Assignment 2 Report

Name: Ibrahim Abdi
Utorid: abdiibra

Implementation

apply2D: This function applies the Laplacian filter to a given pixel and returns the new value.

Apply_filter2d: This function sequentially goes through the target image and changes each pixel value to the return value of apply2D while keeping track of the minimum and maximum new pixel value. Then it normalizes each pixel sequentially using the minimum and maximum pixel value.

sharding _row_work: This thread function is used for applying the sharding row method. Each thread is given a set of rows to apply the filter to which is calculated using the height of the image given. Each thread goes through its task in the target image, traverse across each row and changes each pixel value to the return value of apply2D while keeping track of the minimum and maximum new pixel value locally. After a thread finishes its task, it updates a global min and max with its local min and max values and waits for the rest of the threads to do so (barrier). After that, each thread will go through its tasks and normalize each pixel using the global minimum and maximum pixel value.

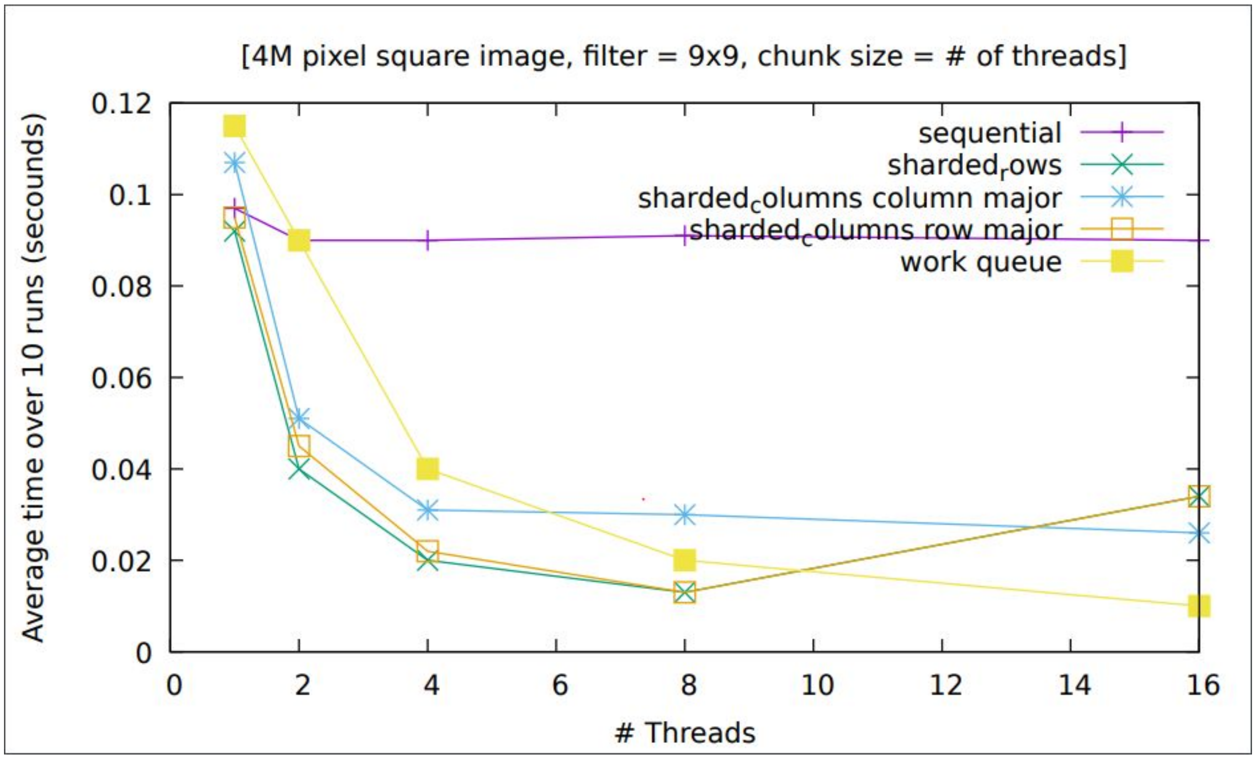
Sharded_columns_column_major_work: This thread function is used for applying the sharding columns column-major method. Each thread is given a set of columns to apply the filter to which is calculated using the width of the image given. Each thread goes through its task in the target image, traverses down each column and changes each pixel value to the return value of apply2D while keeping track of the minimum and maximum new pixel value locally. After a thread finishes its task, it updates a global min and max with its local min and max values and waits for the rest of the threads to do so (barrier). After that, each thread will go through its tasks and normalize each pixel using the global minimum and maximum pixel value.

Sharded_columns_row_major_work: This thread function is used for applying the sharding columns row-major method. Each thread is given a start and end column to apply the filter to which is calculated using the width of the image given. Each thread goes through its task in the target image, traverses across from the first to last column and changes each pixel value to the return value of apply2D while keeping track of the minimum and maximum new pixel value locally. After a thread finishes its task, it updates a global min and max with its local min and max values and waits for the rest of the threads to do so (barrier). After that, each thread will go through its tasks and normalize each pixel using the global minimum and maximum pixel value.

Work_queue_work: This thread function is used for applying the work queue method. The number of chunks is calculated mathematically and stored in a global variable as well as the current chunks finished. Each when each thread is ready to grab a new chunk, the thread increases the current chunks finished variable (using locks). The next chunk location, as well as its boundaries, are also calculated mathematically. Each chunk applies the filter using the sharded row method while keeping track of the minimum and maximum new pixel value locally. When the current chunks finished variable is equal to the number of total chunks, each thread updates a global min and max with its local min and max values and waits for the rest of the threads to do so (barrier). After that, each thread will go back to repeat the previous steps but normalize each pixel with the global min and max instead.

Experiment 1. Increasing threads.

(Figure A)



This experiment was conducted on a lab machine with the hardcoded large image, the given 9x9 filter and a chunk size equal to the number of threads running. This experiment ran the main program for each method while doubling the thread count each time. For each thread, the main program was run 10 times and plotted the average.

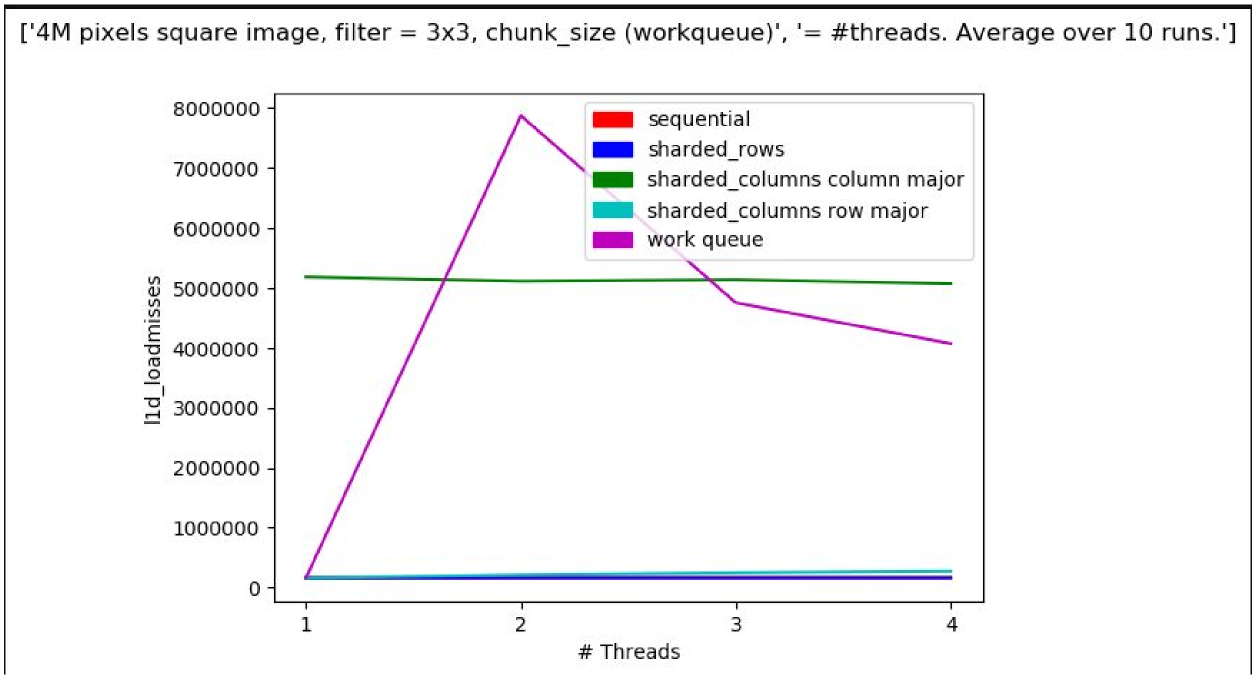
Findings:

Sequential: Except for a hiccup at the beginning, the number of threads has no effect with the implementation of this method so the constant time result is expected.

Work queue: Due to the implementation of the work queue, in order to get a new chunk, threads must pass through a lock to claim the next chunk. This means the more chunks in the queue, more time threads are spent waiting. Because the chunk size is equal to the number of threads, it was expected that the performance increases as the number of threads increases.

<u>Sharded columns column-major, row-major, and sharded rows</u>: When considering the threads up to 8 threads, the sharded row method was the fasted followed by row-major and column-major. This result was also expected due to the implementation of these methods. The sharded row method allows the thread(s) to apply the filter to the matrix sequentially allowing for less l1 cache misses. Less l1 cache misses means less time spent on finding the correct memory (due to cache hierarchies). For the column-major method, the thread(s) apply the filter to each pixel in a column for a given amount of columns. For this image, this distance between each pixel in a column is greater than the lab machine's cache line, the column-major method will have a very high l1 cache miss making it the slowest method of the three. The row-major method is a combination of both column-major and sharded rows so it is expected to have its performance less than sharded row and greater than column-major. These cache misses can be shown with the graph below.

Figure B.

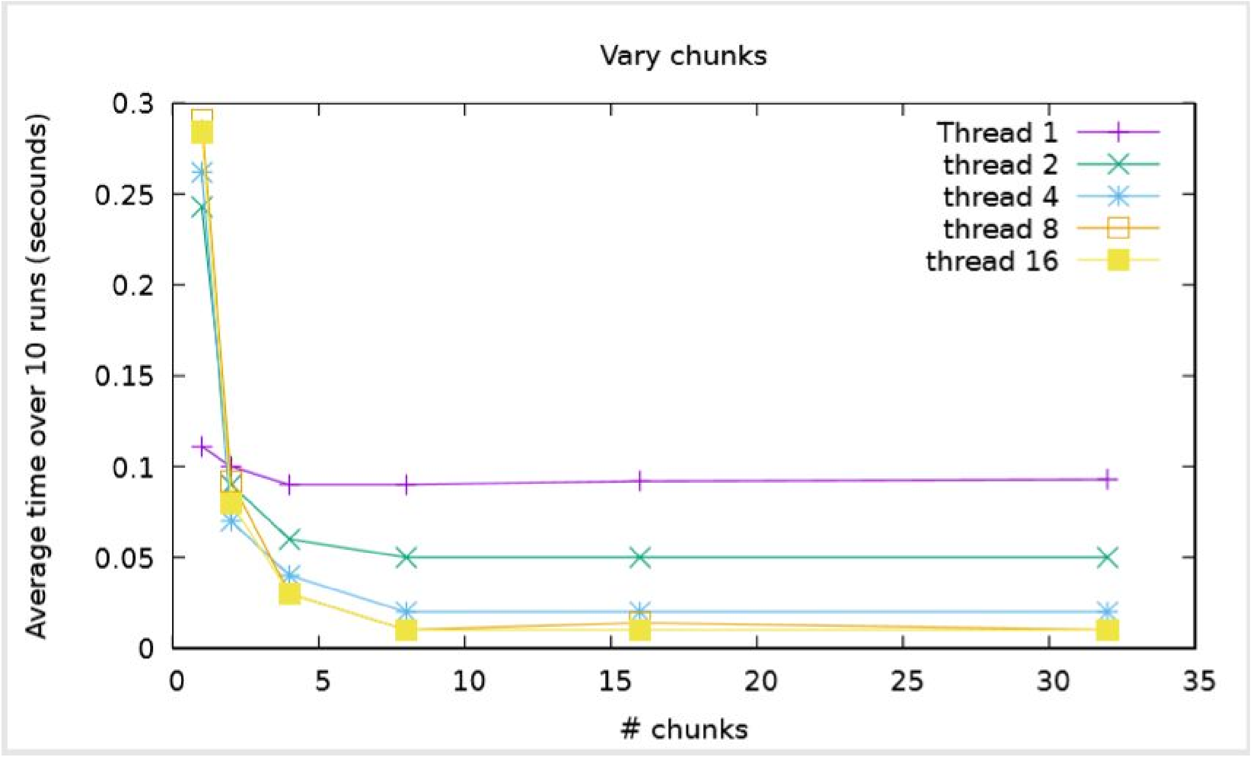


Conclusion:

Keeping in mind of cache hierarchies and how memory is accessed can make a significant difference in performance.

Experiment 2. Increasing chunk size

Figure C.



The experiment was conducted on the lab machine using the hardcoded large image provided as well as the 9x9 filter. This experiment ran the work queue method with different threads and chunk sizes. The experiment took the average of ten runs for every thread at a certain chunk size.

Findings:

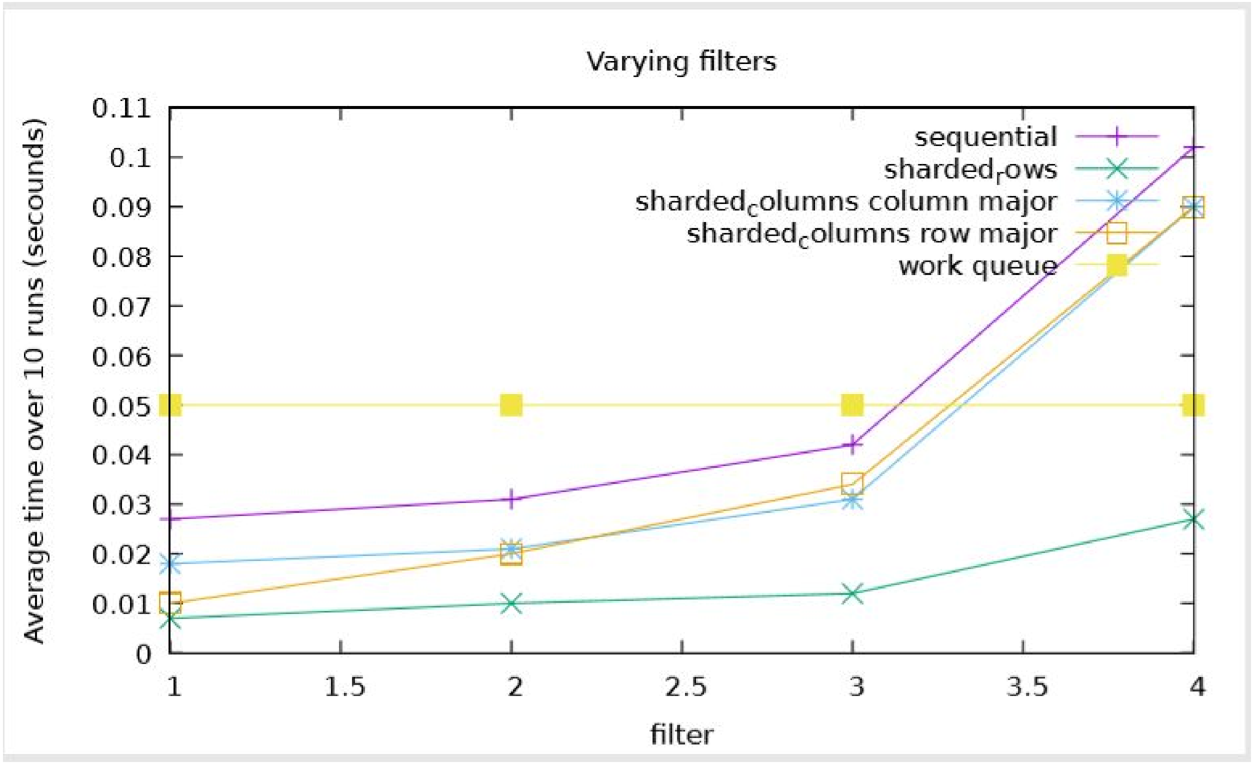
It is clear that having more threads run in the work queue in parallel, the faster the program will run. This is clearly shown in the graph (Figure C) at each chunk size). Also shown in figure B, as the chunk size increases the I1 cache miss decreases as well which explains the time decrease as the chunk size increases.

Conclusion:

Chuck size is a factor when considering performance in the work queue method.

Experiment 3. Varying filters

Figure D.



(filter 1 = 1x1 , filter 2 = 3x3, filter 3 = 5x5, filter 9x9)

The experiment was conducted on the lab machine using the hardcoded large image provided, set the thread and chunk size to 8 (number of cores on the lab machine). This experiment ran each method with different filter sizes. The experiment took the average of ten runs for each method at a given filter size.

Findings:

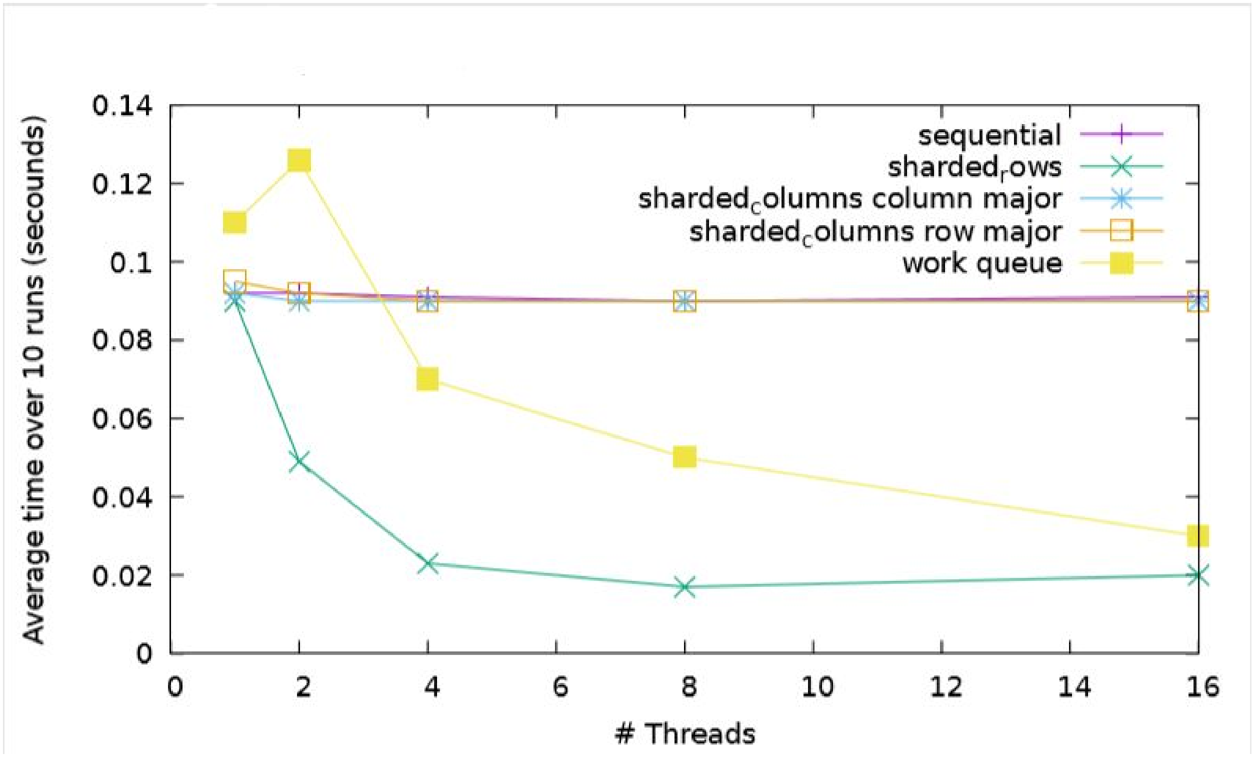
By the implementation of the methods, each method calls apply2d for every filter. As the filter size grows, so does the operations needed to calculate the new pixel value hence increasing the time needed to apply the filter to the given image. This is shown clearly in the graph above (except for the work queue). The method’s performance compared to each other was similar to experiment 1 which is also expected.

Conclusions:

Filter size is a factor to consider for performance. Work queue seems to be displaying odd results and would need further investigation.

Experiment 4. Different Images

Figure E. Experiment 1 ran with the hardcoded tall image



This experiment is a repeat of experiment 1 but with the hardcoded tall image used as the input image.

Findings:

The hard coded tall image given has a column size of one so due to the implementation of sharded columns column-major and row-major will only have one thread filter the image regardless of thread count. This is expected and shown that columns column-major and row-major should have the same run time as sequential. Work queue and sharding rows are not affected by the columns of the graph which shows the decrease in time compared to the other methods.

This differs from the graph in figure A. where sharded columns column-major and row-major performed much better than sequential due to the fact that the image used had an equal amount of rows and columns.

Conclusions:

There are special cases where an image can affect the performance of filtering for some methods.