# Parallelizing Image Filtering Techniques

Kevin Gregor – krg43, Jaydev Kshirsagar – jlk368

## Abstract

The process of image filtering can be inherently parallelized since each operation that happens on any given pixel is completely independent of operations on other pixels. Hence, we have a lot of scope for exploiting the parallel nature of the problem. For large image sizes, applying an image filter serially will take a significant amount of time. We explore multiple methods of performance tuning such as vectorization, memory layout adjustments, and parallelization using OpenMP. We primarily attempt to parallelize the finite-impulse-response type of filters that operate in the spatial domain. We are implementing a generic framework for the FIR filters with a 3x3 kernel. Configuring the kernel appropriately can help achieve the desired type of effect. We would certainly include Gaussian blur (a low pass type) and Sobel (a high pass type). We also plan to do a performance comparison against the existing transform-based techniques that achieve the same filtering effect.

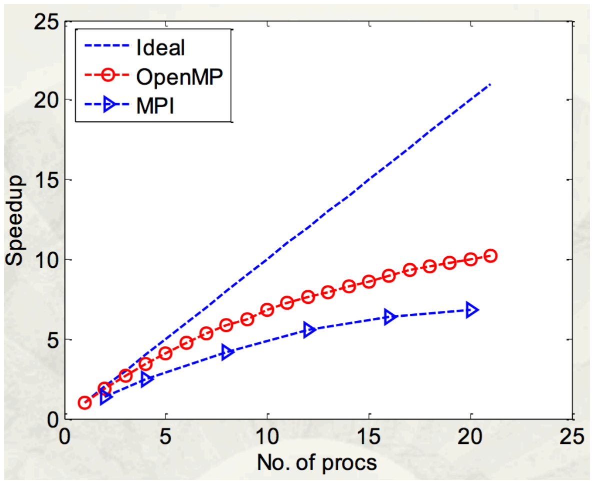
## 1. Prior Related Work

We don't have extensive experience in the area of image processing, but Jaydev has got some preliminary understanding of signal processing concepts.

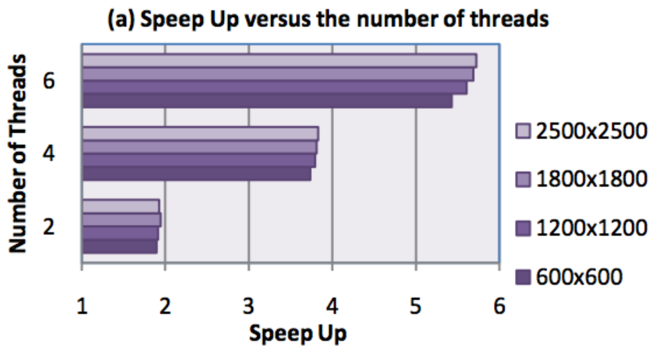
Outside of our own work, there has been a plethora of work done in the field of image processing and filtering. For more information on what exactly image filtering is, here are three great resources:

- http://www.eletel.p.lodz.pl/mstrzel/imageproc/filter.PDF
- http://downloads.micron.ox.ac.uk/lectures/micron_course_2015/Lecture_05_basic_image_analysis.pdf (pages 33-37 - FFT)
- http://www.imageprocessingplace.com/downloads_V3/root_downloads/tutorials/Image%20Processing%20Fundamentals--An%20Overview.pdf (pages 55-68 - Image Filtering)

We also found a few studies on the speedup of parallelization for image filtering. In the first study, which can be found here, the students tested parallelization using both OpenMP and MPI. Their results are in the following graph:



In the other study, found here, a similar speedup experiment is done, except with Java threads instead of OpenMP or MPI. Their speedup results are in the graph below:



As demonstrated by these studies, there is a fair amount of speedup from simple parallelization, which we will aim to replicate in our project. From our research, we did not find much about the speedup from memory layout improvements, so it will be interesting to see what results we find from our experiments.

## 2. Baseline Performance

In this section, we will analyze the performance of our baseline model. The baseline will simply be a serial implementation with the image pixels stored as a simple 2-D array. However we are making sure that the serial algorithm is an optimized one. We plan to apply the filters mentioned in our abstract and record how long it takes to apply each filter for a range of standard image sizes. We also will record the time it takes for greyscale vs RGB images.

## 3. Parallelization & Vectorization

In this section, we will explore parallelization and vectorization of our filter-applying code, and we will record the time it takes to do the same operations mentioned in Section 2. We target to use OpenMP to parallelize the workload of performing the filtering operation on each of the input image-pixels. We have to experiment and see if explicitly batching neighboring pixels and then having the batches processed in parallel helps better. Also the tuning the batch size will be important. We are implementing the filtering code in C++, whereas the tasks of reading image data and building a 2D-array of pixels, as well as displaying the processed output, is being done in Python.

## 4. Memory Layout Adjustments

Just as in Section 3, we plan to make modifications to our baseline and analyze the results. We will make modifications on the way that the image pixels are laid out in memory. In order to do this, we will experiment with similar techniques to those used in the Matrix Multiplication project, like blocking and tiling. We will have three memory layouts. One is a simple strided layout which is certainly not optimal, but it helps to cross check correctness of other techniques. The second one is a block-linear layout with block-sizes of 8x8, 16x16 and 32x32. And another one is a swizzled or twiddled layout based on the Z-Morton. The second and the third layouts attempt to maximize cache hits, taking into consideration the memory access pattern that is associated with filtering operations.

## 5. Combined Model

We plan to combine the methods from sections 3 and 4 in order to attempt creating a model that performs better. Our analysis will be on similar lines as that in sections 3 and 4 – we will measure the speedup for a range of input image sizes.

## 6. Comparison with Transform-based techniques

In addition to testing the impact of parallelization and memory layout improvements, we also want to consider the performance of different filtering algorithms that achieve the same effect. We will time the application of various filters using a simple convolution algorithm vs FFT algorithms, as well as any other algorithms that we feel would be interesting to test. However we plan to do this analysis towards the end and the amount of analysis done will also depend on the time in hand.

## 7. Conclusion

In our conclusion, we will revisit all of the experiments we did and analyze the effectiveness of all of them. In addition, we will also consider possible extensions and areas for improvement. The plots that we would include would be of the following type: running time vs the input image size, for different values of the block-sizes or tile-sizes.

## Progress Report

What we have done: We have completed the simple serial model of image filtering.

What is left to do: We have to tune our code with parallelization, vectorization, and memory layout changes for Sections 3-5. We also have to do a performance comparison against the transform-based serial techniques.

How long it will take to complete: By working over the break, we will be able to complete the project by the next class on Tuesday 11/28.