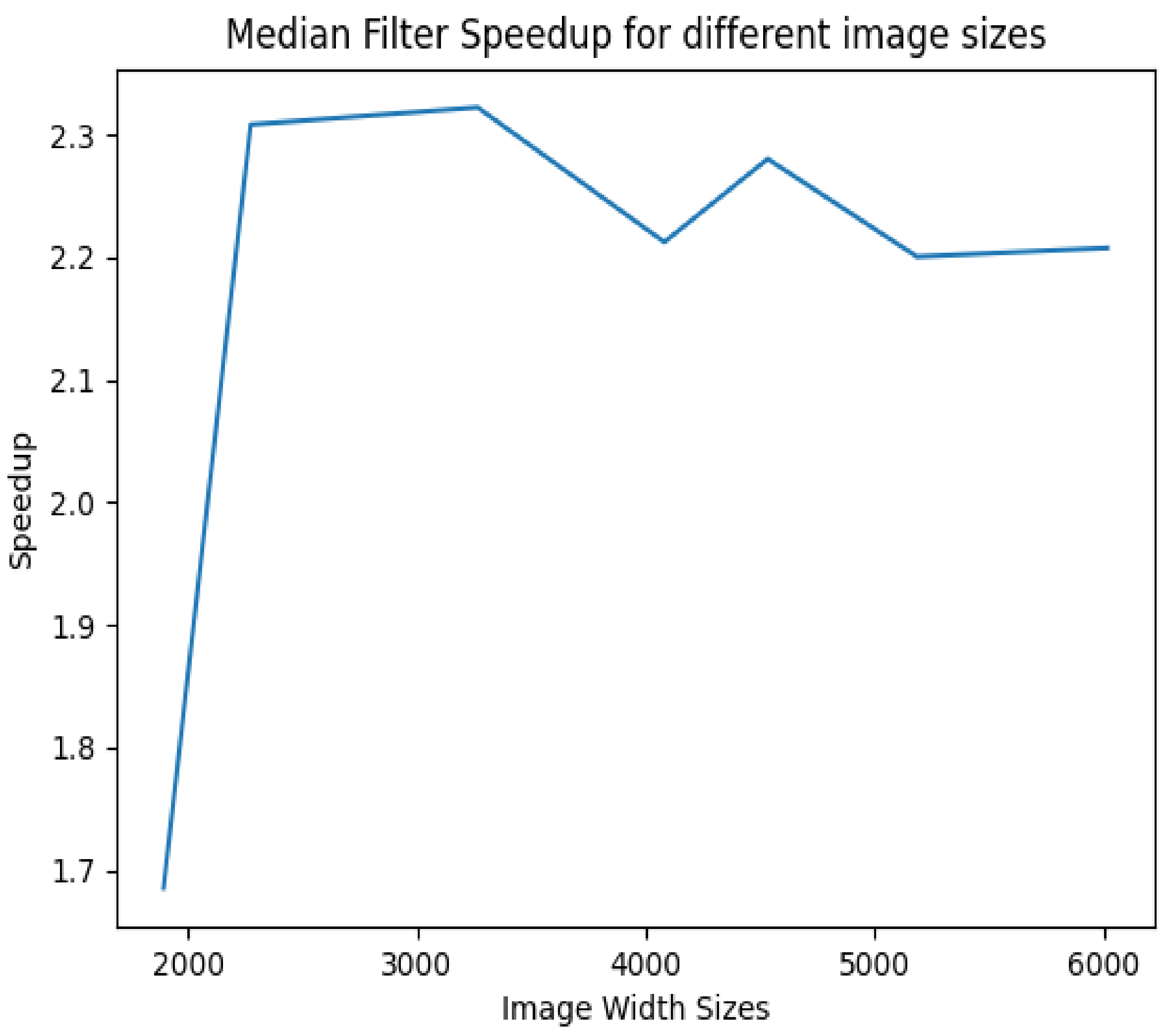
MEAN FILTER RESULTS

Size (width size)	Window Width	Serial Time (in milliseconds)	Parallel Time (in milliseconds)	Speedup (Serial/Parallel)
8192	21	96975	45964	2.10980332
6016	21	52133	23203	2.24682153
5476	21	42292	19293	2.1920904
4554	21	29366	14382	2.04185788
4000	21	22146	11932	1.85601743
3264	21	16961	8149	2.08135968
2272	21	8088	4125	1.96072727
1000	21	818	550	1.48727273
728	11	223	151	1.47682119



MEDIAN FILTER RESULTS

Size (width size)	Window width	Serial Time (in milliseconds)	Parallel Time (in milliseconds)	Speedup (Serial/Parallel)
1891	11	27383	16247	1.68541885
2272	11	26134	11322	2.30824943
3264	11	44696	19247	2.32223204
4080	15	147268	66563	2.21246038
4532	15	181234	79479	2.28027529
5184	15	218069	99098	2.20053886
6016	15	329742	149369	2.2075665



- The mean filter parallel algorithm perfoms well when the width sizes are greater than 4000 and when the filter size (window Width) is greater less or equal to 25. The maximum speedup for the parallel mean filter algorithm is ***2.24682153*** with image width 6016.
- The median filter parallel algorithm performs well when the width size is less that 3500 and the filter size is less than 20. The maximum speedup for the parallel median filter algorithm is ***2.32223204*** with an image width 3264.
- The parallel mean filter is faster in smoothing big images than the parallel median filter.

I did not expect the speedup for the parallel mean filter to decrease around 4000 width size and then increase to the maximum speedup. My parallel algorithms would be faster in when runnuing with the machine with more cores than the machines I used.

Conclusion

It was worth using the parallel algorithms because they are faster than the serial programs. The parallel median filter always takes atleast half the time the serial algorithm takes to execute. The mean parallel filter is always faster than the mean serial algorithm especially with the greater image widths. I think the parallel algorithms both worked correctly although they are not the fastest algorithms but the are faster than the serial algorithms. Therefore I would say that it was worth using the parallelization.