GPU accelerated non-local means image denoising

Adithya Upadhya
Department of Computer Science, Virginia Tech
Blacksburg, Virginia
adithyau@vt.edu

***Abstract* – Non-local means algorithm is a very important algorithm for denoising images corrupted with Gaussian White noise. It exhibits superior denoising quality and retention of fine details and edges when compared to other denoising algorithms such as Gaussian filtering and neighborhood filtering. However, NL means suffers from a very high time complexity and thus, deployment of this algorithm in the real-time systems would be challenging owing to high computational requirements. There is a tremendous scope for parallelization of the algorithm owing to several mutually exclusive and independent computations being performed for every voxel. While there have been previous research work in the field of parallel NL means, in this paper, we elaborate on 1) Successful implementation of Serial & Parallel NL means using C++ and CUDA framework for denoising 3D MR (magnetic resonance) Images, 2) achieve a remarkable speedup in the range of 23-24 using parallel NL means on Nvidia GPU, 3) Fail to achieve good denoising results using custom L_k norm weighting function whose performance is inferior to that of Gaussian weighting function with respect to denoising quality and noise recution**

Keywords – Non local means, CUDA framework, GPGPU, parallel computing, 3D medical image processing

Introduction

Non Local Means

For an arbitrary pixel i in a noisy image I, let $v(i)$ represent the pixel value of the image corrupted with a Gaussian white noise $n(i)$ and $u(i)$ represents the ground truth (original pixel of image). The white noise $n(i)$ belongs to the independent and identically distributed values from a Gaussian distribution with zero mean and σ^2 standard deviation [1]. Our objective is to compute the ground truth with speed and accuracy.

$$v(i) = u(i) + n(i)$$

$$n(i) \sim N[0, \sigma^2]$$

In theoretical NL means, the estimated denoised pixel is computed by taking a weighted average of all the pixels in the noisy image. However, since this would be very computationally expensive, the search window is restricted to smaller dimensions of $S \times S$ for 2D images and $S \times S \times S$ for 3D images [1]. However, for algorithmic analysis, we will assume that the search window will consist of the entire image. For a noisy image $v = \{ v(i) \mid i \in I \}$, let $NL[v](i)$ represent the NL means estimate for a pixel i of image v. The NL means estimate is the weighted average of all the pixels in the image.

$$NL[v](i) = \sum_{j \in I} w(i,j)\, v(j) \qquad (1)$$

Now we will discuss the derivation of the weights $w(i,j)$. In the given Image I, a similarity between any two pixels i and j are measured as the decreasing function of the weighted Euclidean distance $\left|\left|v(N_i) - v(N_j)\right|\right|^2_{2,\sigma}$ where σ represents the standard deviation of the Gaussian Kernel. Therefore, in NL means, the neighborhood of similar pixels is given more weight than neighborhood the of dissimilar pixels.

The weights can be defined as (here the denominator represents the normalizing factor and h represents the degree of filtering. Greater value of h results in the faster exponential decay of the weights) [1]:

Therefore, the high time complexity of the algorithm is quite evident from the explanation given above. The time complexity of the algorithm is $O(L^2S^2N^2)$ for 2D images and $O(L^3S^3N^3)$ for 3D images. L^3, S^3 and N^3 represents the local window, search window and image dimensions respectively for the 3D image. In the next sections, we explain our strategy to ameliorate the high time complexity.

$$w(i,j) = \frac{e^{-\frac{\left\|v(N_i)-v(N_j)\right\|^2_{2,\sigma}}{h^2}}}{\sum_{j\epsilon I}\left(e^{-\frac{\left\|v(N_i)-v(N_j)\right\|^2_{2,\sigma}}{h^2}}\right)} \qquad (2)$$

Figure 1. NL means strategy: pixel neighborhood of A is more similar to B's neighborhood than C's neighborhood. Therefore, B gets more weight than the region C. $w(A,B) > w(A,C)$

Proposed work

We can infer from the algorithm described above that there is a tremendous scope for parallelization of the algorithm. For every pixel i in the image I, NL means computes the weighted average of all the pixel neighborhood within the search window dimensions for denoising. We can infer that the weighted average computation for every pixel of the image is mutually exclusive and independent. For every pixel in the image, instead of computing the corresponding denoised pixel in serial, we could compute the denoised pixel for every image pixel in parallel. This concept lays down the foundation for the implementation aspects of the proposed parallel NL means.

A. Implementation details and the required computational resources

In the original paper, the NL means algorithm was applied on 2D images. In this project, we will implement NL means algorithm for 3D images. 3D images usually consist of medical image data such as MRI scans [8] and CT scans. In 3D images, the picture elements are called 'voxels' and in 2D images, the picture elements are called 'pixels'. Our objective is to compute the NL means estimate in parallel for every voxel i in the 3D image I, i.e. $NL[v](i)$. We will use Nvidia CUDA framework, a modern GPU parallel processing framework for developing a parallel NL means for 3D images. The first stage of the project will consist of developing a serial version of the algorithm for denoising 3D images using C++. Subsequently, the second stage will consist of development of a parallel NL means using CUDA C++. The NL means estimate for every voxel in the 3D image will be computed in parallel on GPU hardware. After the computation is complete, the data consisting of denoised voxels will be copied from the Device to the host. Some Image preprocessing tasks such as reading DICOM images will be performed in the Matlab environment. However, the entire NL means algorithm will be executed in the GPU hardware.

The quality of denoising is measured using MSE (mean squared error) and $PSNR$ (peak signal to noise ratio). Here $\bar{Y}_i$, Y_i and MAX represents the NL means estimate of ian^{th} voxel, Ground truth and maximum value attained by a voxel in the 3D image respectively.

$$MSE = \frac{1}{N} \times \sum_1^N (\bar{Y}_i - Y_i)^2 \quad (3)$$

$$PSNR = 10 \log_{10} \left(\frac{MAX^2}{MSE} \right) \quad (4)$$

The computation and benchmark analysis will be performed using personal computational resources. These computational resources include Intel Core i7-7500U Processor, 16 GB RAM and Nvidia GeForce GTX 950M (640 CUDA cores, 915 MHz clock speed, 80.16 GB/s memory bandwidth and 2GB GDDR5 GPU memory). The software specifications for the development of this project include CUDA Toolkit v9.0.176, Microsoft Visual Studio 2017 v15.4.5, and Matlab 2015a.

B. Research aspects of the project

The three main research aspects of the project have been explained below.

1) Parallel NL means for denoising 3D images

In the original research paper, the algorithm was developed for denoising 2D images. In this research project, we will extend the algorithm for denoising 3D images. First, a serial NL means will be developed using C++ and next, a parallel NL means will be developed using CUDA C++ which will target GPU hardware for computing NL estimates for every image voxel in parallel. Finally, the speedup of the parallel NL means with respect to serial NL means will be determined and analyzed.

2) Optimal CUDA kernel launch parameters

For denoising 3D images, the kernel launch parameter for threads per block will be defined by a $dim3$ data type which consists of x, y & z members. In this project, we will try to determine the optimal x, y and z launch parameters representing the threads per block which gives us the maximum performance/yield from the GPU hardware. The optimal x, y, z launch configurations which result in minimum execution time.

3) Analyzing different weighting functions in NL means for performance and denoising quality

The original NL means computes the weights using a Gaussian weighting function. In this project, we will test the algorithm with different weighting functions. In this project, we plan to test parallel NL means with L_k distance norms [9] for different values of k and use a linear weighted mean instead of Gaussian weighting function. The proposed weighting function will be tested against the Gaussian weighting function for execution speed and denoising quality. The denoising quality will be compared using MSE and $PSNR$. Here, $w'(i,j)$ represents the linear weighting function.

$$L_k = (x_1^k + x_2^k + x_3^k + \cdots + x_n^k)^{\frac{1}{k}} \quad (5)$$

$$T = \sum_{j \in I} \left|\left| v(N_i) - v(N_j) \right|\right|_{K,\sigma}^K \quad (6)$$

$$w'(i,j) = \frac{T - \left|\left| v(N_i) - v(N_j) \right|\right|_{K,\sigma}^K}{\sum_{j \in I} \left(T - \left|\left| v(N_i) - v(N_j) \right|\right|_{K,\sigma}^K \right)} \quad (7)$$

$$NL[v](i) = \sum_{j \in I} w'(i,j)\, v(j) \quad (8)$$

In $eq(6)$, T represents the sum of all the computed L_K norms for all the neighborhood of all voxels in the image (for computational efficiency, all voxels within search window dimensions). The weight assigned to a voxel $w'(i,j)$ in $eq(7)$ is proportional to the difference between T and the computed L_K norm for the neighborhood of the particular voxel. The denominator in $eq(7)$ represents the normalizing factor. Finally, the new NL estimate is computed using taking the weighted average of all the voxels neighborhoods within the search window dimensions as described in $eq(8)$.

Literature Review and Related Research

Since NL means is a very important and highly cited research work in the field of Image processing, a reasonable amount of research papers have been published for improving the denoising quality and performance of the algorithm. NL means has also been modified to suit the needs of various different applications. Recently, Zhu Et al. published a research paper comparing the performance of parallel NL means developed using OpenCL and OpenMP on CPU, GPU and MIC platform [2]. Modified NL means has been used to denoise Magnetic Resonance images with spatially varying Rician and Gaussian noise levels for an improved performance [3].

Coupe Et al. proposed an optimized block wise NL means algorithm for denoising 3D MR images with blockwise implementation and parallel computation [4]. Mingliang Et al. have proposed an improved NL means algorithm for denoising medical images by introducing a novel noise weighting function and parallelization of the algorithm [5].

Recently, Huang Et al. developed a parallel NL means algorithm targeting the Intel Xeon Phi hardware and MIC

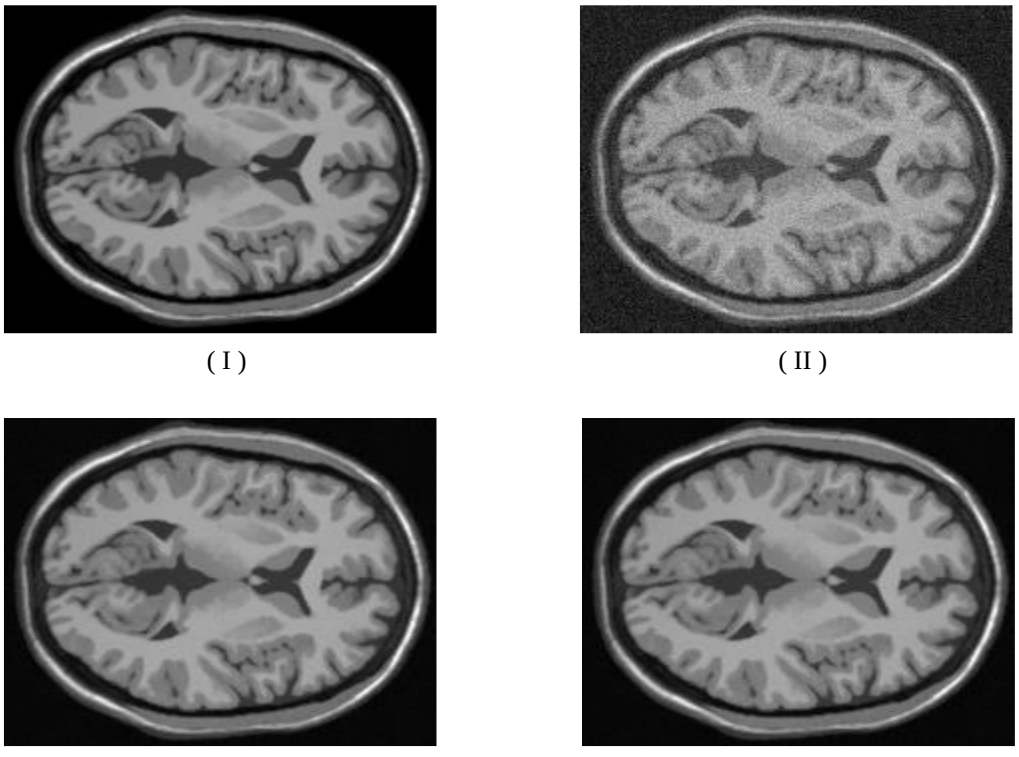
architecture for remote sensing (RS) images processing to satisfy the real-time requirements for certain applications [6]. A lot of work has also been carried out to determine the optimal parameters of NL means and the filter has been improved for denoising the MR images corrupted with Rician noise [7].

Experiments and results

1. GPU accelerated NL Means

Design and development of a GPU accelerated NL means algorithm with a remarkable speedup is the greatest success of this project. All experiments were performed on a Brain 3D MR (Magnetic Resonance) Image which is available on BrainWeb [10]. The dimensions of the 3D MR image was $(181 \times 217 \times 51)$ and a Gaussian white noise with $\boldsymbol{\sigma = 10}$ (standard deviation) was introduced to generate a noisy image corrupted with Gaussian white noise. The initial phase of the project consisted of development of a serial NL means algorithm. In terms of computational dwarfs, NL means algorithm falls into the category of 'Structured Dwarfs'. The serial NL means was implemented in C++ using Visual Studio 2017 IDE. Following a successful serial implementation, emphasis was laid on building a parallel version of NL means using CUDA C++. Programming and development of GPU CUDA code was accomplished using Nsight Visual Studio platform, a powerful and feature rich CUDA development plugin for Visual Studio IDE.

Matlab was primarily used for preprocessing the 3D MR images. The preprocessing stages involved reading the MAT files (Matlab file for storing variables), introducing a Gaussian white Noise of $\sigma = 10$ (standard deviation) and converting the file into a Binary file with floating point precision data. The NL means algorithm would be executed on the noisy image data stored in the binary file. Finally ,the denoised image generated using the algorithm would be written to an output binary file which would be then parsed by Matlab for analysis and visualization. Matlab was chosen owing to its flexibility and built-in support for MAT files, binary files and various other file formats. Other strong advantages of Matlab includes a powerful image processing toolbox, fast and convenient matrix processing operations.



(I) (II) (III) (IV)

Figure 2: **30th Slice** of 3D MR Image (I) Ground Truth, (II) Image corrupted with Gaussian White noise $\sigma = 10$, (III) CPU generated denoised image (serial NL means), (IV) GPU generated denoised image (parallel NL means).

Figure 2 represents the **30th** slice of the 3D MR Image (there are 51 slices in total along the z-axis, $181 \times 217 \times 51$). The remarkable denoising quality of NL means algorithm is quite evident from the parts III & IV of Figure 2. However, on a close comparison of part I with part II & III, we can observe that there is a minuscule loss of detail, edges, and sharpness when denoising is performed. Having said that, NL means algorithm has been proven to outperform several other renowned denoising algorithms in terms of retention of edges, details and denoising quality [1].

1.1. Serial NL Means vs. Parallel NL means performance

In this section, we will briefly compare the execution time and performance of Serial NL means and Parallel NL means. The serial NL means has a complexity of $O(L^3S^3N^3)$ for 3D images where L^3, S^3 and N^3 represents the local window, search window and image dimensions respectively for the 3D image ($L \leq S \leq N$). The variable N^3 would remain constant throughout the experiment since all experiments were performed on a single MR image for consistency. The local window dimensions were maintained constant at $(3 \times 3 \times 3)$ and the Search window dimensions were varied from $(7 \times 7 \times 7)$ to $(17 \times 17 \times 17)$. As a part of future work, we would like to perform similar experiments by varying the local windows dimensions whilst maintain a constant search window dimensions.

In case of Serial NL means, the algorithm execution time was computed using the $clock()$ routines provided by $time.h$ header file. For the Parallel NL means (CUDA), the high precision time measurements were performed using CUDA events. In the following table and graphs, the serial and parallel NL means execution time along with speedup is shown. In addition, the denoising quality measured using PSNR and MSE have also been tabulated below. **Higher** PSNR represents better denoising quality and **Lower** MSE represents better denoising quality.

Search Window Dimensions	Serial NL means (CPU) exec. time (seconds)	Parallel NL means (GPU) exec. time (seconds)	Speedup GPU vs. CPU	MSE	PSNR
$7 \times 7 \times 7$	51.19	2.07	**24.73**	12.09	37.31
$9 \times 9 \times 9$	104.57	4.35	**24.04**	12.02	37.33
$11 \times 11 \times 11$	188.19	7.87	**23.91**	12.09	37.31
$13 \times 13 \times 13$	308.29	12.98	**23.75**	12.19	37.27
$15 \times 15 \times 15$	468.16	19.76	**23.69**	12.28	37.24
$17 \times 17 \times 17$	676.29	28.39	**23.82**	12.38	37.20

Table 1: Tabulated data of GPU vs. CPU speedup for different Search window dimensions. MSE & PSNR are also shown here.

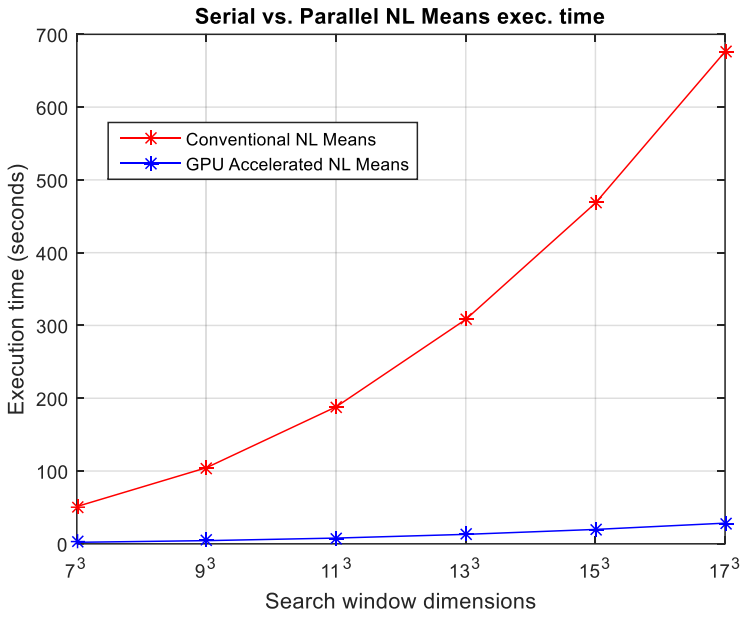


Figure 3: CPU vs. GPU NL Means execution time for increasing search window dimensions

From the Table 1 and Figure 4, we can draw an inference that there is a very minimal change in the **MSE** and **PSNR** for higher Search window dimensions. For a real-world image dataset, it is very difficult to determine the optimal Search window and local window dimensions which could give us the best denoising quality and execution performance.

Theoretically, a bigger search window would give us better denoising quality since the algorithm will search for more number of similar voxel neighborhoods. However, this comes at an expense of a Cubic growth in execution time as the Search window dimensions increase.

2. Optimal CUDA kernel launch parameters

In this experiment, we are evaluating the optimal 3D kernel launch parameters which would give us the best execution time and performance. Specifically, we tried to determine the optimal CUDA thread block dimensions which consists of the tuple $(\boldsymbol{threadsX}, \boldsymbol{threadsY}, \boldsymbol{threadsZ})$ which indicates the number of threads along x, y and z dimensions respectively for a thread block. For GPUs with compute capability 5.0, CUDA framework restricts the number of threads per block to **1024**. Additional restriction is that the maximum permissible dimensions of a thread block is $(1024, 1024, 64)$ for GPUs with compute capability 5.0. Next section consists of a brief discussion on reducing the permutations of thread block dimensions (x, y, z).

2.1 Managing massive permutations of thread block dimensions

For this experiment, we faced a serious challenge because we had to perform simulations on all the permissible (x, y, z) combinations under the framework constraints $x \leq 1024$, $y \leq 1024$, $z \leq 64$ and $x \times y \times z \leq 1024$. The official CUDA toolkit documentation [13] stated that, since the GPU **warp size** is 32, it is always recommended to choose the launch parameters such that the total number of threads per block is a multiple of 32, i.e. $(threadsPerBlock \,\%\, 32 = 0)$ for optimal results and occupancy.

Despite the additional constraint $x \times y \times z \leq 1024$ and $(x \times y \times z) \,\%\, 32 = 0$, we were still left with around 176,000 permissible (x, y, z) permutations. Conducting a simulations on so many combinations would take days and hence, we needed to reduce the scope of our project further.

Finally, we started referring to the **Nvidia CUDA occupancy calculator**. In order to compute the Multiprocessor Warp Occupancy, the calculator requires several variables such as Registers per thread, Shared

Memory per block and Compute Capability. The Registers per thread and Shared memory per block were determined by passing the option $--\boldsymbol{ptxas} - \boldsymbol{options} = -\boldsymbol{v}$ to the NVCC compiler. Using the compilation logs, I was able to determine that my CUDA code was using 56 registers, 372 bytes of constant memory. There wasn't any shared memory used. Hence, based on the selected parameters, we get the results shown in the Figure 5 added below.

Theoretical occupancy is not an indicator of the optimal performance. However, we chose to use the data from the occupancy calculator to reduce the total number of permutations of threads per block. From Figure 4, we can see that theoretical maximum occupancy is attained when the Threads per Block is either **576, 384 or 288.** Hence, our constraints were now updated as:

$$(x * y * z) \leq 1024 \;\; \&\& \;\; ((x * y * z) == 576 \;\; || \;\; (x * y * z) == 384 \;\; || \;\; (x * y * z) == 288) \qquad (9)$$

Using Equation (9), we were able to drastically reduce the number of (x, y, z) permutations to **119**. The Parallel NL means was executed for all the various (x, y, z) configurations satisfying the Equation (9). The experiment was performed on the same 3D MR Image $(181 \times 217 \times 51)$ with local a window $(3 \times 3 \times 3)$ and search window $(11 \times 11 \times 11)$. The execution time for each kernel launch configuration was determined and the results were analyzed.

From figure 5, an inference can be drawn that Parallel NL means performs particularly well for all launch configurations with lower x-dimensions $(x \leq 20)$. The execution time is low even for higher values of y-dimensions as long as the x-dimension is less than 20. This is certainly an anomaly and the reason behind these results could be owing to the intrinsic factors in the GPU hardware or architecture. The **best performing** kernel launch configurations were $(4, 24, 3), (4, 36, 2), (4, 32, 3)$ and $(4, 24, 4)$. The **worst performing** kernel launch configurations were $(48, 4, 3), (48, 4, 2), (36, 4, 2)$ and $(72, 4, 2)$. From these results, it's quite evident that higher value of x-dimensions leads to a significant loss of performance.

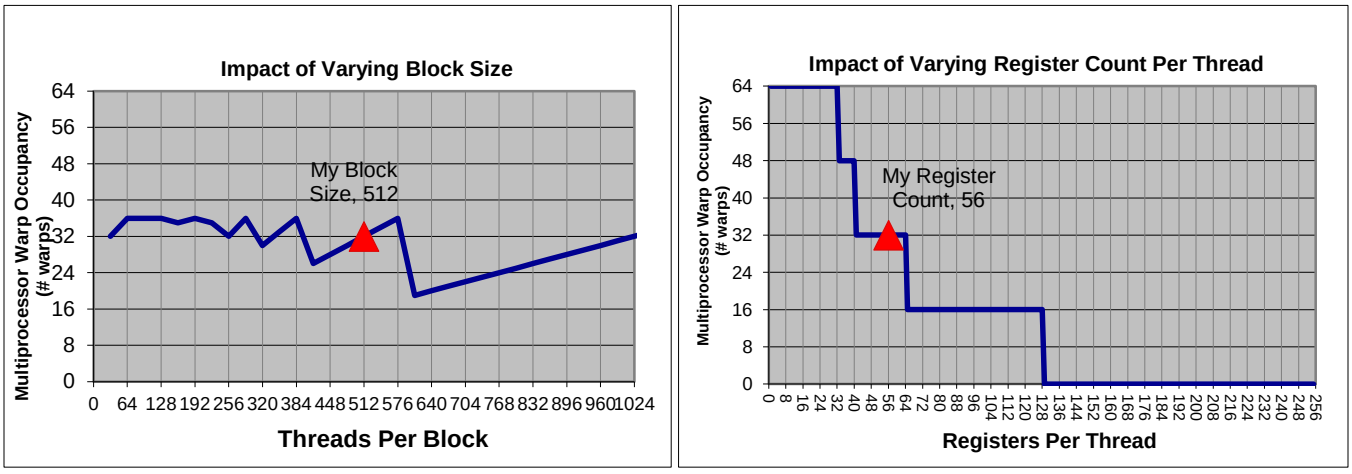


Figure 4: Graphs from CUDA occupancy calculator.
Impact of varying threads per block and registers per thread on Multiprocessor warp occupancy

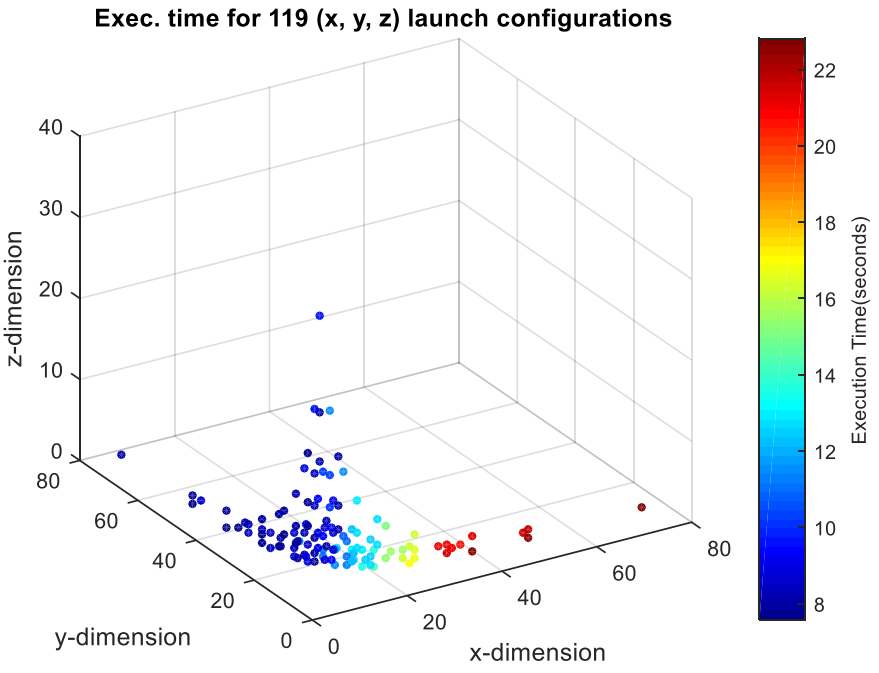
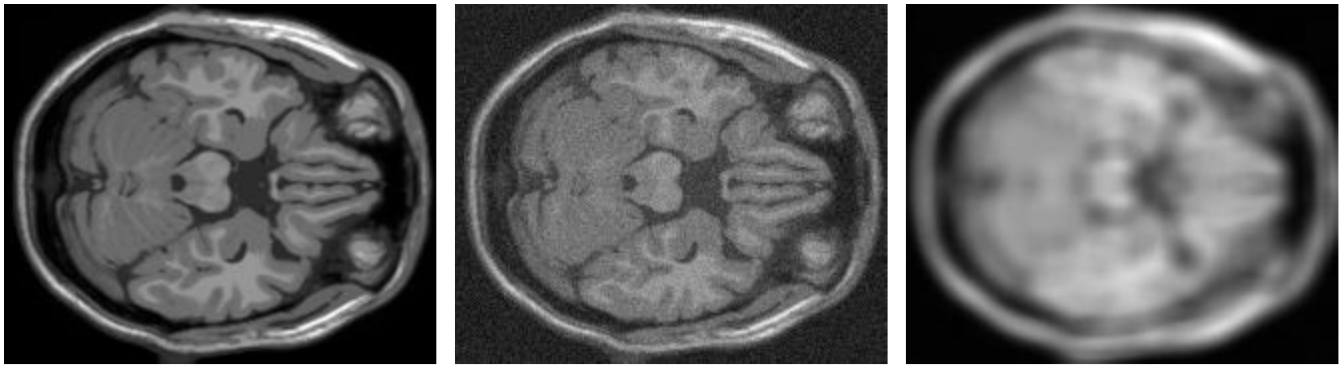


Figure 5: Execution time for different (x, y, z) configurations have been plotted

3. L_k norm linear weighting functions in NL means (Negative results)

We obtained negative results in these experiments. L_1, L_2, L_3 & L_4 norm weighted averages were used as the weighting functions. From these experiments and from Figure 6, we can conclude that the L_k norm weighted average weighting function may not be a good weighting function for denoising Gaussian white noise owing to blurring, loss of edges and details. On the other hand, the Gaussian Weighting function as shown in figure 2 performs much better with respect to the denoising quality.

From the figure 6, it is quite evident that the L_k weighted average leads to a drastic loss of detail and blurring. Hence, L_k weighted average may not be a suitable denoising weighting function. However, it could be used as a suitable blurring function based on the degree of blurring required. More amount of research and mathematical analysis is required for making concrete conclusions. All the weighting functions from L_1 to L_4 generate a blurry image as an output. Thus, Gaussian weighting function is superior in terms of the denoising quality



(I) (II) (III)

Figure 6: 1st Slice of the 3D MR Image. (I) Ground truth, source image, (II) Image corrupted with Gauss noise (σ=10), (III) Denoised image generated using L_4 weighted average (PSNR: 21.5, MSE: 460.8)

Conclusions and Future work

From the aforementioned experiments, we were successful in (i) developing Parallel NL means algorithm using CUDA framework, (ii) achieving a significant speedup in the range 23-24 on GPU and (iii) determining and analyzing the optimal CUDA kernel launch parameters based on the CUDA occupancy calculator. However, our proposed work on L_1, L_2, L_3 & L_4 weighted average weighting functions failed to produce good quality denoising results and this is one failure of this project. We realized that the weighted average estimation may not be the best denoising weighting function for denoising images corrupted with Gaussian white noise.

In our quest to determine the optimal CUDA kernel launch parameters, we encountered a massive challenge of managing several thousand permutations of launch configurations. Using CUDA occupancy calculator and NVCC compiler logs, we restricted our kernel launch parameters such that the product would be either 576, 384 or 288. Hence, the number of possible launch configurations reduced to 119. Of course, the results of this experiment are **not exhaustive**. The multiprocessor occupancy might not always be the only factor which determines the kernel execution time. Hence, the future work could consist of conducting simulations on all the possible (x, y, z) permutations of launch configurations for a comprehensive analysis.

Furthermore, the Gaussian weighting function may not be the best weighting function for denoising. There could be better weighting functions that are yet to be explored. Future research could also consist of statistical analysis and development of improved and efficient weighting functions for denoising.

In the execution time benchmark, the search window dimensions were varied while the local window dimensions were maintained constant. Future work in this field could also consist of a benchmark where local window dimensions would be varied and execution time along with MSE and PSNR would be analyzed for different local window dimensions.

References

[1] Buades, Antoni, Bartomeu Coll, and J-M. Morel. "A non-local algorithm for image denoising." Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on. Vol. 2. IEEE, 2005.

[2] Zhu, Huming, et al. "A parallel non-local means denoising algorithm implementation with OpenMP and OpenCL on Intel Xeon Phi Coprocessor." Journal of Computational Science 17 (2016): 591-598.

[3] Manjón, José V., et al. "Adaptive non-local means denoising of MR images with spatially varying noise levels." Journal of Magnetic Resonance Imaging 31.1 (2010): 192-203.

[4] Coupé, Pierrick, et al. "An optimized blockwise nonlocal means denoising filter for 3-D magnetic resonance images." IEEE transactions on medical imaging 27.4 (2008): 425-441.

[5] Mingliang, X., Pei, L., Mingyuan, L., Hao, F., Hongling, Z., Bing, Z., & Liwei, Z. (2016). Medical image denoising by parallel non-local means. Neurocomputing, 195, 117-122.

[6] Huang, Fang, et al. "A Parallel Nonlocal Means Algorithm for Remote Sensing Image Denoising on an Intel Xeon Phi Platform." IEEE Access 5 (2017): 8559-8567.

[7] Manjón, J. V., Carbonell-Caballero, J., Lull, J. J., García-Martí, G., Martí-Bonmatí, L., & Robles, M. (2008). MRI denoising using non-local means. Medical image analysis, 12(4), 514-523.

[8] Cocosco, C. A., Kollokian, V., Kwan, R. K. S., Pike, G. B., & Evans, A. C. (1997). Brainweb: Online interface to a 3D MRI simulated brain database. In NeuroImage.

[9] "Norm (Mathematics)." Wikipedia, Wikimedia Foundation, 5 Apr. 2018, en.wikipedia.org/wiki/Norm_(mathematics).

[10] BrainWeb: Simulated Brain Database, brainweb.bic.mni.mcgill.ca/brainweb/.

[11] Salmon, Joseph. "On two parameters for denoising with non-local means." IEEE Signal Processing Letters 17.3 (2010): 269-272.

[12] Zhang, Kaibing, et al. "Single image super-resolution with non-local means and steering kernel regression." IEEE Transactions on Image Processing 21.11 (2012): 4544-4556.

[13] "CUDA Toolkit Documentation v9.1.85." *NVIDIA Developer Documentation*, docs.nvidia.com/cuda/