Name : Anele

Surname :Danisa

Year : 2022

2D Median and Mean Filter

Introduction

It is never easy doing things alone, especially when you have a lot of work to do. Looking at computers it is even difficult to run one program to handle a lot of data. It might end up crushing at some point and that type of programming is called **sequential programming**.

What if given the same large data we just divide that data into smaller chunks of tasks and make all those tasks to execute at the same time and type of programming is called **parallel programming**.

Wait a minute is that even possible, well it is, since Java has provided us with a library that can help us work with Recursive actions and tasks. Java also provided us with a **forkjoin framework** which also is responsible for

forking the work into pieces and waiting for each task to finish and then join the results together.

Using these java tools we can do exciting things while saving waste of time like filtering the image using a **mean or median filter** where mean just simple make use of the window and the average of all the surrounding pixels while median make use of the window but it first sort the pixels and set each surrounding pixel to the median.

Aim

The Aim of this experiment is to investigate the importance of java fork-join framework parallel programming over sequential programming by filtering a 2D image with median and mean sliding square window method .

Methods Section

parallelisation algorithms

To handle all the parallel work two different algorithms versions were implemented, a Median Filter and Mean Filter version. The Median filter was handled in the MedianFilterParallel.java file and Mean filter was handled in MeanFilterParallel.java file.

MedianFilterParallel.java File

This file has three classes inside the main.java class, MedianFilterP.java class and Image.java class.

- **The main.java class** - Has a main method only which Handles the user input as a command line argument. After taking the input it then reads the pixels of the image per window making use of the java.awt.image.BufferedImage java class and adds the pixels to the Array and adds that array to the main ArrayList of windows.
  It then loops through the main arraylist, passes the windows to MedianFilterP.java for filtering , calls ForkJoinPool.commonPool().invoke to initialize the library and start the computation of the MedianFilterP.java object, gets the filtered results and simply passes

them with the output image file and Image Object as arguments for the image writing process.

- **MedianFilterP.java class -** Handles the production of filtered pixels, This class extends the RecursiveAction java class which helps in dividing the window Array into small chunks of work recursively until the SEQUENTIAL_THRESHOLD is reached and start the process of sorting the small chunks arrays and joining them that process is handled in the **compute()** method. To do the entire work it makes use of other two helper methods which are the **Addlist()** method which simply adds the two sorted lists and the **getDest()** method which gets the median of the sorted array and produces the filtered pixels and returns them as an Array.

- **Image_Controller.java class -** Handles the last process of setting the filtered pixels back to the image, writing them to the image and producing a new output file of the filtered image the entire process is done on the **Writeimage()** method.

MeanFilterParallel.java File

This file has three classes inside the main.java class, MedianFilterP.java class and Image.java class.

- **The main.java class** - Has a main method only which Handles the user input as a command line argument. After taking the input it then reads the pixels of the image per window making use of the java.awt.image.BufferedImage java class and adds the pixels to the Array and adds that array to the main ArrayList of windows.
  It then loops through the main arraylist, passes the windows to MedianFilterP.java for filtering , calls ForkJoinPool.commonPool().invoke to initialize the library and start the computation of the MedianFilterP.java object, gets the filtered results and simply passes them with the output image file and Image Object as arguments for the image writing process.

- **MeanFilterP.java class -** Handles the production of filtered pixels, This class extends the RecursiveAction java class which helps in dividing the window Array into small chunks of work recursively until the SEQUENTIAL_THRESHOLD is reached and start the process of summing the small chunks arrays and joining in terms of summing them that process is handled in the **compute()** method. To do the entire work it makes use of other one helper method, the **getDest()** method which takes a sum and calculates the mean/average of the sorted array and produces the filtered pixels and returns them as an Array.

- **Image_Handler.java class -** Handles the last process of setting the filtered pixels back to the image, writing them to the image and producing a new output file of the filtered image the entire process is done on the **Writeimage()** method.

Validation

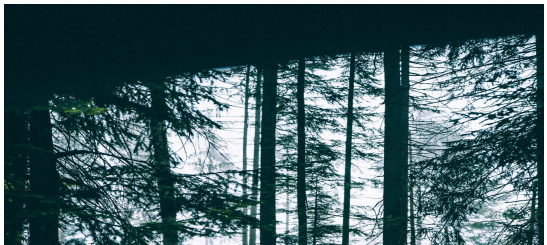
To prove that my algorithms were correct I passed the input image and the window size as a command and all programs produced the filtered image output . Both the original and output picture for median and mean are shown below.

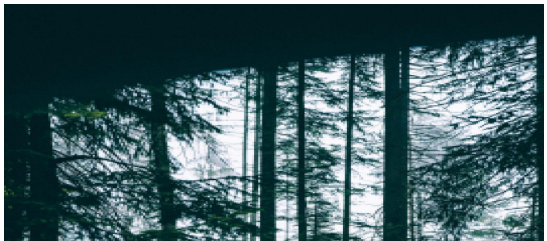
MedianFilter

Original_Image

Output_Image

MeanFilter





Original_Image Output_Image

Timing The Program

All the programs were timed using the benchmarking approach and that works by timing the execution of both serial algorithms and parallel algorithms across a range of image sizes.I used the java library method `System.currentTimeMillis()` which returns the time in millisecond.

For the parallel programs I timed the program from the beginning of the for loop that creates the filtered windows using the parallel programming approach and ended the timing when it exits the loop and for Serial programs I used the same approach for timing the only difference is that the serial program uses the Sequential approach to filter the windows.

.

Parallel programs

```
*This is where the program Timing starts.
long start1 = System.currentTimeMillis();

ArrayList<int[]> dest_windows = new ArrayList();
long start = System.currentTimeMillis();
for (int i = 0; i < RGB.size(); ++i) {
   MeanFilterP parallel_mean = new MeanFilterP(RGB.get(i), 0, ((int[])
RGB.get(i)).length);
   ForkJoinPool.commonPool().invoke(parallel_mean);
   dest_windows.add(parallel_mean.getDest());
}
*This is where the program Timing ends.
```

Serial programs

```
*This is where the program Timing starts.
```

```
long start = System.currentTimeMillis();
for(int i = 0; i < RGB_windows.size(); ++i) {
   MedianFilterS Serial_Median =
newMedianFilterS(RGB_windows.get(i));
   dest_windows.add(Serial_Median.getDest());
}
System.out.println("Finished after : " + (System.currentTimeMillis()
- start) + " Milli Seconds");
*This is where the program Timing ends.
```

Optimal serial threshold

To determine the optimal serial threshold for my fork-join parallel programs, I first ran the program with the same window size by changing the threshold size and recorded the output time and then from those outputs I looked for the one with less time(Best results) then tested that threshold size if it still give best results I then use it as my Optimal Serial threshold.

Machine Architectures

Device name - DESKTOP-FC94G9U
Processor - Intel(R) Core(TM) i3-7020U CPU @ 2.30GHz 2.30 GHZ
RAM - 8.00 GB
System type - 64-bit operating system, x64-based processor

Device name - Dell Latitude 5490
Processor - 4 cores i5
RAM - 8.00 GB
System type - 64-bit operating system, x64-based processor
Clock speed - 1.8GHz

Problems Encountered

While implementing the parallel programs I had a difficulty of getting the mean and sorted array correct since I was doing everything inside a compute method and the results merged incorrectly and it ended up not filtering the image at all. Sometimes it does filter but not with the way I expected. I also had a problem with time complexity whereby the parallel mean filter was slower than the serial one which .

Results Section

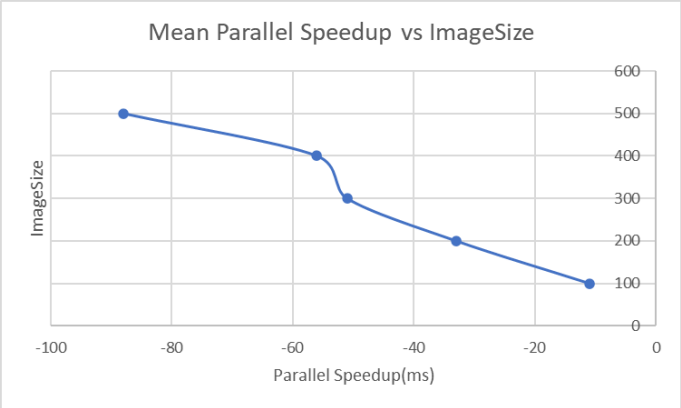
To obtain my results I implemented a java file,**Script.java** that runs each program ten times and then calculate the average time of all the outputs which I then used to find the Parallel speedup by simple calculating the difference between the sequential results and parallel results per Image size thereafter I used those results to plot my graphs. In these graphs time is measured in Milliseconds.Since I was splitting the array of windows and filter it then while taking result for image size vs parallel speedUp so those results will have the same cutoff since the window size is constant but for the window vs time speedUp the cutoff will change due to the change in splitted array size.

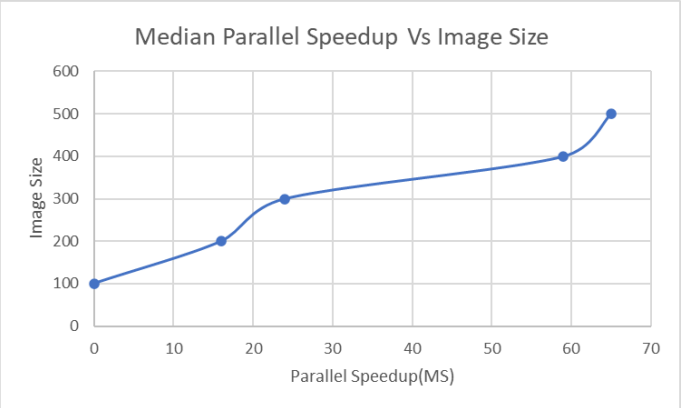
Results for the first machine architecture

Device name - DESKTOP-FC94G9U
Processor - Intel(R) Core(TM) i3-7020U CPU @ 2.30GHz 2.30 GHZ
RAM - 8.00 GB
System type - 64-bit operating system, x64-based processor

Mean and Median parallel speedup vs Image Size with a fixed window and sequential cutoff

Cutoff = 32
WindowSize = 11

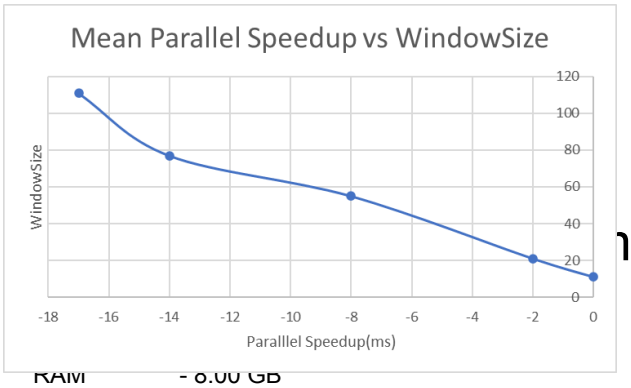


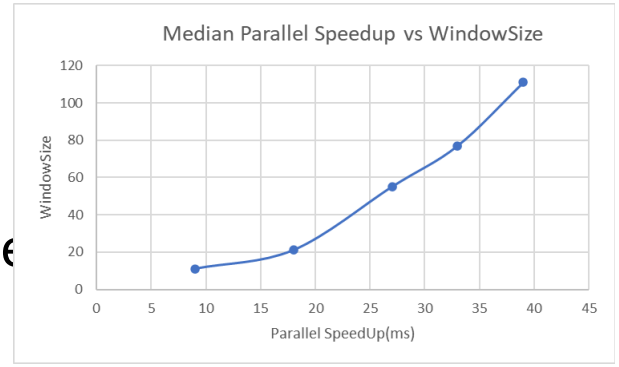


Mean and Median parallel speedup vs window size with a fixed imageSize and changing sequential cutoff

Fixed ImageSize = 512
Window and sequential cutoff for both parallel algorithms
Window = 11 and Cutoff = 30
Window = 21 and Cutoff = 95
Window = 55 and Cutoff = 1001

Window = 77 and Cutoff = 1400
Window = 111 and Cutoff = 2451

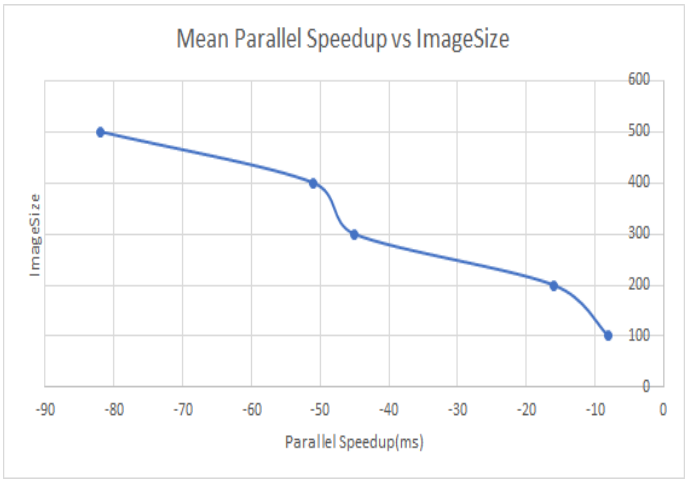


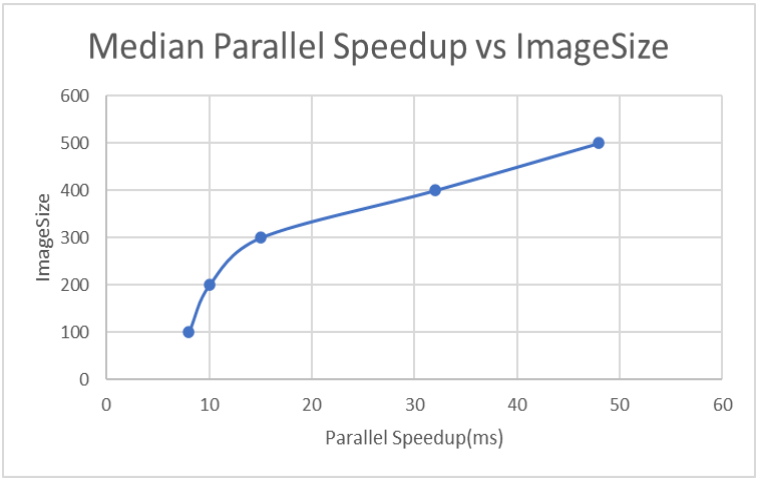


achine

RAM - 8.00 GB
System type - 64-bit operating system, x64-based processor
Clock speed - 1.8GHz

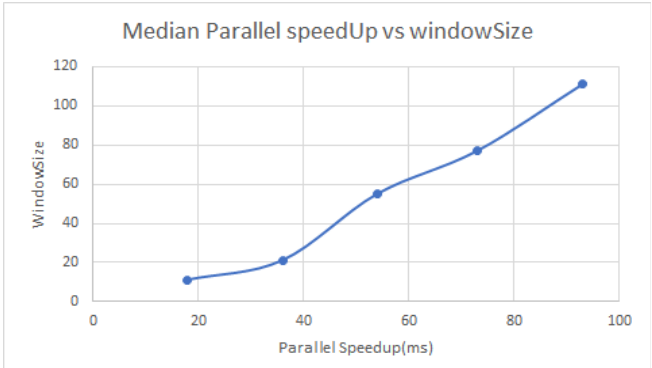
Mean and Median parallel speedup vs Image Size

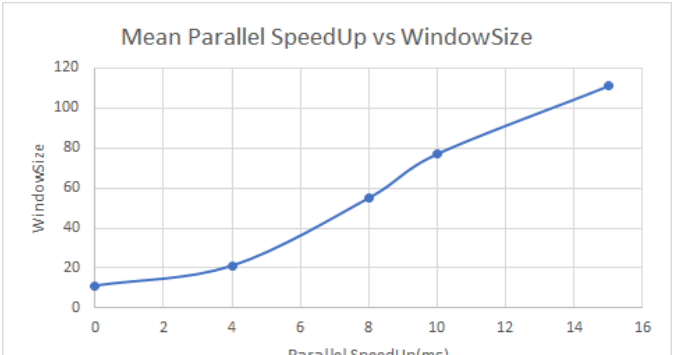




Mean and Median parallel speedup vs window size with a fixed imageSize and changing sequential cutoff

Fixed ImageSize = 512
Window and sequential cutoff for both parallel algorithms
Window = 11 and Cutoff = 35
Window = 21 and Cutoff = 99
Window = 55 and Cutoff = 995
Window = 77 and Cutoff = 1415
Window = 111 and Cutoff = 2451





Results Discussion

Looking at the results obtained in a form of a graph we can see that for the fixed window size the parallel speed value is low the functionality of the program doesn't depend on the image window and the parallel program since only split the window size not the image size it will function almost the same as the Serial program.even the second machine architecture tried to speed up the time but it also failed especial for the mean.

For the second results where the window size was changing we can see that our results were improved and looking at the range from window size greater than 21ms for the median there is a high speedup value meaning that the range where by the parallel programs were performing well.It is very clear that the second architecture improved the results even more since it is even close to the ideal speed value with the linear graph.

The maximum speedUp obtained for the first architecture for mean was 15ms and 73ms for the median. The mean maximum speed up is smaller than the median speed up because the mean lags in improving performance and the serial mean seems to be faster than the serial median. For the mean the speedup is far to away from the ideal speedup and I think this is because the serial mean is faster than the parallel mean at some point, but the median speedup is close to the ideal speedup especially for the second machine architecture.

My results are reliable because I ran the program many times and collected the averages of all the outputs for accuracy. All the results were calculated and stored on the results file.For the mean speedup there are some anomalies, the reason being that the first machine architecture has less number of cores thus less number of threads to take care of the tasks and also it is easier to sum up the array rather than to sort it .

Conclusion Section

Conclusion

From the main aim of the assignment we can finally conclude that at some cases parallelism maybe be useful and helping for doing the work faster my Serial mean was faster than the Parallel mean at some cases meaning that sometimes it easier to do the work in serial rather than doing it in parallel like summing the array with small amount of data. It is also very important to know the type of machine architecture you are using,what is it capable of, like the amount handled due to the number of threads it produces before implementing the parallel program as a solution to a problem.