GPU Computing – Project: An Implementation of Image Processing Library

ANKITA DAGAR, MT18068; AKANKSHA MALHOTRA, MT18062.

The task for the project is to build Image Processing Library on the GPU. We have developed a library which provides basic primitive functions on images using parallel programming which can add acceleration to imaging applications. We have also done a comparison with corresponding naive CPU code for performance.

Concepts: • **Image Primitive Functions** → Digital Image Processing; • **GPU** → Parallel Programming.

1 INTRODUCTION

Digital Image Processing's motive is to improve the quality of an image. Images are used in number of application domains starting from medical, industrial to defense. Digital Image Processing requires processing of large amounts of data which increases the computational power and processing time to a great extent. The image operations are generally highly iterative and often use identical operations on all pixels. A simple operation on grey image with 1000 x 1000 pixels will require millions of operation. However, parallel processing of the image data can significantly reduce the processing time. The programmable Graphic Processor Unit or GPU is useful for such extremely parallel data workloads, where similar calculations are executed on quantities of data that are arranged in a regular grid-like fashion, so it is one of ideal solutions of large size images. Since image operations are iterative in nature, which makes it ideal application for parallel computing. GPUs has matured into a multithreaded, highly parallel, manycore processor with massive computational power and very high memory bandwidth. The highest performing parallel computing processor, Quadro GV100 has 14.8 teraflops of single precision. The potential application of parallelism to reduce the computation cost and increase the performance of various image processing algorithms is the motivation behind building a GPU -accelerated image processing library. We have compared our parallel implementation with the OpenCV library available for Image operations.

2 LITERATURE SURVEY

The rapidly growing technologies based on image processing calls for the need to get certain preprocessing operations faster than before. There can be over 2 million pixels in a single high definition image. Many image processing algorithms demand large number of floating point computations per pixel which cannot be performed with less time even on the fastest of CPUs. This hampers productivity. CUDA offers solution to this using parallel computing. Each pixel is responsible for pixel output.
In our implementation , we have covered the following image operations:

LOGICAL:

- Not- Inverts the bits of each pixel.
- And- Bitwise AND operation between two images.
- Or- Bitwise OR operation between two images.
- Xor- Logical bitwise XOR operation between two images.

GEOMETRY:

- Flip Horizontal- Mirrors an image about a horizontal axis.
- Flip Vertical- Mirrors an image about a vertical axis.
- Rotate Anti-Clockwise- Rotates an image in anti-clockwise direction.
- Rotate Clockwise- Rotates an image in clockwise direction.

- Crop- Crops the desired portion of an image.

POINT OPERATIONS:

- Invert- Inverting the pixel values in the image similar to be found in a film negative.
- Brightness- To change the brightness of the image by passing positive value to increase brightness and a negative value to decrease brightness.
- Contrast – The one of best algorithm for contrast is Histogram equalization.

LOCAL OPERATIONS:

Before we describe local operations(filters), we define what convolution is which is used almost in the implementation of all the filters. Convolution is a neighborhood operation in which each output pixel is the weighted sum of neighboring input pixels. The matrix of weights is called the convolution kernel, also known as the kernel

- LINEAR FILTERS:
  Linear filters comprises of output pixel as a linear function of input pixels in the input pixel's neighborhood.
  1. Mean Filters: A mean filter smoothens local variations in an image and noise is reduced because of reduction in blurring.
  2. Gaussian Filters: It is a type of linear filter that uses kernel that represents the shape of Gaussian distribution.
  3. Edge Detection: Edge Detection refers to finding regions of sharp change in intensity or color. Shallow change is represented by low values. Sobel operator is a common way for edge detection, which is an approximation to derivative of an image.

- NON LINEAR FILTERS:
  Non Linear or Order statistic filter are filters whose response is based upon ordering of pixels contained in the kernel or filter and then replacing the center value with the computed result.
  1. Median Filter: According to [3], it is the best known algorithm in this category. This filter replaces the value of pixel by median of neighborhood pixels defined by kernel size. They possess excellent noise removal capabilities with less blurring as in comparison to linear filter of same size. Images with Salt and Pepper noise give effective results when applied with median filter.
  2. Min Filter : Min filter modifies each pixel value with the minimum pixel value out of the neighborhood pixels encompassed in convolution kernel.
  3. Max Filter: It is a counterpart to the Min filter where we select maximum value instead of minimum.

- MORPHOLOGICAL OPERATIONS:
  Morphological operations apply structuring element[1] to an input image creating output image of the same size.
  1. Erosion : The output pixel value is minimum among all the neighborhood pixels in structuring element. It is beneficial in highlighting substantive objects.
  2. Dilation : It uses the maximum value among all the neighborhood pixels in structuring element. It basically fills the holes if any in the image
  3. Hole Filling : A hole may be defined as a background region surrounded by connected border of foreground pixels. It uses a dilation, complement and intersection for filling holes in an image.

COLOR OPERATIONS :

- Channel Split : It splits the channels of colored images into RGB components.

- RGB to Grayscale : It converts from RGB to Grayscale. It uses the formula 0.3R +0.59G+0.11B.
- Grayscale to Binary : It converts Grayscale image to Binary image. The binary image has pixels either 0 or 255.

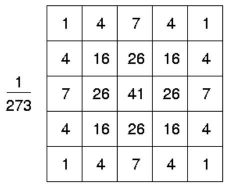
ADAPTIVE FILTERS :
Adaptive filters whose behavior changes based on statistical characteristics of image have been implemented.

- Noise Reduction Filter: The statistical measures of a random variable are its mean and variance which are closely related to the appearance of the image. The mean gives measure of average density and variance measures the contrast of the region.
- Median :It seeks to preserve detail while smoothing non-impulse noise, something that traditional median filter does not do.

3 ANALYSIS OF ALGORITHM

In sequential algorithms the time taken by almost all operations is O(height*width*imageChannels). In our parallel implementation we have created Tile_width*Tile_width number of threads and (width/ Tile_width)*(height/ Tile_width) number of blocks. And assigned constant work to each thread.

- LINEAR FILTERS:
  1. Gaussian filter: We have used integer valued kernel that approximates a gaussian with variance of 1.0. The corresponding kernel is shown as below:



$$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

Fig 1. Gaussian kernel

  2. Mean Filter : The arithmetic mean filter computes the average value of pixels in the neighborhood area spanned by the convolution filter and modified value of a pixel is this calculated mean .We have used a 3x3 kernel for computing mean Filter.

  3. Edge Detection : We have used a pair of size 3x3 convolution kernels, which are orthogonal to each other. The kernels identifies the edges running vertically and horizontally in relativeness to the pixel grid. Gradient component is produced separately and the final gradient is computed using $|\sqrt{(Gx^2+Gy^2)}|$.

- NON-LINEAR FILTERS:
  1. Median Filter : The median filtering of a pixel in an image is produced by first sorting the pixels covered in pixel area, finding the median and replacing that corresponding pixel with the median value .We have used Insertion sort for sorting. We have implemented median filter for kernel of size 3x3 and 5x5.

  2. Min and Max: It uses the min and max value out of the sorted pixels and replace the value of pixel at center of kernel/filter with this modified value. We have used kernel of size 3x3 for implementation.

- COLOR OPERATIONS:
  1. Channel Split : In this the three channels of a RGB image are stored in three output matrices which gives us the split channels which are a form of grayscale images each.

  2. RGB to Grayscale : In this we use the formula 0.3R+0.59G+0.11B to convert an RGB image into grayscale image. This formula means that we will take 30% of red pixel, 59% of a green pixel and 11% of a blue pixel.

  3. Grayscale to Binary : In this we give a level from 0 to 1. Then the pixel value of images are divided by 255 and then based upon that we can compare it to the given level. If the value is greater than the threshold level we make the pixel value as 255 or else we make it as 0.

- MORPHOLOGICAL:
  1. Erosion : It removes pixels on object boundaries of objects in an image. The value of the output pixel is the minimum value of all pixels in the neighborhood. In a binary image, a pixel is set to 0 if any of the neighboring pixels have the value 0.

  2. Dilation : It adds pixels to the boundaries of objects in an image. The value of the output pixel is the maximum value of all pixels in the neighborhood. In a binary image, a pixel is set to 1 if any of the neighboring pixels have the value 1.

  3. Hole-filling : For binary images, we need to give a starting point and it changes connected background pixels to foreground pixels, stopping when it reaches object boundaries.

- ADAPTIVE FILTERS:
  1. Noise Reduction : In this we calculate the mean and variance of the image. Then we take a kernel of size three and using that we calculate the mean and variance of the neighbourhood. Using these we calculate value of a variable named as alpha, which is equal to total_image_variance / neighborhood_variance. If the value of alpha is greater than 1 then it is reset as 1. And using this we calculate the output pixel value by the formula :
     outputImageData[i][j][k] = (1-alpha) * inputImageData[i][j][k] + alpha * neighborhood_mean

  2. Median Filter : This algorithm has three basic purposes- to remove salt-and-pepper noise, to provide smoothing of other noise that may not be impulsive, and to reduce distortion. We calculate it in two stages. Stage A, in which we determine if median filter output is impulsive or not. If it is not impulsive then we go to Stage B, to reduce the distortion in image.

4 PARALLELIZATION STRATEGY AND IMPLEMENTATION NOTES

For all the convolution based operations , we have used shared memory of size [Tile_Width+s][Tile_Width+s]. Each thread is responsible for copying 4 pixel values from the input image to the shared memory. The four pixels positions are (i-s,j-s), (i+s,j+s), (i+s,j-s),(i-s,j+s) where s refers to kernel radius of the convolution kernel.

They copy from image data from host to device and device to host is implemented on host and the computation part in each operation is implemented on device.

- Logical, Point and Geometry operations: Each thread in the launched block is responsible for modifying each pixel value of the image.

- Histogram Equalization: The histogram equalization is computed in sequence of kernel launches. The first kernel launch is for computing frequency plot of all pixels in range 0-255. The second kernel then works upon the result of first kernel to compute cumulative distributive function. This is calculated using reductions . The third kernel uses the computed results of CDF to generate contrast in the images.

- Local operations: Linear (Mean, Gaussian, Edge Detection), Non-Linear (Median, Min, Max) and Morphological (Erosion, Dilation) uses the concept of shared memory in which the data which is to be used again and again like the convolution kernel is kept in shared memory which makes it easy for access.

- Color Operations: Each thread in the launched block is responsible for modifying each pixel value of the image.

- Adaptive Filters: The operations which need to use some data repeatedly is kept in the shared memory like the mean and variance calculation use atomic add to calculate the total sum of the image. Also, the kernels used for convolutions are also kept in the shared memory.

5 RESULTS

We have compared sequential and parallel implementation of Image operations and have compared with OpenCV library which is one of the best ones for Image processing .
The dips observed in the OpenCV graph might be due to fine tuning and implementation of the OpenCV library.

For Linear and Non Linear operations :

- It is observed that better CPU vs GPU speed ups have been attained with increasing size of image

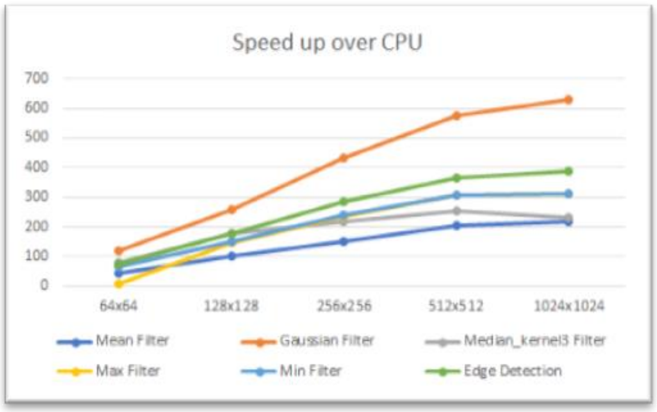


Fig1: Speed Up for linear and non-linear filters Operations over CPU

- It has been observed that for greater size image , the speed up for edge detection tends to dip. This could be due to fine tuning used in the OpenCV library.

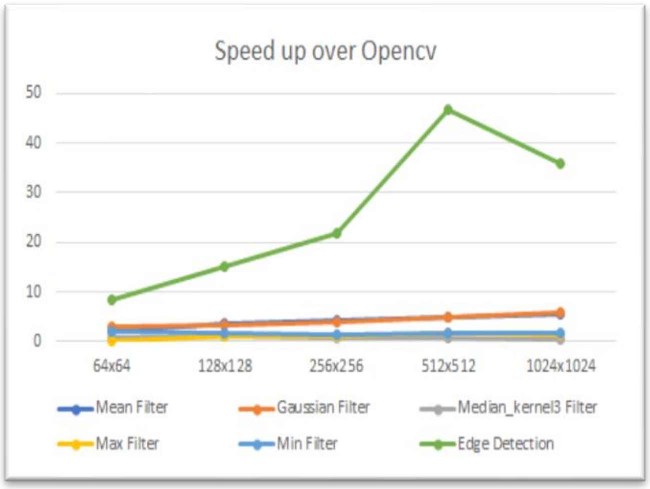


Fig2. Speed Up for linear and non-linear over OpenCV.

Color operations:

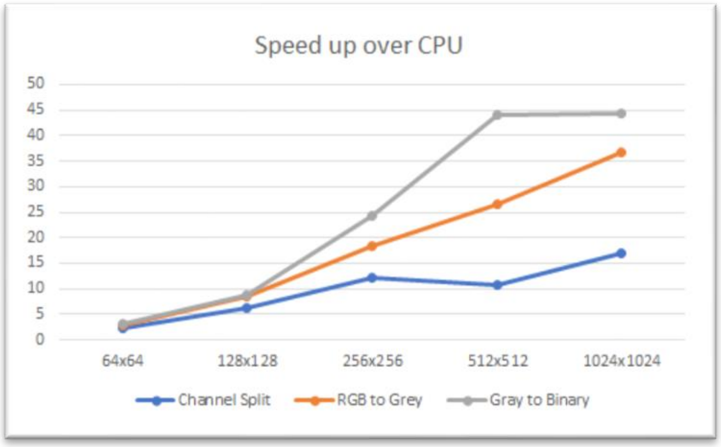


Fig 3. Speed up of color operations on CPU

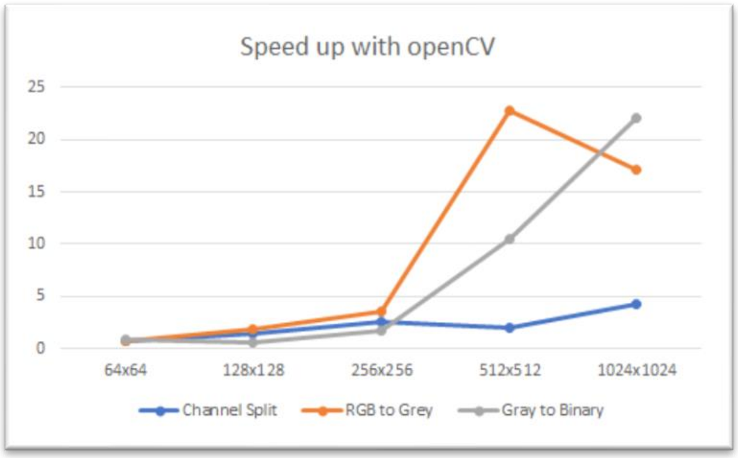


Fig 4. Speed up of color operations over openCV

Morphological Filters:

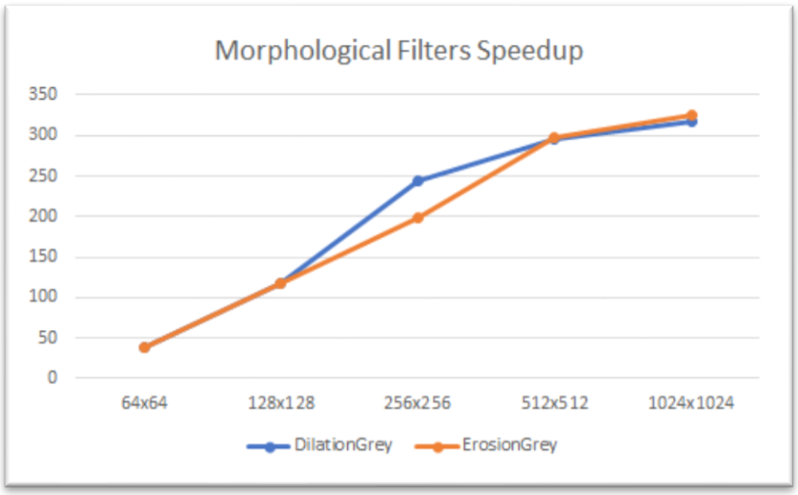


Fig 5. Speed up for Morphological Filters over CPU

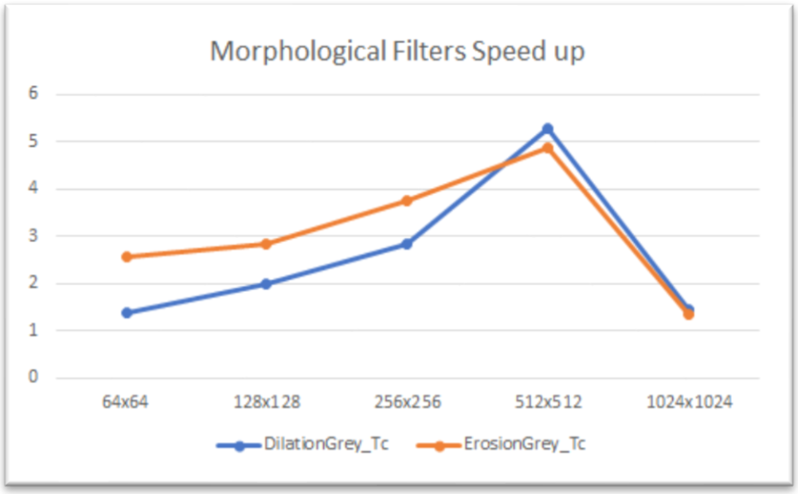


Fig 6. Speed up of morphological filters over OpenCV

Adaptive Filters: (due to hardware constraints, it is only working till 256x256 size of image on GPU)

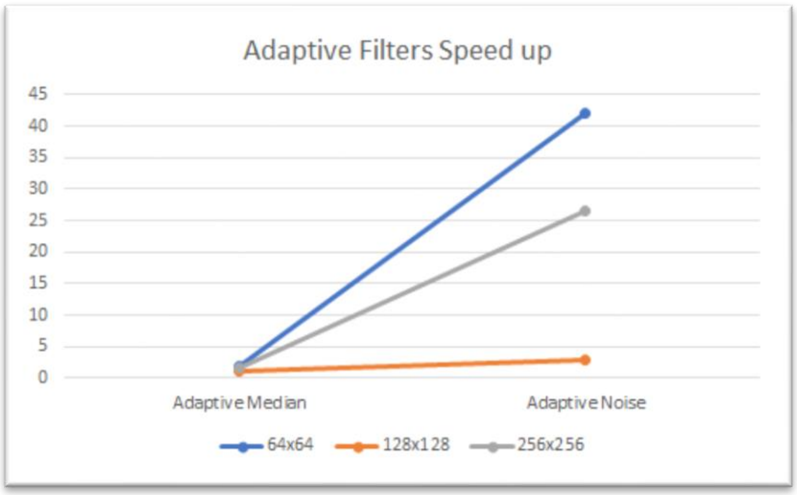


Fig 7. Speed Ups for Adaptive filters for CPU vs GPU.

6 MID SEMESTER DELIVERABLES

We had implemented the local, geometry and point operations along with gaussian and mean filter.

The speed up vs Image Size table achieved is shown in Table.1.

	64x64	128x128	256x256	512x512	1024x1024	2048x2048
Inverse	0.824282389	0.681719023	20.23613722	21.49470899	23.20746036	24.32946621
Brightness	5.319148936	14.38601322	25.68667345	35.40152016	34.21762179	33.50563556
Flip Vertical	3.460207612	9.876179245	17.63416578	23.26872822	24.3927309	26.07579519
Flip Horizontal	3.242924528	9.588068182	17.12372449	22.6148558	24.1774363	25.71064815
Anticlockwise rotation	4.024621212	10.94543147	22.73061105	41.43518519	50.5056165	82.90256715
Clockwise Rotation	3.876879699	11.13782051	21.90420561	28.62323753	51.73137461	104.7224896
And	4.765070922	11.68536325	19.33019301	24.29950639	26.51844246	28.18681862
OR	4.765070922	12.09900442	19.09090909	24.00947459	26.5144041	27.55291329
XOR	4.952830189	11.76948052	18.48088752	23.92074742	26.52906104	28.12479436
Not	3.578244275	9.876943005	15.3385947	19.36848958	21.76507538	23.68156899
Crop	0.119731801	2.232142857	6.90874036	17.36634036	23.74195989	22.02659816
Mean	42.36111111	91.28166915	118.116414	143.8175478	148.6560685	160.5446896
Gaussian	74.90335052	139.0035377	151.3317511	173.32527	176.7336633	189.3105203

Table.1 Results table

The speed up vs Image Size graph is shown in Fig.2

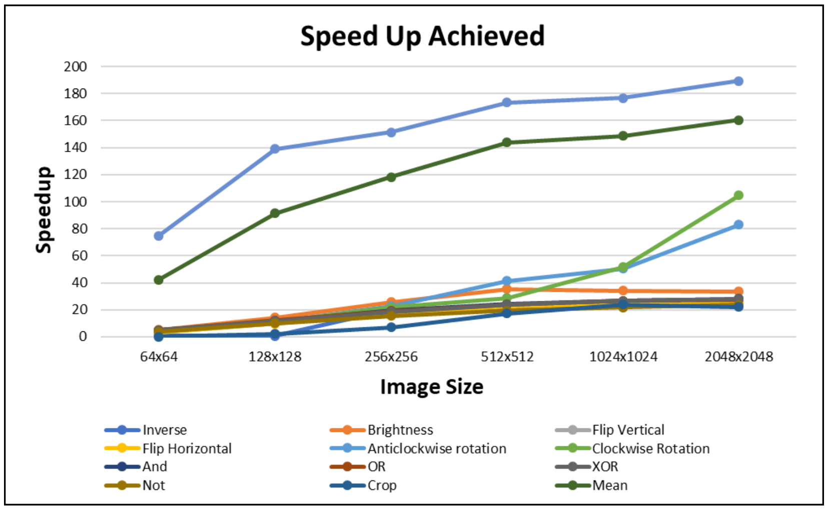


Fig.8 Speed-up graph

ACKNOWLEDGMENTS

The code for reading and writing the image was provided by Aradhya Neeraj Mathur, Teaching Assistant during the assignment.

REFERENCES

[1] https://books.google.co.in/books?hl=en&lr=&id=YumpCAAAQBAJ&oi=fnd&pg=PA1&dq=parallel+image+processing&ots=IbxY8iioXX&sig=wFjAYW35VxRq53G16mVhCYrlE0c#v=onepage&q&f=false.

[2] https://en.wikipedia.org/wiki/Volta_(microarchitecture).

[3] Gonzalez, Rafael C., Woods, Richard E., Digital Image Processing: Pearson 3rd edition.
[4] https://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm
[5] Soyata, Tolga., GPU Parallel Program Development Using CUDA: CRC Press, 2018 edition.