

Media Filter

USING SCALA CONCURRENCY

Contents

Implementation	1
Brain Storming	1
Algorithm	2
Parallel & Serial approaches.....	2
Performance	2
Serial vs Parallel.....	2
Single Case.....	2
Trend	4

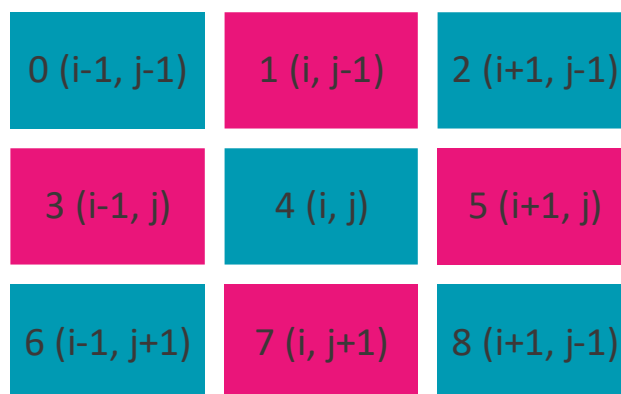
Implementation

BRAIN STORMING

The idea behind the median filter is to reduce noise from an image by finding the median of sequence and replacing a target by the median of said sequence.

In our case we have a two-dimensional grid that we can divide into small pixel groups of size 9. Since, each pixel has a mixture of red, blue, green (RGB) that we can separate into 3 separate groups to determine the median color code for each group (red, green, blue).

Pixel Array Position Diagram



For each pixel we will separate its RGB values save them in into a group of 9 and find the median for each color, placing in the target pixel the median of the RGBs.

ALGORITHM

First determining and saving the length and width of a given image, the program will iterate to over each neighbor of a target pixel saving them and itself into an array. Then, the program will separate each pixel in the array of pixels into three arrays that are representative of each color red, green, blue (RGB). Afterwards, each of the three arrays will be sorted to determine the median value. Finally, the median will be placed in the spot of the target pixel and this process will be repeated for every pixel except for the borders (to avoid index out of bound the program starts and ends 1 position before off).

PARALLEL & SERIAL APPROACHES

The serial implementation will follow the same process described in the algorithm section. However, the parallel implementation will have a few adjustments, to process the image in a parallel fashion. In Scala the “.par” collection can be used to create concurrency. Additionally, instead of using the “sort” method the parallel implementation uses “parallelSort”. Finally, since all iterations will run in parallel, all necessary arrays must be defined within the scope of the first loop. That way, simultaneous procedures can run independently, and the entire image will be processed correctly.


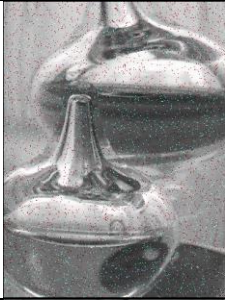

Performance

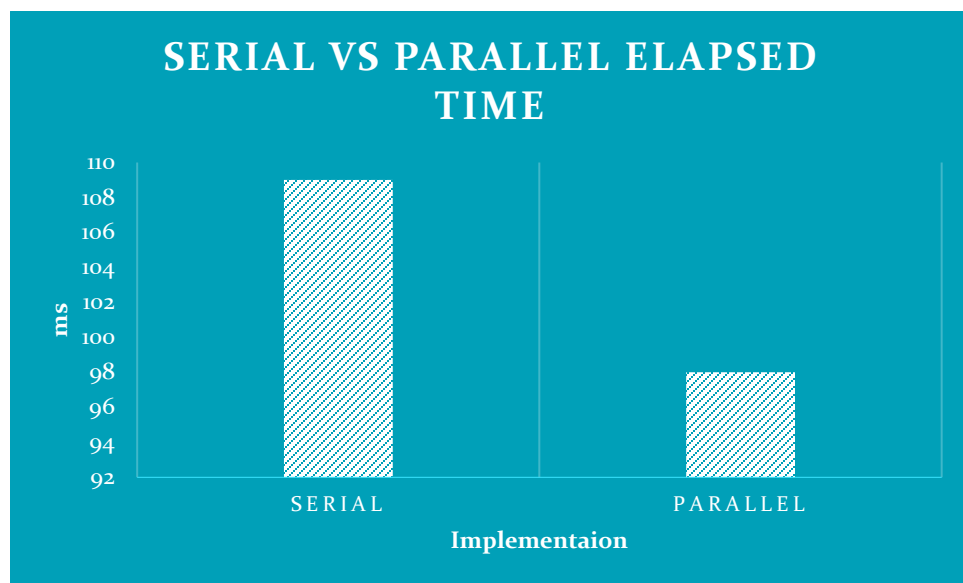
SERIAL VS PARALLEL

In most cases, parallel processing is faster than serial, unless the process being run is very small. Thus, the larger the processing load the greater the efficiency of parallel processing scales. Nevertheless, it can be substantially harder to incorporate parallel processes when implementing due to the learning curve and possible lack of intuitiveness. Finally, it is important to identify the use case and determine if the pros of performance are worth the cons of implementation difficulty, especially if the performance gain is marginal.

SINGLE CASE

Name	Image
------	-------

3		
Serial_filter_3		
Parallel_filter_3		



We can see that the performance was substantially better for the parallel approach. Additionally, the image clarity is also significantly better, which may be a side effect of the implementation differences.

TREND

In general, we can expect parallel implementation to be faster than serial. This trend is further backed by the table below of all the runs completed.

<i>Image</i>	<i>Series (ms)</i>	<i>Parallel (ms)</i>	<i>Difference (ms)</i>
1	68	65	3
2	129	81	48
3	109	98	11
4	68	57	11
5	59	42	17
6	70	63	7
7	70	50	20
8	195	106	89

