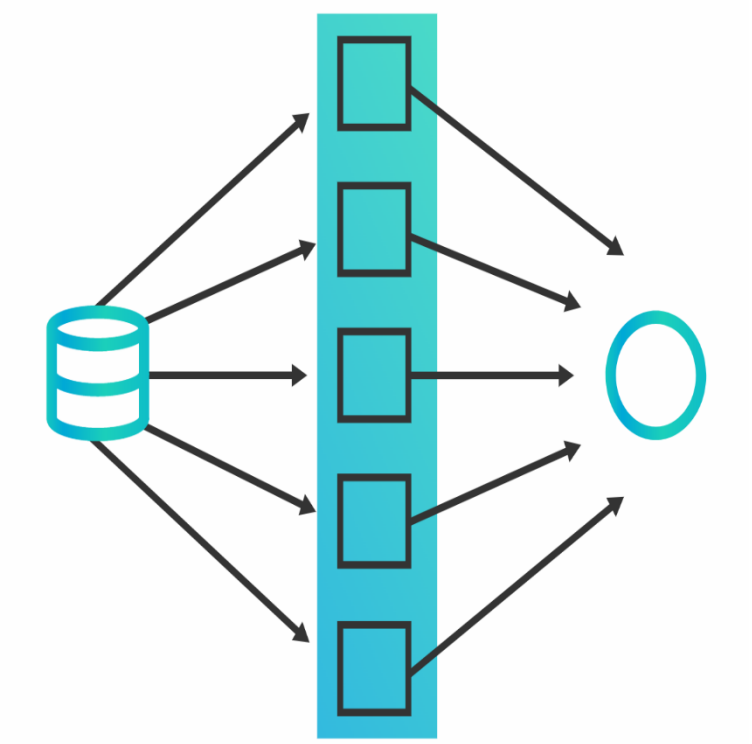


CSC2002S

Assignment 1

Report

Parallel Programming with the Java Fork/Join framework:
2D Median Filter for Image Smoothing

By Yusuf Kathrada

KTHYUS001

August 2022

Methods

Parallelization Algorithm

The program utilizes a constructor to set the values of the starting pixel and the width of the image that will be filtered. A Compute() method is used to determine if the width of the image is enough to make the sequential cut off. If the width is less than the threshold it will invoke the ComputeDirectly() method which iterates through the image using a system of four for loops. The outer two for loops traverses the columns and rows of the image whilst the inner-most for loops control the sliding window often referred to as the kernel. The kernel retrieves the contents of each pixel including the RGB values which are stored as integer values. An array stores the RGB values of the pixels which will be then used to manipulate the image depending on the filter.

In the case of the Mean filter, instead of an array being used, a cumulative total of each individual RGB value is stored and the selected pixel is set to the value of the average RGB value of the surrounding pixels.

For the Median filter, an array stores the RGB values of each pixel. The pixel in each iteration is then assigned the RGB value of the middle pixel once the surrounding pixels were sorted in ascending order.

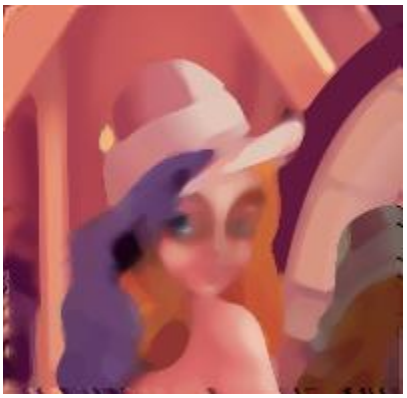
The parallelization of the algorithm is implemented through the use of 2 threads which splits the image in half along its width. Each thread then has the work of filtering half the image. In theory, these simultaneous processes allow the parallel version of the filters to achieve faster execution times as the work is spread over multiple cores instead of one.

A left and a right thread are made and are initialized such that the end point for the left thread will be the beginning of the right thread. The algorithm utilizes a ForkJoinPool and a join method in order for one thread to wait for the completion of the other before finally outputting the image to a file.

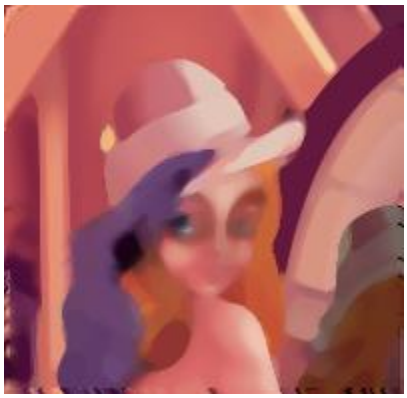
Validity Of Algorithm

In order to justify the use of parallel algorithms, they have to display correct and efficient results. To demonstrate the correctness, the parallel algorithm was compared to its serial counterpart in order to validate the results across a range of inputs. This was achieved via two methods of ensuring its validity. The first method was simply through visual analysis. The same image file was used as input as well as the same kernel size staying constant at 7x7. The serial and parallel filters were placed side-by-side in order to spot any mismatched areas. Although this method does not involve the comparing of any metrics, it is still effective in this scenario due to the nature of this task and its central focus on manipulating images. The images below shows the comparison between the two outputs.

(Original Photo)

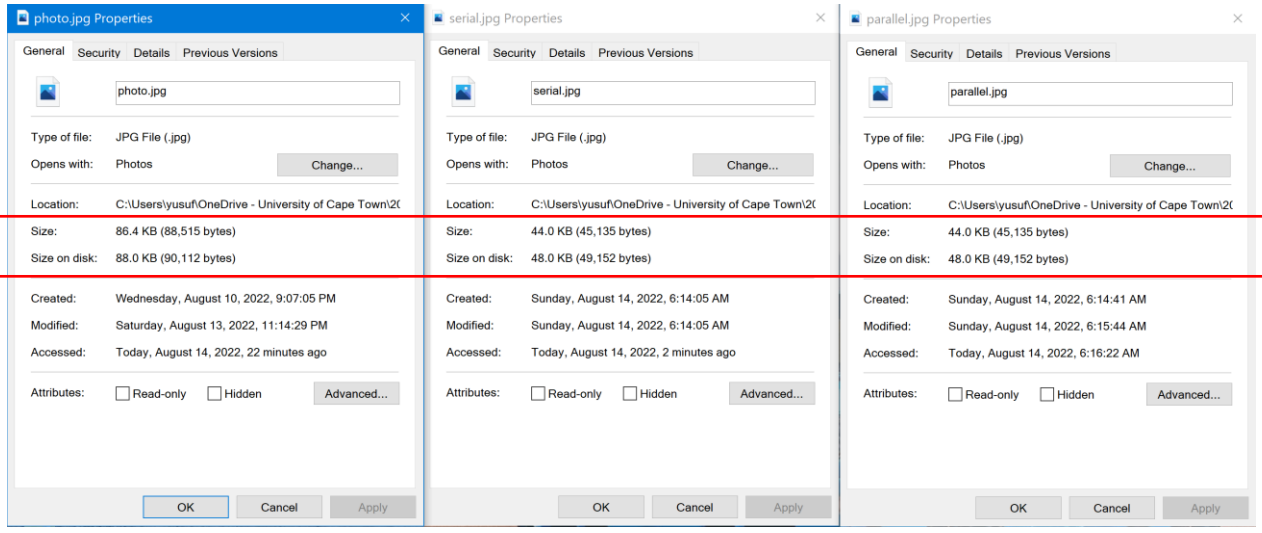


(Serial Filter applied)



(Parallel Filter applied)

The second method used was analyzing the properties of the two output files and in particular the size of each file. Even though the process to filter the image was different, the serial and parallel version should both produce the same output. Thus, the file size of both outputs should be the same and it was found that it was indeed the case. The images below verify that both the serial and parallel versions of the program produced files of the same size.



Timing

The System.currentTimeMillis method was used to accurately measure the time taken for the program to execute. This was done by creating a currentTimeMillis object both before and after the core work of the filters in order to get a true sense of the difference in execution time between the serial and parallel methods without other processes tainting the results. The difference between the two objects then gave the time taken to complete the program.

Note, all other applications and processes on each computer that were not essential for the running of the operating systems were terminated before running the tests in order to achieve the most precise results. Timing was done at least 5 times and the smallest value of each set was then recorded.

Optimal Serial Threshold

Although parallel algorithms have the ability to perform better quicker than the serial version due to the use of multiples cores, they also come with a cost of being intricate to work with as well as needing a lot more work in order to properly execute a program as it was designed. Therefore, there is a sequential cut off which is a minimum amount of work needed to be done in order to justify the of use multiple threads. Anything less would simply be done sequentially as the expected improved speedup will either be minimal or non-existent due to the lack of work needed to be done by each thread. A sequential cutoff was found to be 1000. The process of finding this figure is explained in the Discussion/Analysis section.

Machine Architectures Used

The tests were conducted on two different computer architectures. This will allow to see the change in performance of the programs when more or less cores are available.

Machine Architecture 1 (**2 Core Machine**):

Operating System: Windows 10 pro

CPU: Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz 2.90 GHz

Cores: 2

Machine Architecture 2 (**4 Core Machine):**

Operating System: Ubuntu 20.04.2 LTS

CPU: CPU Version: Intel(R) Xeon(R) CPU E5620 @ 2.40GHz

Cores: 4

Problems/Difficulties Encountered

Initially controlling race conditions were an issue to be dealt with as the program would successfully split the image in two and filter each side however, it would output the image immediately of each process instead of waiting for both to be complete. This led to the output being half filtered on the side which was last processed as the file would be updated. This was fixed by having a join which waited for the threads to be complete before setting the image and outputting the file.

Results

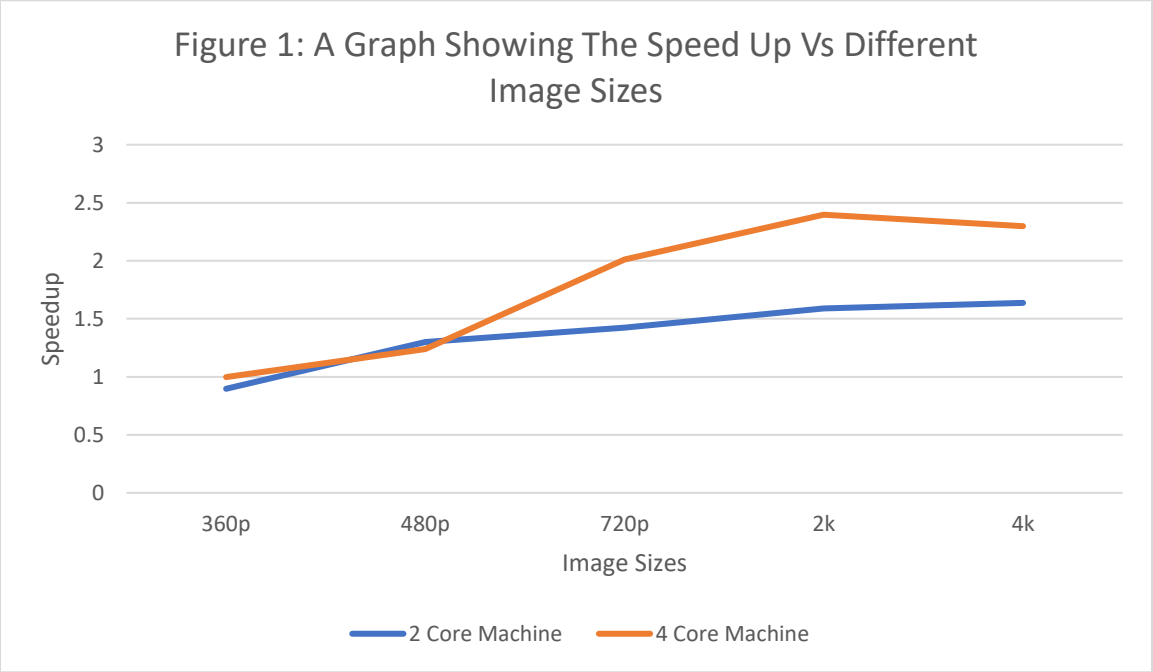


Figure 1: A Graph Showing The Speed Up Vs Different Image Sizes
Speedup
3
2.5
2
1.5
1
0.5
0
360p
480p
720p
2k
4k
Image Sizes
2 Core Machine
4 Core Machine

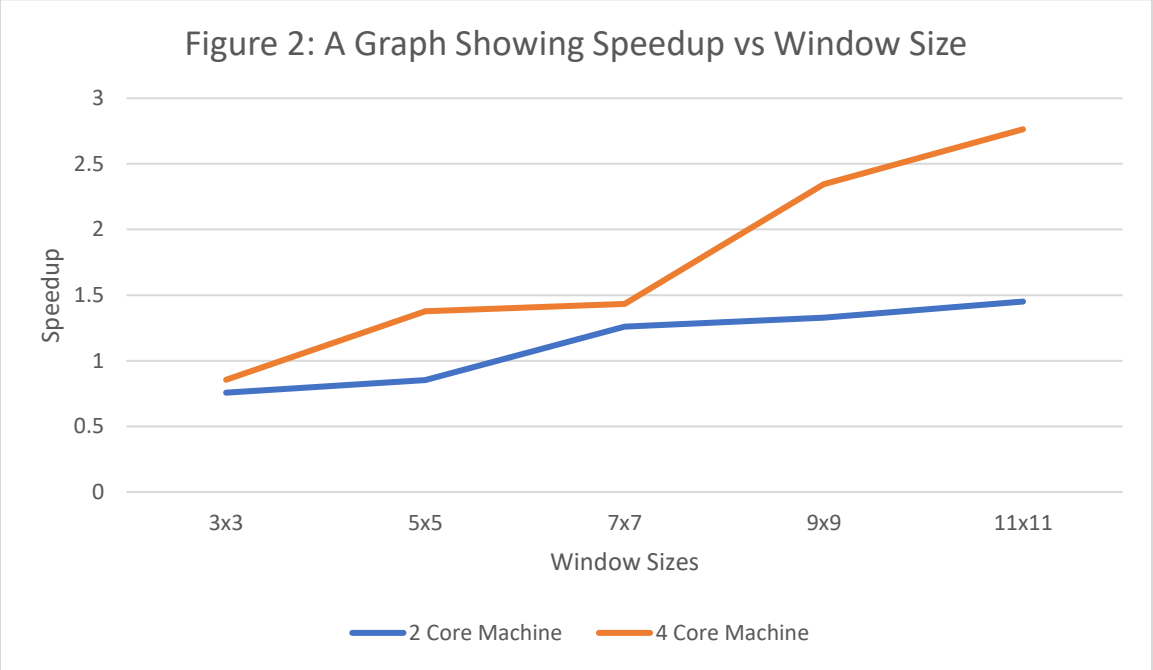


Figure 2: A Graph Showing Speedup vs Window Size
Speedup
3
2.5
2
1.5
1
0.5
0
3x3
5x5
7x7
9x9
11x11
Window Sizes
2 Core Machine
4 Core Machine

Discussion/Analysis

Looking a figure 1, the speedup displayed is slight and gradual up until 720p which is an image with the dimensions of 1280x720. Thus, through testing, an optimal sequential cutoff was found to be 1000 pixel wide. Any image inputted less would compute directly and will not be parallelized as the benefits of multiple threads would not be justified as a serial algorithm would produce a similar time to execute.

This leads to the next question which is for what range of data set sizes or in this scenario the filter size does the parallel algorithm perform better. As Figure 2 shows, for both the 2 and 4 core machine the smaller image sizes has a smaller speedup value but as the window sizes increases, the speed up increases proportionally. This would be due to the fact that more work needs to be done in order to execute the task and thus the use of multiple threads produces better results as the work is split.

The maximum speed up possible is limited by the sequential portion of the code. According to Amdahl's law maximum speedup = 1/((1-P)+P/S) where P is the percentage of code which is parallelizable and S is the speedup factor. An estimation of 90% of the code in both Mean and Median Parallel programs are parallelizable. Thus, for a 2-core machine the maximum speedup is: 1/((1-0.9)+0.9/2) = 1.81. For a 4-core machine a maximum speedup of: 1/((1-0.9)+0.9/4) = 3.08 is possible. The Mean and Median programs also differ slightly due to the Median program having more expensive processes needing to be executed. The speedups displayed are within the maximum speedups but never actually reach the theoretical maximums but as the graphs suggest, they tend to produce outputs which fall within the expected range of speedups.

The reliability of measurements in this experiment were very much subject to the load of the CPU at the time of running the programs. A few anomalies presented themselves, outputs would take unexpectedly long to run and after assessing the performance of the CPU it was found that background processes were causing slower execution times. This was resolved by giving the machine to finish background processes and the tests were rerun.

Conclusions

The window sizes and the image sizes had a direct impact on the speedup of both the Median and Mean programs. The larger the window size the larger the speedup similarly, the larger the image size the greater the speedup. The impact of having more cores, in this case a 2-core machine versus a 4-core machine had the effect of greater speedups with an increasing positive trend as the work was able to be appropriately split amongst the higher core machines.

The sequential cutoff is significant as it will determine firstly, whether multithreading will be used and secondly, how many times the image gets split. Through analysis, it was found that the parallel algorithm produces significantly greater results with speedup as the metric of measurement when more work is needed to be executed.

Through this experiment, it was found that with the expected benefit of improved performance when using parallel algorithms, they are also accompanied by a hefty setup and management cost as they have to be correct and more efficient in order to justify their existence. This includes irradiating elusive race conditions.

With regard to whether it was worth using parallelization to tackle this problem, using the data recorded as evidence, I believe the answer is dependent on the scale to which the program will work. If the program is needed to filter large-scale high-resolution images perhaps in the film or digital art industry, then utilizing parallelization would be not only beneficial but justified. However, if the program just needs to filter common medium to high resolution images then a serial filter would be the more practical decision as the time saved when using the parallel version would be negligible.