# Reconstructing Optic Fiber Bundle Images

Harshil Gandhi, Chaitanya Patel, Preet Amin, Ishan Mehta, Rushi Patel, Ved Joshi

DAIICT, Gandhinagar

Course: IT 506
Group Name: AC/DC

{*201801026, 201801040, 201801051, 201801061, 201801166, 201801171*}*@daiict.ac.in*

## I. Introduction

Information loss is a practical problem which is observed in day to day life. We are going to address two such types of information loss problems in image processing. One is honeycomb pattern and other is similar to pepper noise.

When Optical fibers are bundled together with an alignment such that orders of the fibers at both ends are identical, it is called a coherent bundle. Their important properties are great flexibility and compact size. Since fiber bundles have hundreds of individual fibers, it gives the advantage of having a large field of view because of large bundle diameters. It also allows for selective illumination of a subgroup of fibers. Coherent bundles are increasingly used in endoscopy, optical coherence tomography, fluorescence imaging, neural imaging, live organism imaging and other imaging applications.

Although coherent optical fiber bundles provide great flexibility, they suffer from some known significant limitations:

- low spatial resolution
- fixed honeycomb pattern

Using fiber bundles for imaging results in the unfortunate consequence that an unwanted artifact is present on the original image. This honeycomb-like artifact hides the original features of the image. To tackle this honeycomb noise, various algorithms such as Gaussian smoothing, TwIST(deconvolution and restoration), in conjunction with TV(Total-Variation) have been explored.TV method is used for various problems[1], [2] Gaussian smoothing seem to improve contrast, but it comes at the cost of blurring the image and reducing its sharpness. Approximation using interpolation works well but it has the same drawback of over-smoothing. To overcome this predicament, authors from [3] suggested two-step iterative shrinkage threshold(TwIST) algorithm with total-variation(TV).

Another way to reduce honeycomb noise is with the use of various filters from Image Processing domain [4]. To reduce the noise, fiber optic image is first passed through low-pass filter and then median filter to enhance their quality. Enhanced image is then co-registered and passed through average filter to reconstruct the image. The results show that the artifacts are removed as well as signal contrast and signal to noise ratio are increased.

Most recent approaches for image denoising and inpainting uses Deep Learning based Neural Networks [5]. GARNN(generative adversarial restoration neural network) suggested in [5] learns a direct mapping from FB(Filter Bundle) to their corresponding GT(Ground Truth). It is found that fixed pattern noise can be completely removed and hidden details are also recovered significantly when training and testing images are of the same type. Using Content loss function in a simple GAN, which leads to unwanted smoothing, modified discriminative network is used with modified loss function. However, this method uses Deep Learning so it is out of scope for this project.

Salt and pepper noise is an impulsive kind of noise which is present as black and white pixels randomly spread throughout the image. The authors of [6] put forth a variational model and a numerical algorithm for removal of this type of noise. Then using a mask, they identify the pixels that are not noisy. Finally, they apply the convex conjugate technique in such a manner that the total-variation(TV) term and data term are written as a primal dual formulation. So, the solution of the primal dual problem can be easily found using a quick primal dual algorithm.

In the paper [7], a general framework for denoising by inpainting is proposed. It averages multiple inpainting results from different selections of known data. Two concrete implementations of this framework are evaluated: The first one specifies known data on a shifted regular grid, while the second one employs probabilistic densification to optimise the known pixel locations w.r.t. the inpainting quality.

## II. Hardware Information

### A. CPU

| **Property Name** | **Value** |
|---|---|
| Architecture | x86 64 |
| CPU op-mode(s) | 32-bit, 64-bit |
| Byte Order | Little Endian |
| CPU(s) | 16 |
| On-line CPU(s) list | 0-15 |
| Thread(s) per core | 1 |
| Core(s) per socket | 8 |
| Socket(s) | 2 |
| NUMA node(s) | 2 |
| Vendor ID | GenuineIntel |
| CPU family | 6 |
| Model | 62 |
| Model name | Intel(R) Xeon(R) CPU E5-2640 v2 @ 2.00GHz |
| Stepping | 4 |
| CPU max MHz | 2500 |
| CPU min MHz | 1200 |
| BogoMIPS | 4000.02 |
| Virtualization | VT-x |
| L1d cache | 32K |
| L1i cache | 32K |
| L2 cache | 256K |
| L3 cache | 20480K |
| NUMA node0 CPU(s) | 0-7 |
| NUMA node1 CPU(s) | 8-15 |

### B. GPU

| **Property Name** | **Value** |
|---|---|
| Major revision number | 3 |
| Minor revision number | 5 |
| Name | Tesla K40c |
| Total global memory | 11996954624 |
| Total shared memory per block | 49152 |
| Total registers per block | 65536 |
| Warp size | 32 |
| Maximum memory pitch | 2147483647 |
| Maximum threads per block | 1024 |
| Maximum dimension 0 of block | 1024 |
| Maximum dimension 1 of block | 1024 |
| Maximum dimension 2 of block | 64 |
| Maximum dimension 0 of grid | 2147483647 |
| Maximum dimension 1 of grid | 65535 |
| Maximum dimension 2 of grid | 65535 |
| Clock rate | 745000 |
| Total constant memory | 65536 |
| Texture alignment | 512 |
| Concurrent copy and execution | Yes |
| Number of multiprocessors | 15 |
| Kernel execution timeout | No |

## III. Algorithms

As mentioned in the above sections, various algorithms have been developed to tackle the problem.Here we are trying to solve two problems namely i) Information loss of random pixels ii) Honeycomb pattern noise.

As, information loss of random pixels is a comparatively older problem and has various methods to solve. General methodology would be to only work on pixels which are lost , and to estimate their values on the basis of its neighbourhood.This type of problem can also be tackled using algorithms solving salt pepper noise.Less expensive algorithms such median filter can be used but its setback is blurring the known pixels.So here we have used a variation of diffusion based equations to identify the lost pixel and to estimate its values. As mentioned above honeycomb pattern noise is present due to gaps in the optic fibre bundle.There are various methods to address this problem , they can be identified as i) single image group and ii) multiple image group as mentioned in [8].Multiple image group is out of our scope.For single image group various methods are there as mentioned in [9] , we are going to implement spectral filtering as it offers least time taken.[1]

### A. *Algorithm 1*

Pixels with pixel value of 0 (i.e., black) are considered to be lost. Our method to recover these pixels is similar to the approach used in [10], in which we first identified noisy pixels of the image and then used various methods for noise removal. The loss we are dealing with is similar to salt pepper noise in the image. Our method is distributed in two steps:

1) identifying the pixels whose information is lost
2) Recovering the pixel from the known neighbour pixels

**Step 1)** We have used a novel method to identify lost pixel on the basis of diversion of gradients around the pixel. If we assume Image to be a continuous function $I\colon X \times Y \rightarrow [0, 255]$, the gradient of I at pixel

[1]https://github.com/mehtaishan205/Data-Impingement

(x, y) will be a vector. For every pixel with the pixel value of 0 will be suspected for information loss. To figure out if a suspicious pixel is responsible for information loss, we will take the following steps:

$$\nabla I(x,y) = \frac{\partial I}{\partial x}\hat{i} + \frac{\partial I}{\partial y}\hat{j}$$

After we calculate gradient, by taking scalar multiplication with $(\frac{\partial}{\partial x}, \frac{\partial}{\partial y})^T$ known as spatial nabla operator, which can be represented as $\boldsymbol{\nabla} \cdot \nabla I$, here $\boldsymbol{\nabla}\cdot$ corresponds to divergence operator.

$$\boldsymbol{\nabla} \cdot \nabla I = (\frac{\partial}{\partial x}\hat{i} + \frac{\partial}{\partial y}\hat{j}) \cdot (\frac{\partial I}{\partial x}\hat{i} + \frac{\partial I}{\partial y}\hat{j}) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

This expression can also be represented by $\nabla^2 I$ also known as laplacian operator. This will give us information about the changes between the neighbourhood values and the given pixel value. If the laplacian value is high enough then it can be considered that there is a high probability that the pixel's information is lost.

As image contains discrete data , we considered differences between neighbourhood pixels and given pixel as analogues to gradient calculation. So, gradients in 8 neighbourhood directions are carried out as follows:

$$\begin{gathered}
\nabla_N I(x,y) = I(x-1,y) - I(x,y) \\
\nabla_S I(x,y) = I(x+1,y) - I(x,y) \\
\nabla_W I(x,y) = I(x,y-1) - I(x,y) \\
\nabla_E I(x,y) = I(x,y+1) - I(x,y) \\
\nabla_{NW} I(x,y) = I(x-1,y-1) - I(x,y) \\
\nabla_{NE} I(x,y) = I(x-1,y+1) - I(x,y) \\
\nabla_{SW} I(x,y) = I(x+1,y-1) - I(x,y) \\
\nabla_{SE} I(x,y) = I(x+1,y+1) - I(x,y)
\end{gathered}$$

Here, x represents the row number and y represents the column number. Direction of the gradient vector is represented as $\nabla_{DIR}$.

We have gradients calculated for 8 neighbourhood pixels, to calculate laplacian, which applies double partial differentiation, we will consider four directions for double differentiation which are:

$$\begin{aligned}
\boldsymbol{\nabla} \cdot \nabla I \quad = \quad & |\nabla_N I(x,y) + \nabla_S I(x,y)| + \\
& |\nabla_E I(x,y) + \nabla_W I(x,y)| + \\
& |\nabla_{NE} I(x,y) + \nabla_{SW} I(x,y)| + \\
& |\nabla_{SE} I(x,y) + \nabla_{NW} I(x,y)|
\end{aligned}$$

Adding the following four expressions will provide us a value analogous to laplacian. Now we have a metric to determine whether the image pixel is known or lost.We can set an upper bound for the laplacian of a pixel to be considered as a known pixel.

we can use $L_{i,j}$ as to denote if pixel is known or unknown,

$$L_{i,j} = \begin{cases} 0, & pixel_{i,j}'s \ \ information \ \ is \ \ lost, \\ 1, & pixel_{i,j} \ \ is \ \ correct \end{cases}$$

Step ii) Now to recover lost pixels , there are various methods that can be used.Pixel values strongly correlate with its neighbourhood , various methods estimate the pixel values on the basis of its neighbourhood. Some of them are median filtering, avg filter ,etc. Authors in [10] had selected one of the neighbourhoods, which can be vertical , horizontal , tilted at 45 degree or 135 degree. The selection process of the neighbourhood was based on standard deviation of all the neighbourhood known pixel values, neighbourhood with minimum standard deviation is selected as optimal.Then weighted avg is used to estimate the pixel. We are using a similar approach , but we are taking a 5x5 neighbourhood around the pixel to speed up the computation , the neighbourhood only consists of known non-black pixels only. Pixels which are near to the lost pixel has more impact , hence the weights for known neighbourhood pixels will be euclidean distance from our pixel (rp) with lost information.

$$||p, rp||_2 = \sqrt{(p_x - rp_x)^2 + (p_y - rp_y)^2}$$

Hence the restored pixel value will be,

$$Ir(x,y) = \frac{\sum_{\forall p \in neighbours} \frac{I(i,j) \times L_{i,j}}{||p,rp||_2}}{\sum_{\forall p \in neighbours} \frac{L_{i,j}}{||p,rp||_2}}$$

Ir(x,y) will the recovered pixel value. The Time complexity of the code will $O(rows * cols)$.

**Results:**

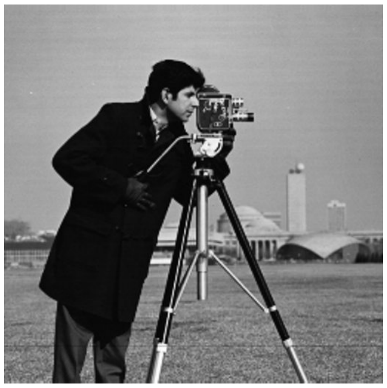


Fig. 1. Original Image

Fig. 2. Original Gaussian Image

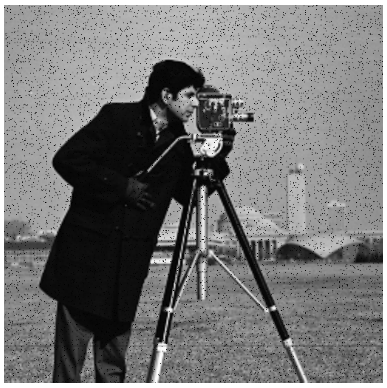


Fig. 3. Randomly 10,000 pixels removed

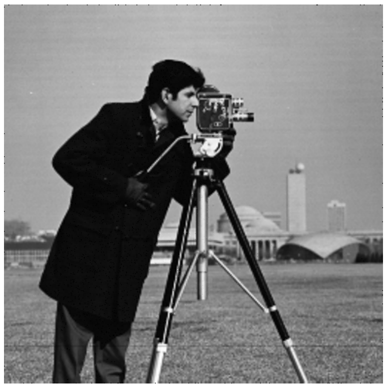


Fig. 4. Recovered Image from Fig 3

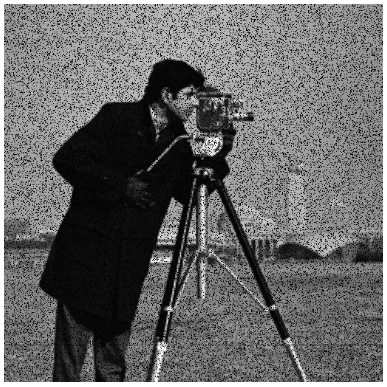


Fig. 5. Randomly 50,000 pixels removed

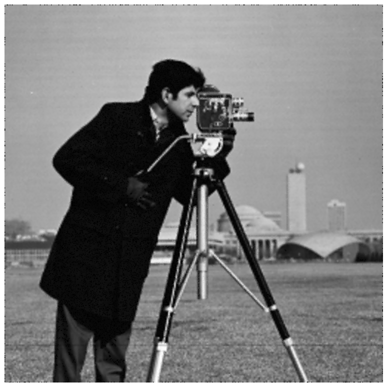


Fig. 6. Recovered Image from Fig 5

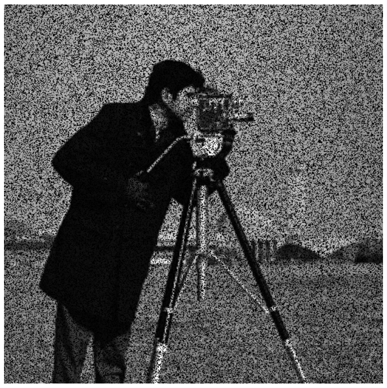


Fig. 7. Randomly 100,000 pixels removed

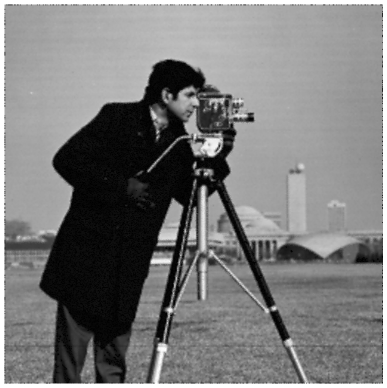


Fig. 8. Recovered Image from Fig 7

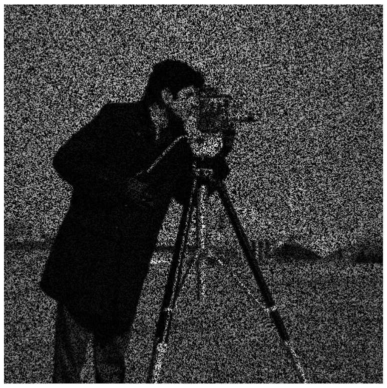


Fig. 9. Randomly 200,000 pixels removed

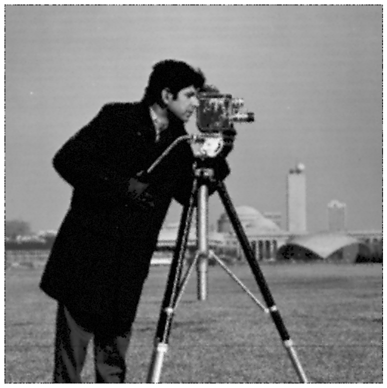


Fig. 10. Recovered Image from Fig 9

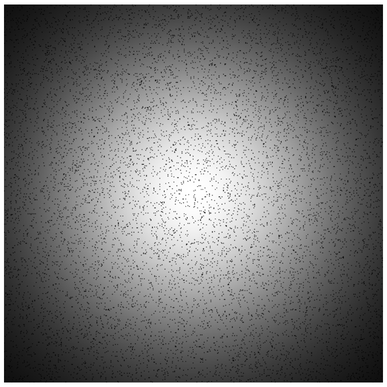


Fig. 11. Randomly 10,000 pixels removed

Fig. 12. Recovered Image from Fig 11

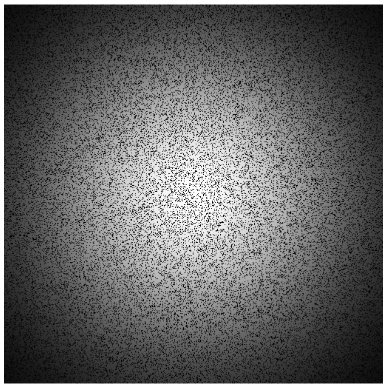


Fig. 13. Randomly 50,000 pixels removed

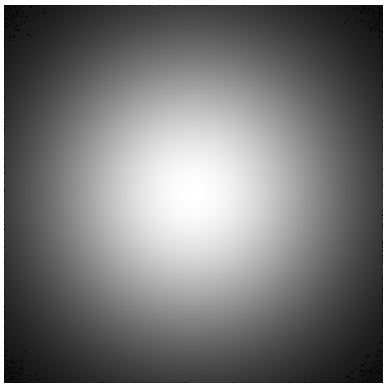


Fig. 14. Recovered Image from Fig 13

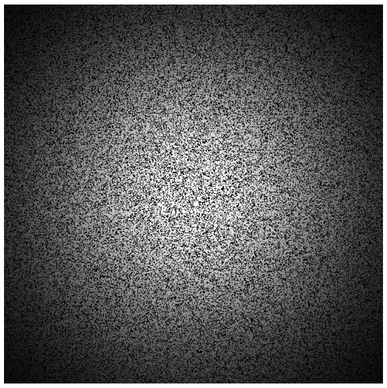


Fig. 15. Randomly 100,000 pixels removed

Fig. 16. Recovered Image from Fig 15

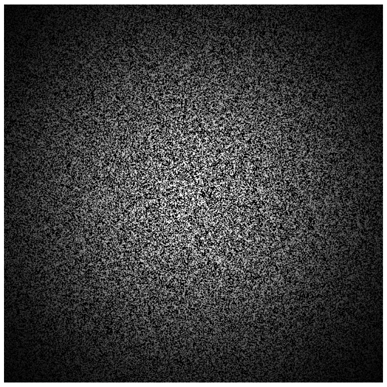


Fig. 17. Randomly 200,000 pixels removed

Fig. 18. Recovered Image from Fig 17

The block diagram shown below shows the working of the serial algorithm 1. Firstly, an image is given as an input. For every pixel, if the pixel value is non-zero, we do nothing. Else, we calculate the gradient between the pixel and its 8 neighbours. Now, if the divergence is less than the chosen threshold value, then we do not process it further. Else we process it and replace it with the weighted average of surrounding pixels which are not missing. So, in this manner, we get the denoised output image.

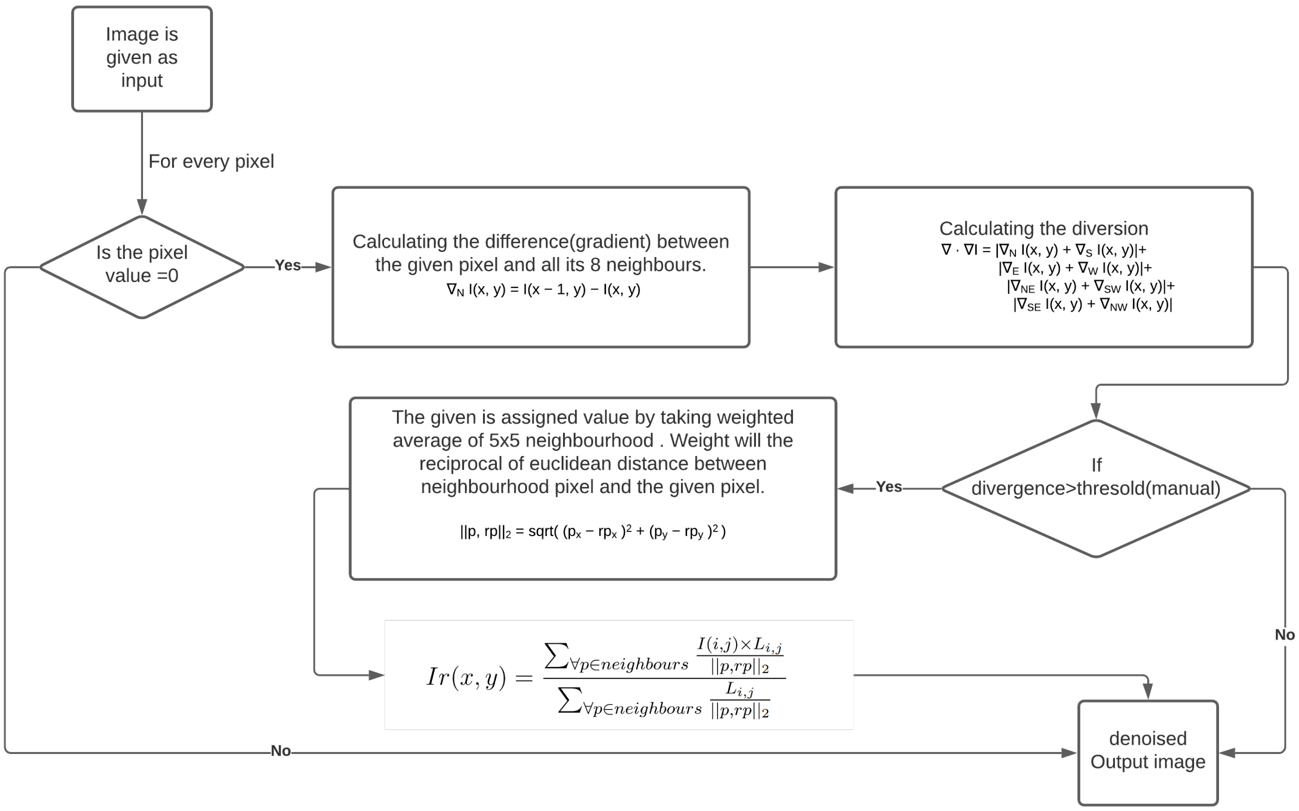


Image is given as input
For every pixel
Is the pixel value =0
Yes
Calculating the difference(gradient) between the given pixel and all its 8 neighbours.
$\nabla_N I(x, y) = I(x-1, y) - I(x, y)$
Calculating the diversion
$\nabla \cdot \nabla I = |\nabla_N I(x, y) + \nabla_S I(x, y)| + |\nabla_E I(x, y) + \nabla_W I(x, y)| + |\nabla_{NE} I(x, y) + \nabla_{SW} I(x, y)| + |\nabla_{SE} I(x, y) + \nabla_{NW} I(x, y)|$
If divergence>thresold(manual)
Yes
The given is assigned value by taking weighted average of 5x5 neighbourhood . Weight will the reciprocal of euclidean distance between neighbourhood pixel and the given pixel.
$||p, rp||_2 = \mathrm{sqrt}((p_x - rp_x)^2 + (p_y - rp_y)^2)$
$Ir(x,y) = \frac{\sum_{\forall p \in neighbours} \frac{I(i,j) \times L_{i,j}}{||p,rp||_2}}{\sum_{\forall p \in neighbours} \frac{L_{i,j}}{||p,rp||_2}}$
No
No
denoised Output image

Fig. 19. Block diagram of the serial algorithm 1

### *B. Algorithm 2*

Honeycomb patterns appear in fibre optics bundle images, the pattern size and shape depends on how the optic fibres are bundled. Due to repetitive nature of this pattern, it can be considered as a high frequency component while the original image pixels will be low frequency components. Hence, we can apply low pass filtering[11] in frequency domain of the image. This method was suggested in [9], they suggested to apply predefined filter mask to remove high frequency components. Here, we have tried to remove frequency components with frequency greater than a threshold. The mask formed will be in a circular manner.

$$\mathcal{F}(x,y) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f(x,y)e^{-j2\pi(x\frac{m}{M}+y\frac{n}{N})}$$

$$f(m,n) = \frac{1}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} \mathcal{F}(x,y)e^{j2\pi(x\frac{m}{M}+y\frac{n}{N})}$$

The code is developed in MATLAB, for 2D-FFT many algorithms are available to increase its computation speed such as Cooley-Tukey algorithm. After applying inverse Fourier transform we can recover the image. Pre-processing such as histogram equalization can be helpful to improve result. Various parallelization methods have also been developed to improve 2D-FFT speed. Many methods to restore fibre bundle images have been developed, which gives better result than Low Pas Filtering, but spectral filtering are still the fastest methods. It is beneficial when probe imaging system is used for real time imaging.

**Results:**

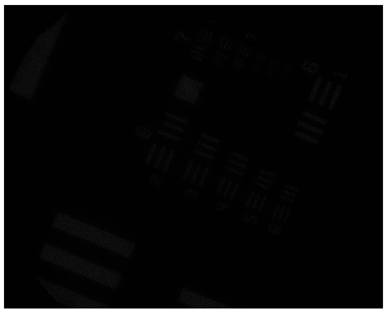


Fig. 20. Fiber Bundle Image (USAF)

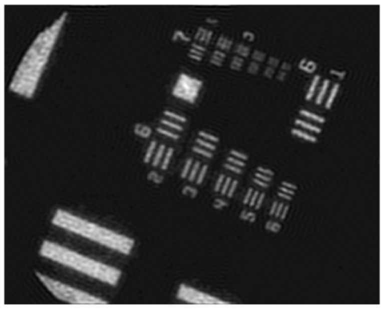


Fig. 21. Recovered Image from Fig 19

## IV. Why need of parallelization:

Serial approach has major setbacks for images having bigger size.As our main goal is to provide a approach which can perform well in real time application , we have to optimise further. This can be achieved by using parallel approach.Our task is to scan every pixel and to perform operation only on lost pixels.The operation also requires memory access of its neighbourhood pixels.Our approach includes mapping computing each output pixel using one thread. We cannot parallelize further as operation has O(1) time complexity.We can optimise by working on memory access pattern and to minimise global memory access by utilising shared memory.Our challenge is to manage shared memory resources as we are not processing all pixels in particular manner; but processing only lost pixels having random pattern (having pixel value 0).

## V. Problems in Parallelization

Many image restoration methods involves solving of the Langrangian optimization problem, which requires frequent updates of the pixel values, and ours is no different. Naive parallelization can be achieved by assigning restoration of each pixel to a single thread. This implementation will require constant global memory reading and writing. As each pixel value after every iteration will be result of its neighbour pixels, whose values are required to be calculated from previous iteration. For which we will require constant writing in global memory. Using shared memory can cause problems, since some pixels will not have their neighbours in the same block. Here, we are trying to develop a serial code for which the computational cost can be reduces such that it can be used in real time imaging. Our challenge will be to develop a highly optimize 2D-FFT parallel code and similarly for 2D-IFFT, and to implement CUDA based code for Algorithm 1.

## VI. Parallelized algorithm : Approach 1

The block diagram shown below shows the working of the parallelized algorithm 1. Firstly, an image is given as an input. Now, each output pixel is mapped to a single thread. And for each thread, we perform the serial algorithm 1 that we mentioned above. So, in this manner, we get the denoised output image by using a parallelized algorithm.

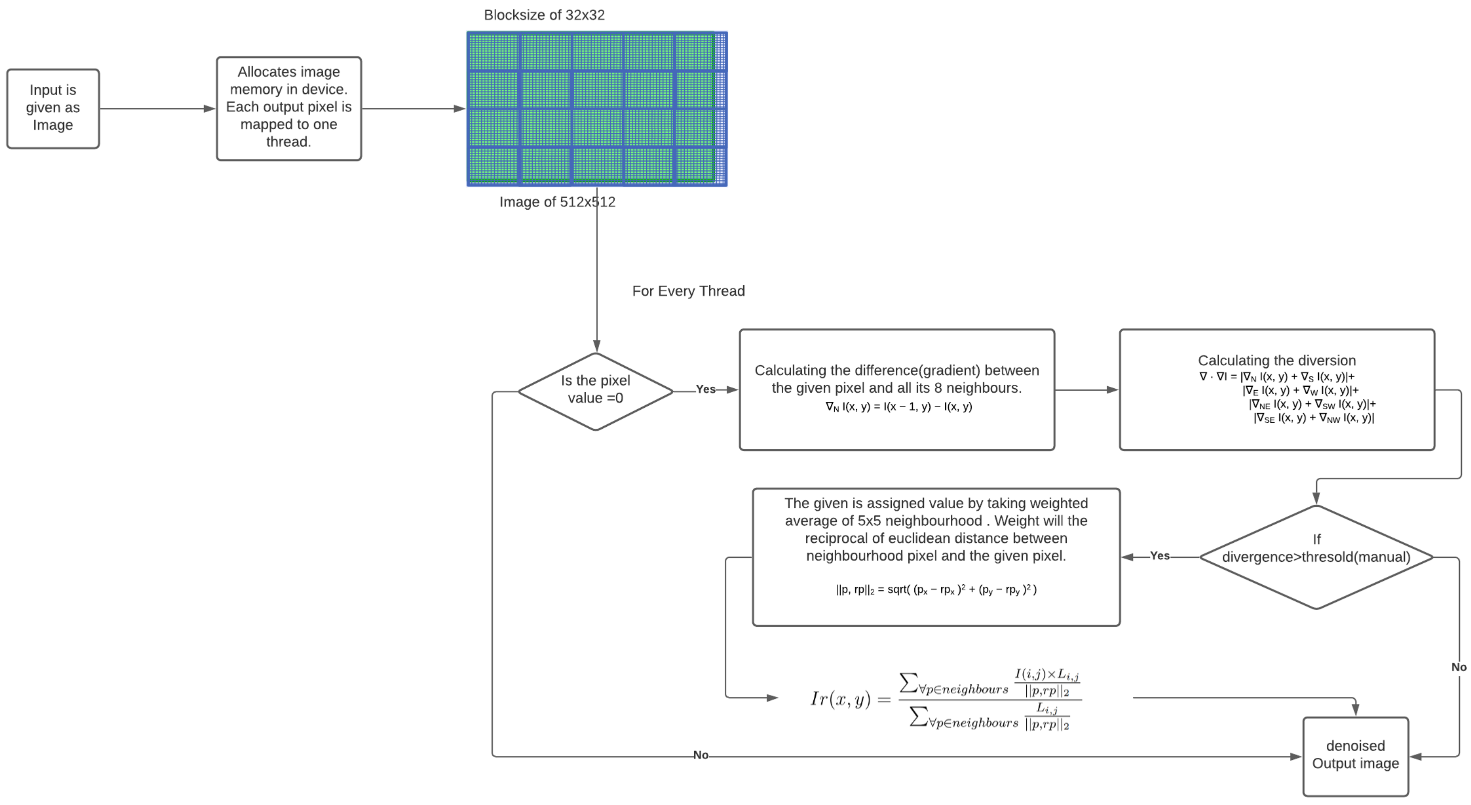

Blocksize of 32x32
Input is given as Image
Allocates image memory in device. Each output pixel is mapped to one thread.
Image of 512x512
For Every Thread
Is the pixel value =0
Yes
Calculating the difference(gradient) between the given pixel and all its 8 neighbours.
$\nabla_N I(x, y) = I(x - 1, y) - I(x, y)$
Calculating the diversion
$\nabla \cdot \nabla I = |\nabla_N I(x, y) + \nabla_S I(x, y)| + |\nabla_E I(x, y) + \nabla_W I(x, y)| + |\nabla_{NE} I(x, y) + \nabla_{SW} I(x, y)| + |\nabla_{SE} I(x, y) + \nabla_{NW} I(x, y)|$
If divergence>thresold(manual)
Yes
The given is assigned value by taking weighted average of 5x5 neighbourhood . Weight will the reciprocal of euclidean distance between neighbourhood pixel and the given pixel.
$||p, rp||_2 = sqrt( (p_x - rp_x )^2 + (p_y - rp_y )^2 )$
$Ir(x,y) = \frac{\sum_{\forall p \in neighbours} \frac{I(i,j) \times L_{i,j}}{||p,rp||_2}}{\sum_{\forall p \in neighbours} \frac{L_{i,j}}{||p,rp||_2}}$
No
No
denoised Output image

Fig. 22. Block diagram of the parallelized algorithm 1

### A. *Parallel vs Serial*

Algorithm 1's major time is consumed in identification and correction of noisy pixels.Our first approach is to allocate one thread for each pixel of output image.Now, for parallel approach we have measured total time taken by memcpy and kernel to compare with serial's identification and correction time.Hence, as shown in figure(25), serial time for smaller size images are less compared to parallel.For smaller size images parallel code's major time is consumed by data transfer operation (memcpy).Time taken by memcpy remains almost constant throughout different sizes, while time taken by kernel varies.For image size of 1024x1024 or higher has better performance on GPU for parallel code than serial code.As shown in figure(26) , the speedup is gained upto 11.8 for image of size 8192x8192.Parallel code is giving better performance than serial code on greater scale while serial code will be more preferable for smaller size of images. In earlier approach, we were getting different  high values of MSE(18 for serial  23 for parallel). For correcting  improving this result, we modified our approach slightly which incorporates using more memory to maintain the updated data  avoid in-place update which was affecting subsequent calculations. As a result, MSE value was brought down to 2.06 for both serial and parallel approaches. In account of using double memory, it is resulting in speedup getting halved than previous code.

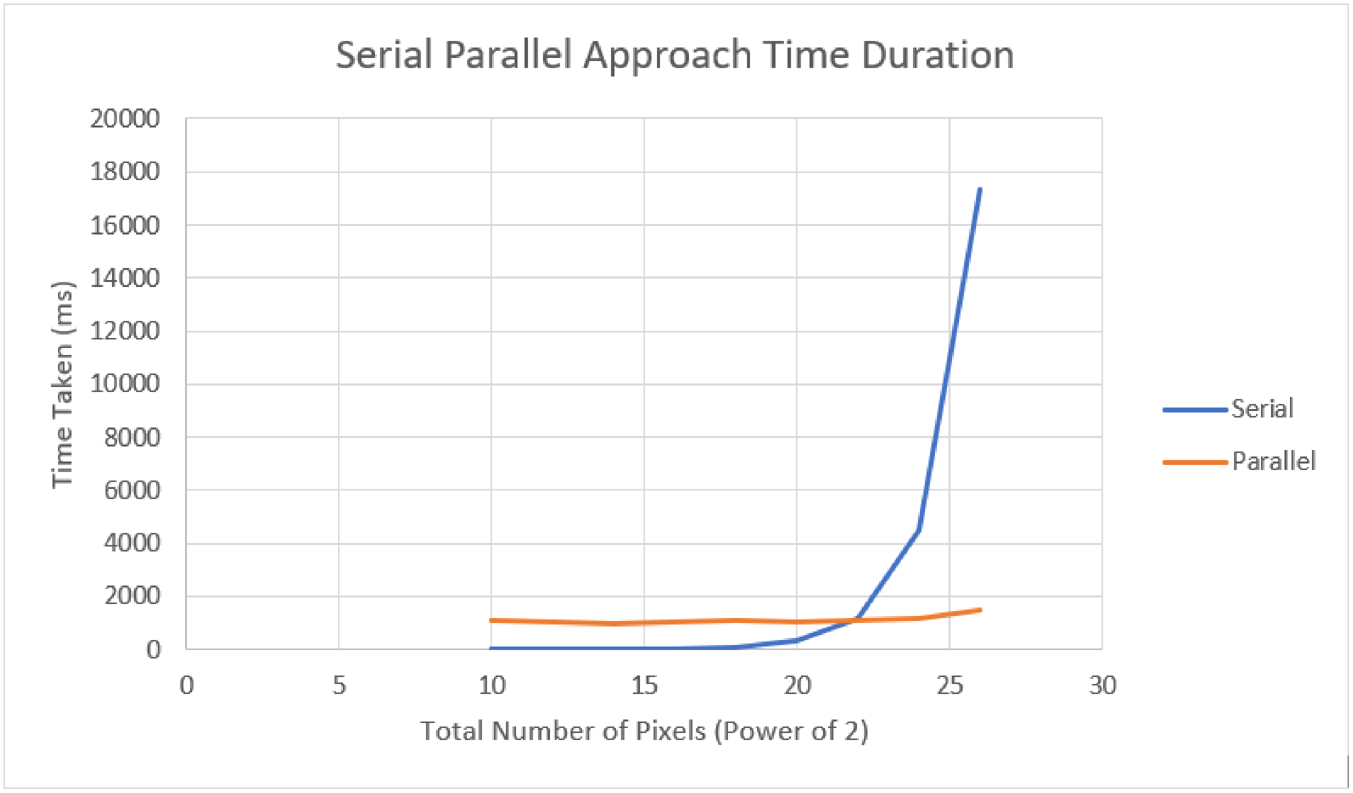


Fig. 23.  Time taken by serial and parallel codes

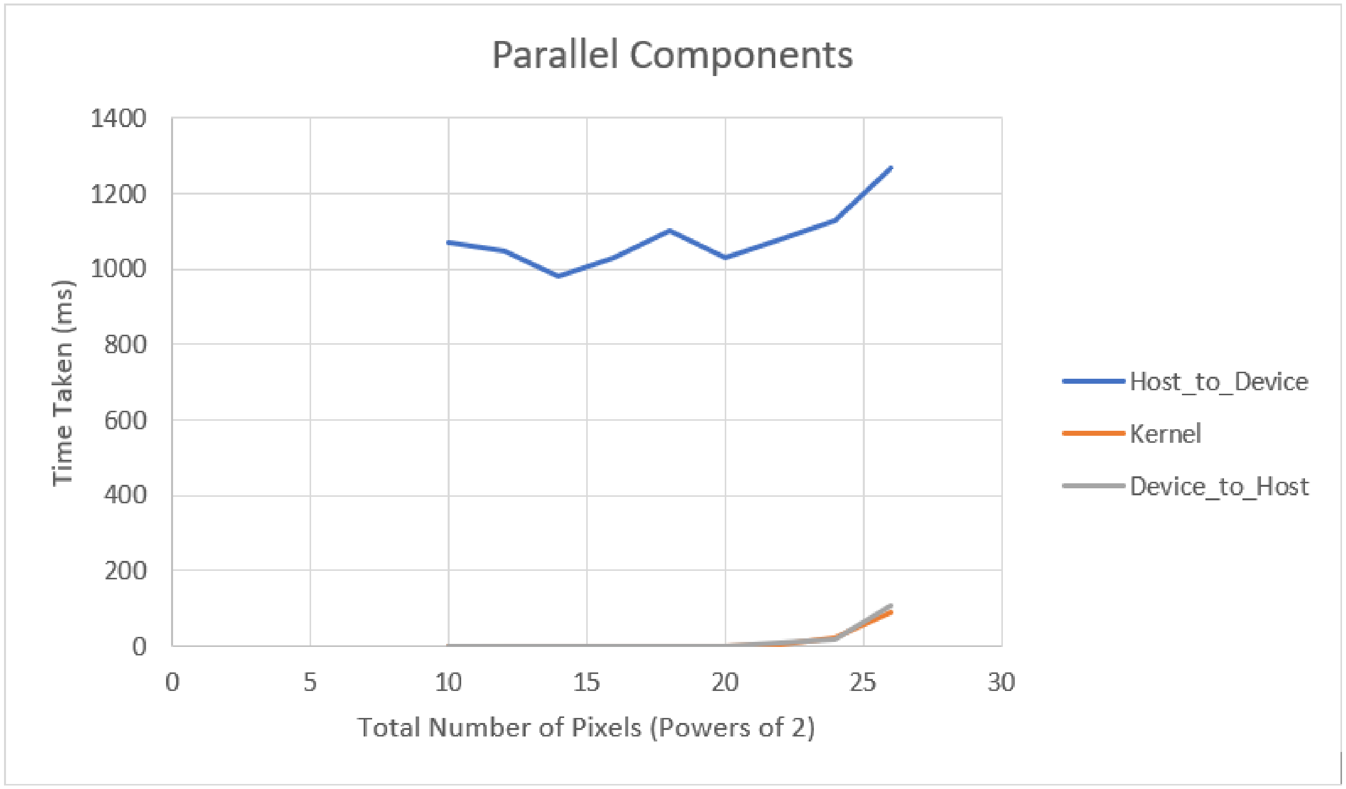


Fig. 24. Components of the parallel code

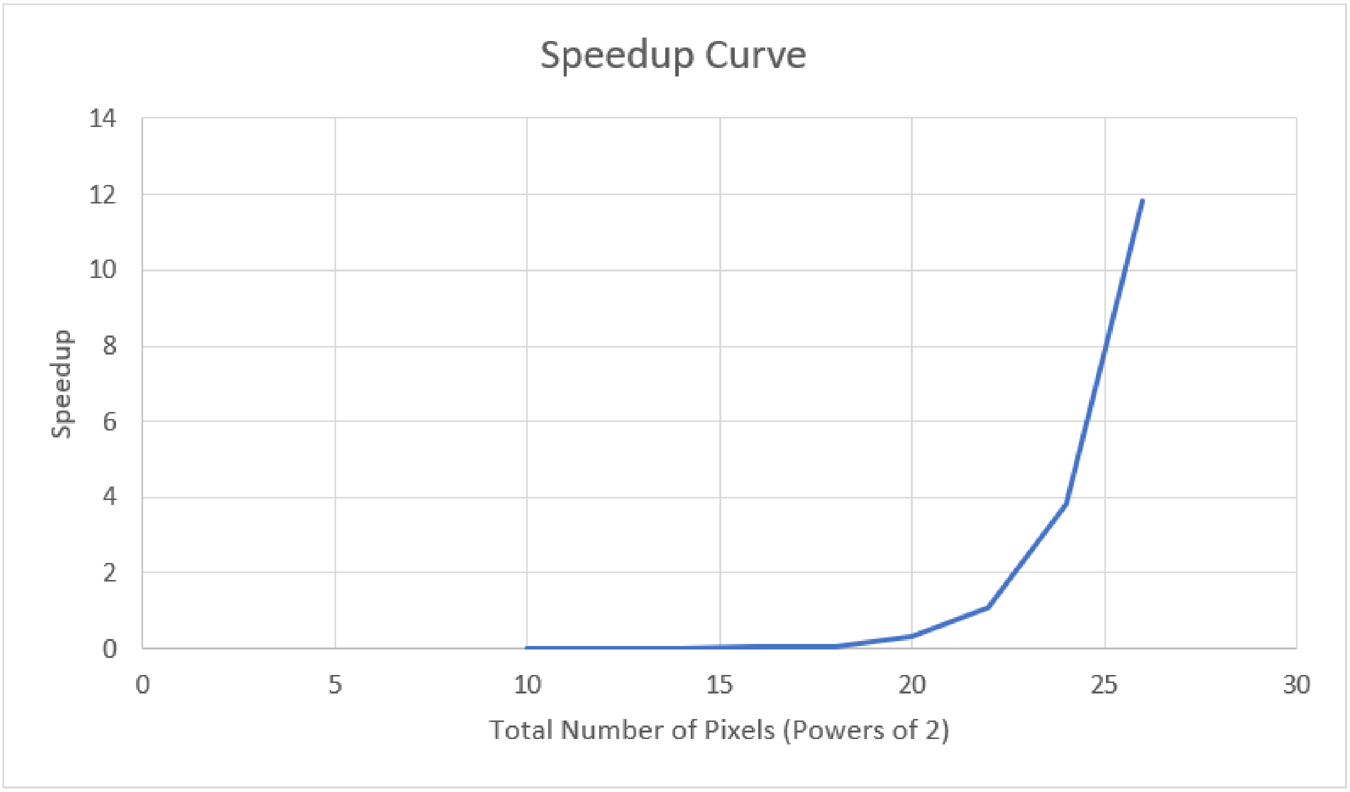


Fig. 25. Speedup curve

## B. Theoretical Performance

Here, the GPU we have used for our task is Tesla K40 which has computational peak performance of 4290 GFlops, whereas 288Gb/s is the peak memory bandwidth performance. Lets say p = Total no. of black pixels/ Total no. of pixels ,

We have assumed that original black pixels (which are not lost pixels) are zero.

For Calculating Memory Bandwidth of our code: [ (3*n$^2$) $+ (1-p)(n^2) + p(60-35*p)*n^2] * 4/TotalTimeTakenbyourAlgorithm$

For Naive implementation CGMA will depend on the number of black pixels present in image, as majority of work is applied on black pixels.

Avg CGMA ratio = p*(CGMA for black pixels) + (1-p)*(CGMA for non black pixels)

Avg CGMA ratio = p*(( 287+400*(1-p) ) /( 35+ 25*(1-p))) + (1-p)* (2/1)

( Here 1-p is there in CGMA for black pixels as we only consider non black pixels for weighted average)

Avg CGMA ratio = p*[(687-400p)/(60-25p)] + (1-p)*2

Avg CGMA ratio = p*[(567-350p)/(60-25p)] + 2

As, p increases CGMA ratio increases, so for p=0 , CGMA=2 ,which is a memory bound problem, but for p=1/2, CGMA= 6.12, which is compute bound.

We can optimize both number of computations and memory accesses.

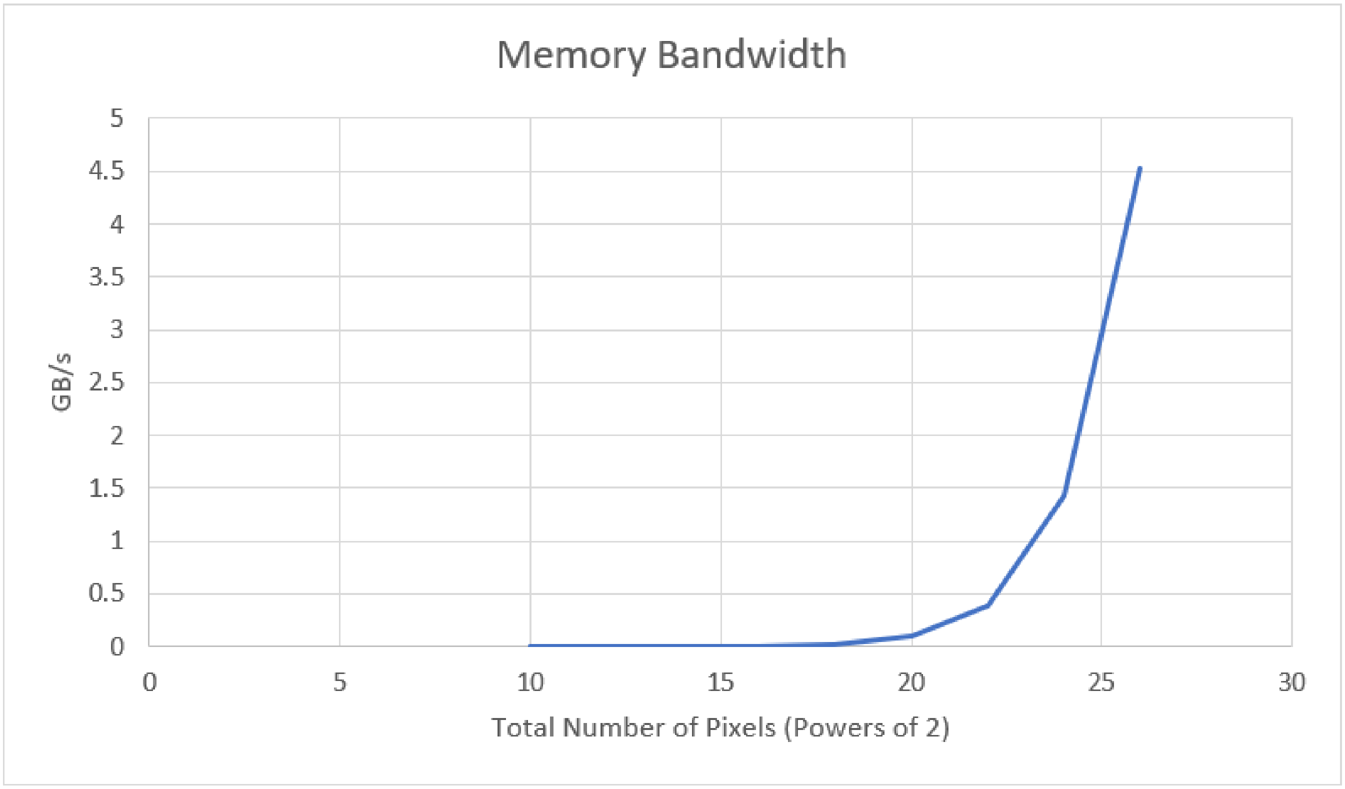
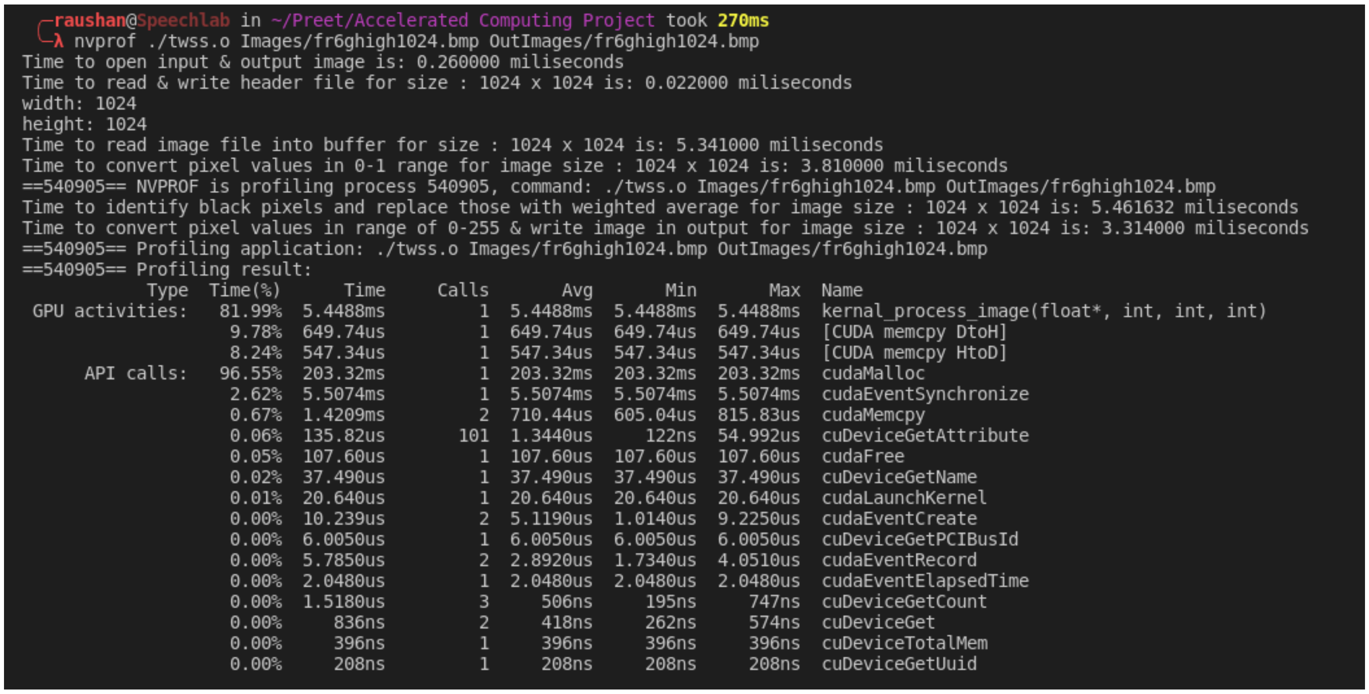


Fig. 26. Bandwidth curve



```
╭─raushan@Speechlab in ~/Preet/Accelerated Computing Project took 270ms
╰─λ nvprof ./twss.o Images/fr6ghigh1024.bmp OutImages/fr6ghigh1024.bmp
Time to open input & output image is: 0.260000 miliseconds
Time to read & write header file for size : 1024 x 1024 is: 0.022000 miliseconds
width: 1024
height: 1024
Time to read image file into buffer for size : 1024 x 1024 is: 5.341000 miliseconds
Time to convert pixel values in 0-1 range for image size : 1024 x 1024 is: 3.810000 miliseconds
==540905== NVPROF is profiling process 540905, command: ./twss.o Images/fr6ghigh1024.bmp OutImages/fr6ghigh1024.bmp
Time to identify black pixels and replace those with weighted average for image size : 1024 x 1024 is: 5.461632 miliseconds
Time to convert pixel values in range of 0-255 & write image in output for image size : 1024 x 1024 is: 3.314000 miliseconds
==540905== Profiling application: ./twss.o Images/fr6ghigh1024.bmp OutImages/fr6ghigh1024.bmp
==540905== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   81.99%  5.4488ms         1  5.4488ms  5.4488ms  5.4488ms  kernal_process_image(float*, int, int, int)
                    9.78%  649.74us         1  649.74us  649.74us  649.74us  [CUDA memcpy DtoH]
                    8.24%  547.34us         1  547.34us  547.34us  547.34us  [CUDA memcpy HtoD]
      API calls:   96.55%  203.32ms         1  203.32ms  203.32ms  203.32ms  cudaMalloc
                    2.62%  5.5074ms         1  5.5074ms  5.5074ms  5.5074ms  cudaEventSynchronize
                    0.67%  1.4209ms         2  710.44us  605.04us  815.83us  cudaMemcpy
                    0.06%  135.82us       101  1.3440us     122ns  54.992us  cuDeviceGetAttribute
                    0.05%  107.60us         1  107.60us  107.60us  107.60us  cudaFree
                    0.02%  37.490us         1  37.490us  37.490us  37.490us  cuDeviceGetName
                    0.01%  20.640us         1  20.640us  20.640us  20.640us  cudaLaunchKernel
                    0.00%  10.239us         2  5.1190us  1.0140us  9.2250us  cudaEventCreate
                    0.00%  6.0050us         1  6.0050us  6.0050us  6.0050us  cuDeviceGetPCIBusId
                    0.00%  5.7850us         2  2.8920us  1.7340us  4.0510us  cudaEventRecord
                    0.00%  2.0480us         1  2.0480us  2.0480us  2.0480us  cudaEventElapsedTime
                    0.00%  1.5180us         3     506ns     195ns     747ns  cuDeviceGetCount
                    0.00%     836ns         2     418ns     262ns     574ns  cuDeviceGet
                    0.00%     396ns         1     396ns     396ns     396ns  cuDeviceTotalMem
                    0.00%     208ns         1     208ns     208ns     208ns  cuDeviceGetUuid
```

Fig. 27. GPU Profiling (nvprof)

Fig. 28. Original Image

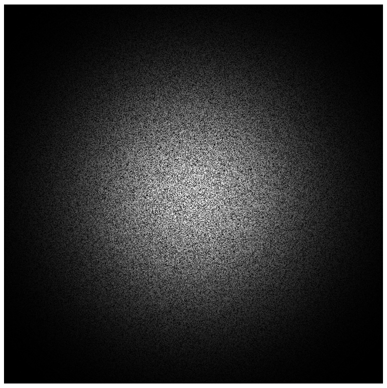


Fig. 29. Noisy Gaussian Image ( 50 % of pixels are removed)

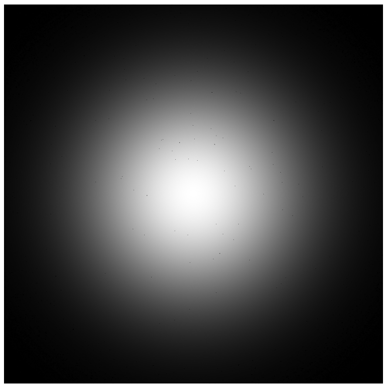


Fig. 30. Image recovered by CPU
MSE : 2.06

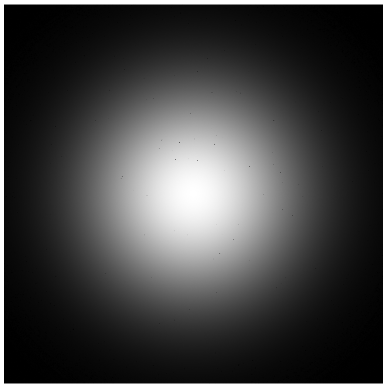


Fig. 31. Image recovered by GPU
MSE : 2.06

## C. Further Parallelization

We tried to further optimize the code using the concept of tiling. Our main idea was to perform calculation for block of dimension 32x32 by bringing data of size 36x36 having the same center from global memory to each block. Following this, each corner-thread will bring the data of 3x3 = 9 data-values. Threads on the sides will bring two more pixels and all other threads will bring single value associated with respective thread ids. But we were not able to produce any significant results in the given time frame.

## References

[1] P. Getreuer, "Rudin-osher-fatemi total variation denoising using split bregman," *Image Processing On Line*, vol. 2, pp. 74–95, 2012.

[2] P. Rodríguez, "Total variation regularization algorithms for images corrupted with different noise models: a review," *Journal of Electrical and Computer Engineering*, vol. 2013, 2013.

[3] J. Liu, W. Zhou, B. Xu, X. Yang, and D. Xiong, "Honeycomb pattern removal for fiber bundle endomicroscopy based on a two-step iterative shrinkage thresholding algorithm," *AIP Advances*, vol. 10, no. 4, p. 045004, 2020.

[4] C. Renteria, J. Suárez, A. Licudine, and S. A. Boppart, "Depixelation and enhancement of fiber bundle images by bundle rotation," *Appl. Opt.*, vol. 59, pp. 536–544, Jan 2020.

[5] J. Shao, J. Zhang, X. Huang, R. Liang, and K. Barnard, "Fiber bundle image restoration using deep learning," *Opt. Lett.*, vol. 44, pp. 1080–1083, Mar 2019.

[6] F. Dong, Y. Chen, D.-X. Kong, and B. Yang, "Salt and pepper noise removal based on an approximation of l0 norm," *Computers Mathematics with Applications*, vol. 70, no. 5, pp. 789–804, 2015.

[7] R. D. Adam, P. Peter, and J. Weickert, "Denoising by inpainting," in *Scale Space and Variational Methods in Computer Vision* (F. Lauze, Y. Dong, and A. B. Dahl, eds.), (Cham), pp. 121–132, Springer International Publishing, 2017.

[8] J. Shao, "Computational methods for improving fiber bundle imaging systems," 2019.

[9] A. Shinde and M. Vadakke Matham, "Pixelate removal in an image fiber probe endoscope incorporating comb structure removal methods," *Journal of Medical Imaging and Health Informatics*, vol. 4, 04 2014.

[10] X. Zhang, F. Ding, Z. Tang, and C. Yu, "Salt and pepper noise removal with image inpainting," *AEU - International Journal of Electronics and Communications*, vol. 69, no. 1, pp. 307–313, 2015.

[11] goodday451999, "MATLAB – Ideal Lowpass Filter in Image Processing." https://www.geeksforgeeks.org/matlab-ideal-lowpass-filter-in-image-processing. 2010-09-30.