REPORT

CSC2002S: Assignment 1

ORATILE RAPOO

RPXORA001

1.EXPERIMENT AIM

The aim of this experiment is to determine conditions where parallel programs are worth using and how they perform compared to sequential programs, we are going to do this by filtering noise from images using two filtering techniques median Filter and mean Filter this will be implemented in parallel and a sequentially with different image sizes and window size.

2.Hypothesis

Since Parallel programs divide work among threads, they are usually expected to perform faster than sequential programs. But in some cases, parallel programs are slower and more complicated to implement than sequential programs but this depends on the machine architecture, data size and other variables.

3.Method Section

(A) Sequential Approach

- MeanFilterSerial.java

  This class reads an image and then set each pixel in the image to the average of its neighboring pixels, it does this sequentially by setting one pixel at the time according to the size filter used and then writes the filtered image.

  - Instance Variables
    - `inputImageName`→ this holds the string input image name and path
    - `outputImageName`→ this holds the string output image name and path
    - `windowWidth`→ this holds integer size of window e.g 3x3 that is odd
  - Methods
    - Read_image_And_SetWidthHeight→this method reads an image and then sets the image width and height
    - Set_Mean_filter_two→this method takes and image and apply the mean filter to it according to the specified filter size

        - is_windowWidth→checks that the filter size is odd and >=3
        - Write_image→ this method writes new filtered image to path

- MedianFilterSerial.java

  This class reads an image and then set each pixel in the image to the median of its neighboring pixels, it does this sequentially by setting one pixel at the time according to the size filter used and then writes the filtered image.
    - Instance Variables
        - `inputImageName`→ this holds the string input image name and path
        - `outputImageName`→ this holds the string output image name and path
        - `windowWidth`→ this holds integer size of window e.g 3x3 that is odd
    - Methods
        - Read_image_And_SetWidthHeight→this method reads an image and then sets the image width and height
        - Set_Mean_filter_two→this method takes and image and apply the median filter to it according to the specified filter size
        - is_windowWidth→checks that the filter size is odd and >=3
        - Write_image→ this method writes new filtered image to path

(B) Parallel Approach

Instead of doing one task at a time I am dividing the work into smaller pieces implementing a divide and conquer algorithm and I am using the Fork/Join Framework to create threads that will do the work that is divided, and this work will be done in parallel, rather than having to wait for one task to complete in order for another to start.

Parallel Algorithm

My parallel programs extends RecursiveAction that is used to compute in parallel. My programs then override the Compute() method which will hold the logic of my parallel program. The program uses a sequential cut-off to determine if the work is enough to split or not and if the work is less than the sequential cut-off then work filter can be applied on set of pixels but if the work is more than the sequential cut-off then it is split into two parts left and right with new instances. The left is forked (fork() method) and the right is compute(compute() method) and then the left is joined (join() method) this ensures thread safety and prevents race conditions. In my main I used the ForkJoinPool to create a pool of worker threads then used pool invoke start everything running and to give the task to the pool.

- MeanFilterParallel.java
    - Instance Variables
        - `inputImageName`→ this holds the string input image name and path
        - `outputImageName`→ this holds the string output image name and path
        - `windowWidth`→ this holds integer size of window e.g 3x3 that is odd
        - low→ Start of the dataset size used to divide work
        - high → end of the dataset size used to divide work
    - Methods
        - Read_image_And_SetWidthHeight→this method reads an image and then sets the image width and height
        - Compute→ this method uses a sequential cut-off to divide the work into smaller pieces and will keep on creating new threads until cut-of is met and then it will apply a mean filter to data set this is our parallel algorithm.
        - is_windowWidth→checks that the filter size is odd and >=3
        - Write_image→ this method writes new filtered image to path

- MedianFilterParallel.java
    - Instance Variables
        - `inputImageName`→ this holds the string input image name and path
        - `outputImageName`→ this holds the string output image name and path
        - `windowWidth`→ this holds integer size of window e.g 3x3 that is odd
        - low→ Start of the dataset size used to divide work
        - high → end of the dataset size used to divide work
    - Methods
        - Read_image_And_SetWidthHeight→this method reads an image and then sets the image width and height
        - Compute→ this method uses a sequential cut-off to divide the work into smaller pieces and will keep on creating new threads until cut-of is met and then it will apply a median filter to data set this is our parallel algorithm.
        - is_windowWidth→checks that the filter size is odd and >=3
        - Write_image→ this method writes new filtered image to path

Validation Of My Parallel Alogorithm

Checked if it produced the same and correct output as the serial programs across all test cases even with a threshold of 1 and threshold that is equal/greater than dataset size and it produced the same output.

Timing Of My Parallel Algorithm

I used System.currentTimeMillis() method for timing the startTime of my parrallel program. Then I used invoke to start my parallel and after they were done running I used System.currentTimeMillis() – startTime to calculate elapsed time after the method is run and this elapsed time is the time it took my parallel method to run and in my timing I did not include the time it takes to read the image/write the new image I only recorded time for my parallel method.

optimal serial threshold (sequential cut-off)

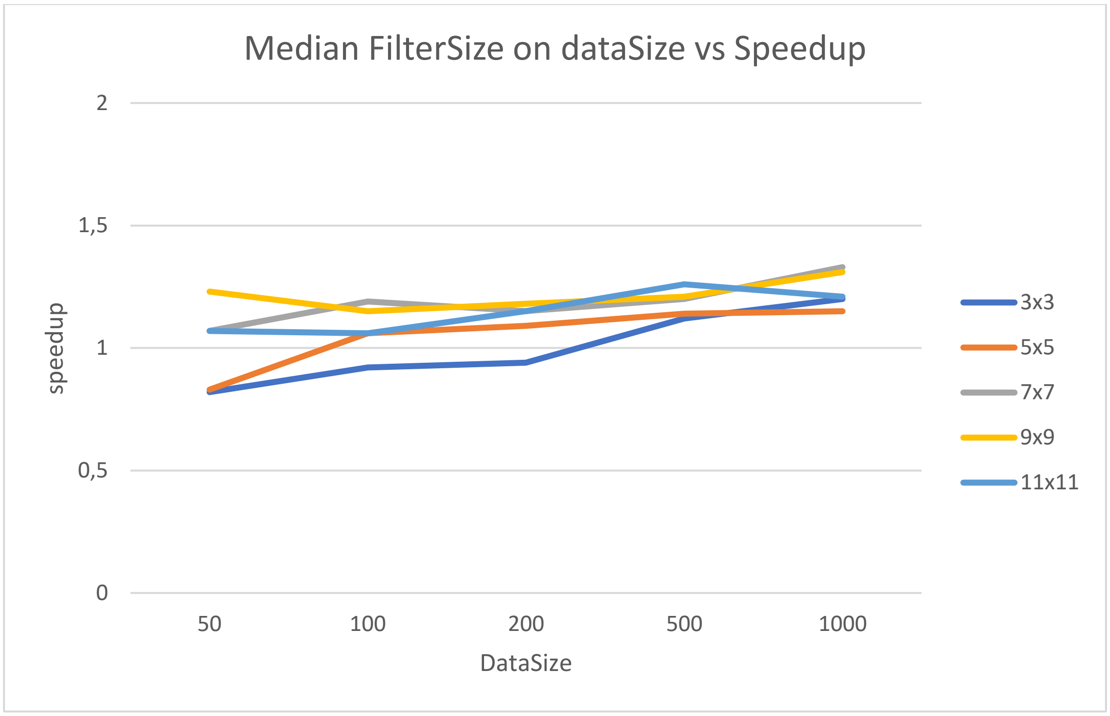
While experimenting I found out that using a quarter of the dataSize(width/height of the picture) gives the best performance. When the cut-off(threshold) was too high or above the dataSize the program only implemented the sequential part and when the cut-off(threshold) was too low the program used many threads to compute small data which resulted in the program being slow. So, the threshold(cut-off) should not be too low or too high which is why my threshold is a quarter of the dataSize.

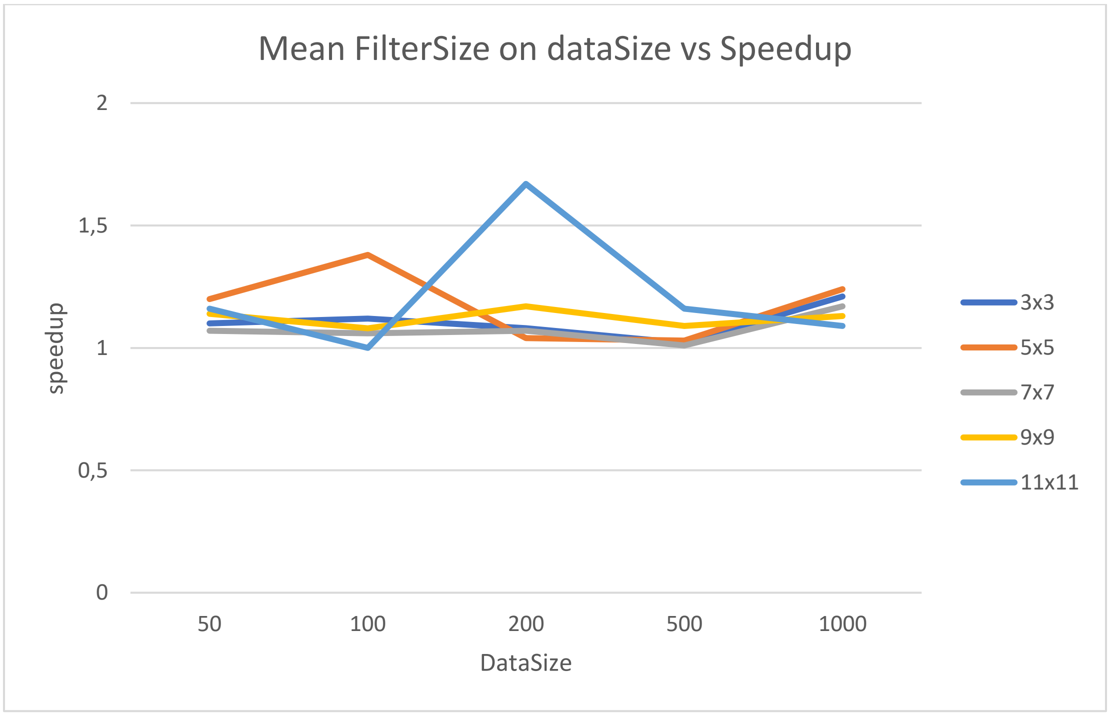
Problems/difficulties

- When I was using small dataset(small pictures) my parallel programs were slower than my sequential programs I was not getting any speedup. I was not sure if this was because of the machine I'm using or threshold I chose.
- When I started to increase the dataset(picture size) my parallel programs started to perform faster than my sequential programs achieving speedup of nearly x1,4.
- I suspected that background programs and previous tests affected my results.

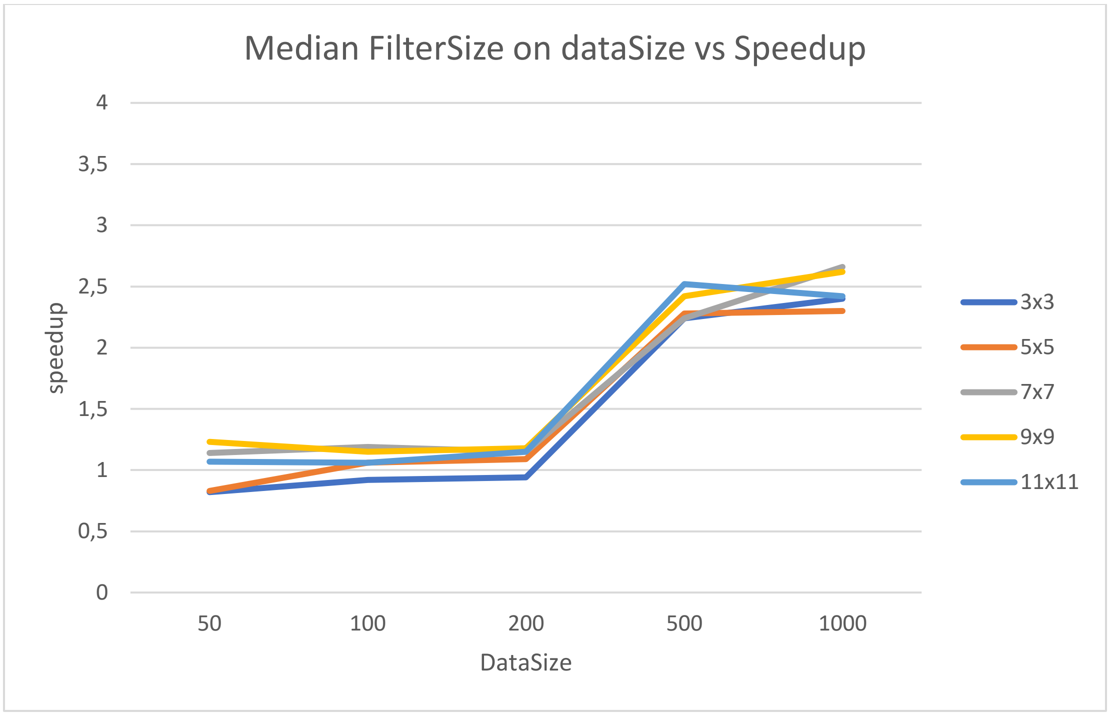
Results Section

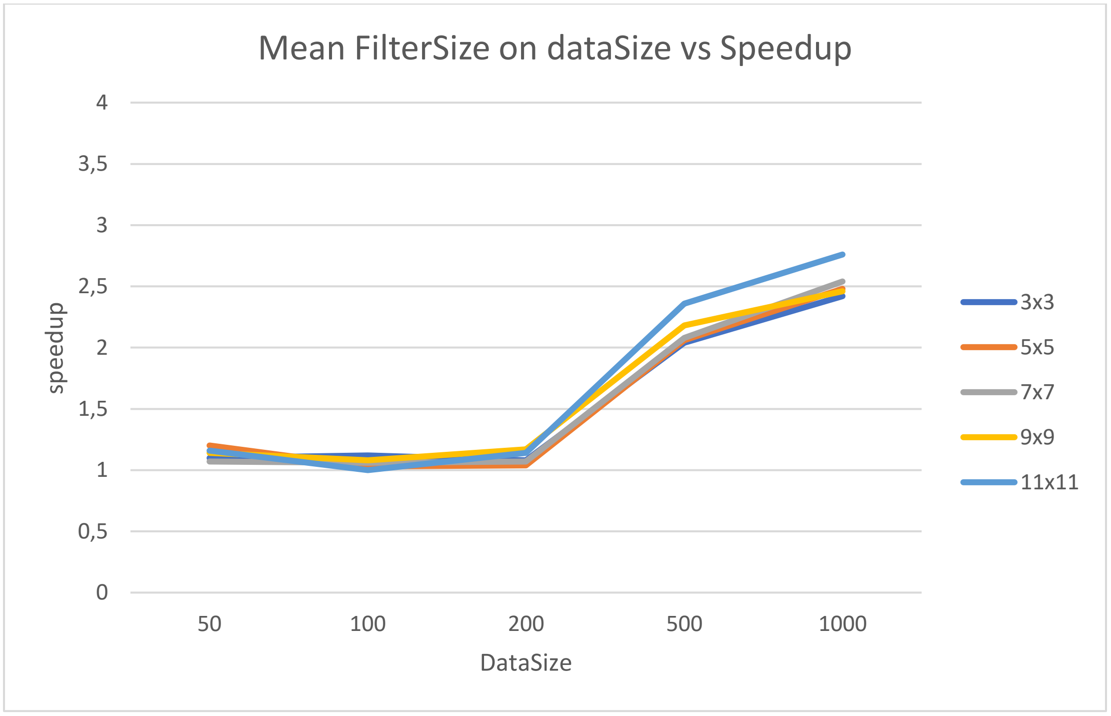
Graphs on a 2 core machine





Graphs on a 4 core machine





Discussion:

- Both Median and Mean parallel programs got a bigger speed up as the dataset increased as expected.
- While testing on the 2 core machine the speed up was significantly less than when testing on the 4 core machine, so the architecture of the machine determines the speedup.
- Using a bigger filter size takes more time to compute and sometimes makes the parallel program slower than the sequential when using less dataSize on a weaker machine.
- This graphs proves that the median filter is takes a longer time to compute than the median filter
- Parallel speedup is dependent on mainly the datasize and the architecture of the machine, as the 2 core machine proves that too much data on a less powerful machine does not achieve that big speedup for example highest speedup I got on 2 core machine was 1.42 compared to the 2.8 I got from the 4 core machine.
- My parallel programs best performance is for dataSize greater than 200
- Through trial and error I got to the conclusion of using a sequential cut-off of width/4 this works best for machines that have 4 or more cores and its not too low or high for most dataSize.
- Biggest speedup I got for median is 1.42 on 2 core which is nearly close to ideal of 2 and I got 2.8 on 4 core which is close to the ideal 4.
- The maximum speed up for MeanFilterParallel on 2 core is 1.78 on 200 data set this is an anomaly as it is faster than bigger dataSize and is too close to the ideal of 2, this may because of the cache because I was running each test 5 times.
- My results are reliable because I ran each test five times and recorded the lowest but I suspect that previous tests might have affected the results of other tests.

Conclusion:

My conclusion it that parallel programing depends the size of the data , the machine architecture as it is dependent on the number of cores your machine have and you also need to have a suitable threshold(sequential cut-off) that ensures that work is divided properly resulting to parallelism, if your sequential cut-off is too low you will use more resources than you need making the program slower and more complex to manage also if your cut-off is too high or higher than dataSize then the program will run sequentially. In conclusion you parallel programing when dealing with large amount of data or computation on a machine architecture with more cores.