CUDA IMPLEMENTATION OF NONLOCAL MEANS ALGORITHM FOR GPU PROCESSORS

Article *in* Indian Journal of Computer Science and Engineering · February 2020

DOI: 10.21817/indjcse/2020/v11i1/201101057

3 authors:

Farha Wahid
Kannur University
16 PUBLICATIONS **20** CITATIONS
SEE PROFILE

Sugandhi K.
Kannur University
12 PUBLICATIONS **15** CITATIONS
SEE PROFILE

Raju G.
Christ University, Bangalore
61 PUBLICATIONS **314** CITATIONS
SEE PROFILE

CUDA IMPLEMENTATION OF NON-LOCAL MEANS ALGORITHM FOR GPU PROCESSORS

Farha Fatina Wahid
Department of Information Technology, Kannur University, Kerala, India
farhawahid@gmail.com

Sugandhi K
Department of Information Technology, Kannur University, Kerala, India
sugandhikgs@gmail.com

Raju G
Department of Computer Science and Engineering,
Christ (Deemed to be University), Bengaluru, India
kurupgraju@gmail.com

Abstract - Non-Local Means algorithm (NLM) is a prominent image denoising algorithm. One of the major limitations of NLM algorithm and its variants is the time requirement. In this era of high performance computing, an efficient alternative to reduce the time complexity of any algorithm is its parallelization. In this paper, a parallelized version of basic NLM algorithm using CUDA architecture is proposed. The algorithm is developed on NVIDIA GeForce 940M GPU which follows Maxwell architecture with 3 SMs and 384 CUDA cores. Experiments are carried out using selected set of natural and medical images of various sizes. Our proposed parallelized version of NLM algorithm reduces the time requirement approximately by 50% in comparison to its basic version and also achieves comparable denoising performance in terms of PSNR, SSIM and FSIM evaluation metrics. The proposal is a model which can be customized for newer GPU architectures.

***Keywords*: Image Denoising, Non Local Means Filtering, GPGPU, NVIDIA, CUDA**

1. Introduction

Non Local Means algorithm (NLM) is an image denoising algorithm developed by Buades, et al in 2005[A. Buades *et al*., (2005)] [Gonzalez & Woods, (2006)]. As the name implies, NLM algorithm uses the concept of non-local neighborhood instead of the conventional local neighborhood for image denoising. In local neighborhood filtering, a denoised pixel is obtained by performing some computations using its surrounding pixels whereas in non-local filtering, the entire pixels in an image are considered for denoising. The simplest local neighborhood filtering technique is mean filtering, where a noisy pixel is replaced using the average of pixels in its local neighborhood [Gonzalez & Woods,(2006)]. Non local Means algorithm, the foundation stone for non-local concept uses the weighted average of all the pixels in an image to replace a noisy pixel. The significance of NLM algorithm lies in its self-similarity property where the contribution of similar pixel neighborhoods for denoising a particular pixel is high compared to dissimilar pixel neighborhoods. This leads to a better denoised result compared to local mean filtering [Antoni Buades *et al*., (2005)]. The performance deterioration of mean filtering is mainly due to the equal contribution of all neighboring pixels irrespective of the similarity. Even though NLM algorithm produces better denoising results, there is possibility that the resultant image is smoothed due to variations in different parameters used in NLM algorithm. Another issue with NLM algorithm is its time complexity, as comparison of a pixel neighborhood with other pixel neighborhoods is resource demanding. [Antoni Buades *et al*., (2005)].

Parallelization of algorithm is one of the efficient ways to shorten the time requirement of complex problems. In a standalone system, parallelization can be achieved with multiple CPUs (cores), or GPUs [Su *et al*., (2014)], [Kadah *et al*., (2011)], [L. Wang *et al*., (2013)], [X *et al*., (2016)]. The limitation of cores in a CPU motivated the development of General Purpose computing on Graphics Processing Units (GP-GPU), with capabilities of performing computations generally executed by a CPU. Highly parallel architecture and massive number of cores in a GPU marked a huge shift from CPU processing to GP-GPU. One of the proponents of GP-GPU is NVIDIA Corporation [Sanders & Kandrot, (2011)]. NVIDIA developed an Application Programming Interface (API) to enable non-graphics applications to run on GPUs, called Compute Unified Device Architecture (CUDA) [*CUDA*, (2019)]. It allows C programmers to write parallel codes for GPU using a few easy to use language extensions. The number of CUDA cores available for processing depends upon the graphic card's CUDA architecture [*CUDA*

C programming Guide, (2019)], [Chakrabarti *et al*.,(2012)], [Che *et al*., (2008)], [Nickolls *et al*., (2016)]. For example, NVIDIA Quadro K600 graphics cards contain 192 CUDA cores [*NVIDIA Quadro*, (2019)] whereas NVIDIA GeForce 940M contains 384 CUDA cores [*NVIDIA GeForce 940M*, (2019)].

CUDA architecture is suitable for image processing applications (Zhiyi et al., 2008), (Y. K. Wang & Huang, 2014), (Kang et al., 2014), (Ferreiro et al., 2013), (J. Wang et al., 2014), (Gulo et al., 2016). In this paper, to address the time complexity of NLM algorithm, a model for parallelization of NLM algorithm using CUDA is proposed. Experiments conducted with chosen data sets proved that the speed of execution of NLM is considerably increased, without compromising on the output quality.

2. Related Works

In recent years, several image processing applications were developed using CUDA architecture. In 2008, Zhiyi, et al. proposed parallelization of several classical image processing algorithms such as histogram equalization, removing clouds, edge detection and DCT encode and decode. The authors mentioned that they ignored data transfer time between host memory and device memory while calculating computation time of CUDA code. They claimed that high speed up was obtained for all the parallelized versions of selected image processing algorithms [Zhiyi *et al*., (2008)]. A Cellular Neural Network (CNN) based high performance computing for real time image processing on GPU was proposed by Potluri et al. They suggested that the inherent massive parallelism of CNN along with GPU makes it an advantage for high performance computing platform, by accelerating the processing speed and reducing the execution time [Potluri *et al*., (2011)]. In order to overcome the weakness of Seeded Region Growing (SRG) algorithm due to its computation time, Park et al. proposed a novel method for its parallelization using CUDA architecture. The experimentation was carried out with single-core CPUs, quad-core CPUs and shader language programming using synthetic data sets. The authors claimed that CUDA based SRG algorithm outperforms the other three implementations [Park *et al*., (2014)]. To overcome the limitation of Fuzzy C-Means algorithm when applied on large data sets, Al-Ayyoub, et al. proposed a parallel implementation of brFCM, a faster variant of FCM algorithm, on two different GPU cards. The authors claimed that their implementation showed significant improvement over the traditional CPU sequential implementation[Al-Ayyoub *et al*., (2015)].

For denoising magnetic resonance images, GPU implementation of non-local maximum likelihood estimation (NLML) method was proposed in 2016 by Upadhya, et al. The authors claimed that the GPU based implementation of NLML on image denoising achieves significant speed up compared to the serial implementation [Upadhya *et al*., (2016)]. In 2016, Gulo, et al. presented and discussed implementation of a highly efficient algorithm for image noise smoothing based on general purpose computing on GPU techniques. They suggested that the use of GPU techniques facilitate quick and efficient smoothing of images even when performed on large dimensional data sets [Gulo *et al*., (2016)]. In 2017, Bahri, et al. introduced an algorithm to optimize the computing time of feature extraction methods for colour images. They compared the experimental results on CPU and GPU and demonstrated that the execution time was considerably reduced [Bahri *et al*., (2017)]. Hernández et al., in [Hernández *et al*., (2017)], proposed CUDA implementation of bio-inspired model for object classification by taking the advantage of the computational capabilities of GPU and claimed that their method was able to process images up to 90 times faster than the existing system.

3. GPU Architectures

GPU plays a prominent role in the performance of a graphics card. NVIDIA's Tesla architecture, introduced in 2006, is the first one which enabled non-graphics computations on GPU. In 2010, NVIDIA introduced its first complete GPU architecture, known as Fermi architecture[*NVIDIA Fermi*, (2019)]. Based on the foundation laid by the Fermi architecture, NVIDIA introduced a more efficient GPU in 2012, Kepler architecture[*GeForce_GTX_680*, (2018)]. In 2014, NVIDIA released a new GPU architecture called Maxwell architecture. It introduced a new design for the heart of GPU-SM (GPU – Streaming Multiprocessors). There are two generations of Maxwell architecture. The first generation contains GM10x series and second generation contains GM20x series architectures [Contributors, (2019)].

The first architecture in GM10x is GM107. It contains one GPC with 5 Maxwell Streaming Multiprocessors (SMMs). There are two 64 bit memory controllers, raster engine and an L2 cache in GPC. SMM contains a polymorph engine and an instruction cache. There are 128 CUDA cores in a single SMM. These CUDA cores are divided into 4 separate processing blocks. Each block contains an instruction buffer, a single warp scheduler with 2 dispatch units, a register file, 32 CUDA cores, 8 Load/Store units and 8 Special Function Units. Each pair of processing blocks shares a texture cache and 4 texture filtering units. An SMM also contains an L1 cache and 64 Kb shared memory. Here, L1 cache is combined with texture cache rather than shared memory as in previous architectures. The partitioning of CUDA cores helps in the reduction of computational latency and simplifies the design and scheduling logic of SMM. GTX850M and GTX860M follow Maxwell GM107 architecture. Another architecture in GM10x series is GM108. Its major difference from GM107 is that GPC contains only 3 SMMs, thereby resulting in 384 CUDA cores. A well-known architecture in GM20x series is GM204. It is composed of

4 GPCs with 16 SMMs and 4 memory controllers. GeForce GTX980 follows GM204 architecture [*GeForce-GTX-750-Ti*, (2018); *GeForce _GTX_980*, (2018)]. The current work is implemented using GeForce 940 M GPU which follows NVIDIA Maxwell GM108 architecture.

4. CUDA Programming Model

CUDA enables programmers to write simple and efficient parallel programs with less effort. CUDA programs are similar to conventional C programs with minor changes in syntax. For any CUDA program, there are Host and Device codes. Host codes runs serially on CPU whereas Device codes runs parallel on GPU. Generally, functions which run on GPU are known as kernel functions [*cuda-c-basics*, (2018)].

The basic processing flow of any CUDA program begins with copying required data from host memory to device memory. Once data are available in device memory, kernel functions perform necessary concurrent computations and finally returns the resultant data back to host memory.

The elementary unit of CUDA programs are threads. Collection of threads forms a thread block (or block) and collection of thread blocks forms a grid. Dimensions of grid and thread block are the essential prerequisite for kernel functions. The dimensions can be 1D, 2D or 3D. The CUDA defined structures uint3 and Dim3 are used to define the number of blocks in a grid (gridDim) and the number of threads in a block (blockDim). cudaMalloc() is used to dynamically allocate required memory to device pointers. The copying of data from host memory to device memory and vice versa is achieved by cudaMemcpy()[Sanders & Kandrot, (2011)].

Once gridDim, blockDim and Device data are available, kernel function is called. The entry to kernel function is marked by triple angle bracket. The general syntax of kernel function call is kernel_function_name <<<gridDim,blockDim>>> (parameters of kernel function). Kernel functions begin with __global__ qualifier. Generally, this qualifier is used for functions which are called from host and executed on device. Within the kernel functions, the current block in a grid and the current thread in a block are identified using an index value. blockIdx and threadIdx gives the index of block in a grid and thread in a block respectively [Wilt, (2013)].

When 2D grids and blocks are considered, respective 2D indices are denoted as blockIdx.x, blockIdx.y and threadIdx.x, threadIdx.y. The 2D global indices i and j of a thread are calculated as

$$x = \text{blockIdx.x} * \text{blockDim.x} + \text{threadIdx.x} \quad (1)$$

$$y = \text{blockIdx.y} * \text{blockDim.y} + \text{threadIdx.y} \quad (2)$$

The corresponding 1D global index of a thread in column major order is

$$\text{index} = x + y * \text{numRows} \quad (3)$$

where numRows indicate the number of rows in 2D input data [*cuda-c-basics*, (2018)].

Communication between threads generally can be done either using global memory or shared memory. Shared memory allows threads within a block to communicate with each other. __shared__ qualifier is used to define a shared variable to which data is copied from global memory and __syncthreads() command synchronizes write operation to a shared variable. Even though shared memory gives fast access to data, its smaller size is a limitation. Global memory is large and allows threads from different blocks to communicate each other but access time required is more. [*cuda_tutorial*, (2018)] [*cuda-c-basics*, (2018)] [Farber, (2011) [He *et al*., (2015)] [Wilt, (2013)].

5. NLM Algorithm

Non-local neighborhood based techniques plays a significant role in image denoising. The non-local means (NLM) algorithm was proposed by Buades et al. in 2005[Antoni Buades *et al*., (2005)]. In order to denoise a pixel value, weighted average of all the pixel neighborhoods are considered in which similar pixel neighborhood gives high weight and vice versa. This self-similarity concept is the backbone to all non-local image denoising techniques.

Given a noisy image I corrupted with additive white Gaussian noise, the de-noised image I' is given as,

$$I'(x) = \sum_{y \in I} w(x,y).I(y) \quad (4)$$

where $w(x,y)$ is the weight assigned to the y^{th} pixel for modifying the value of the x^{th} pixel. The weight can be calculated as

$$w(x,y) = \frac{1}{z(x)} e^{\frac{-\|I(N_x)-I(N_y)\|^2_{2,a}}{h^2}} \quad (5)$$

where $(x) = \sum_y e^{\frac{-\|I(N_x)-I(N_y)\|^2_{2,a}}{h^2}}$.

N_x and N_y denotes pixel neighborhoods of x and y respectively. h is a filtering parameter and $\|.\|^2_{2,a}$ denotes squared Euclidean norm with Gaussian kernel standard deviation, $a > 0$. The weights are in the range $0 \leq w(x,y) \leq 1$ and should satisfy the condition $\sum_y w(x,y) = 1$ [A. Buades *et al*.,(2005)]. When $x == y$, the weight

value is replaced by maximum weights among the remaining pixels in order to overcome the overweighting of noisy pixel to itself [Raju *et al*., (2015)].

For a noisy image of size MxN, the initial step of NLM algorithm is to extract search window of a pixel. To perform similarity computation, a patch surrounding the pixel to be denoised (reference patch) and the remaining patches within a search window (current patches) are extracted, weights are computed and weighted average of pixels forms the denoised pixel value. This process of denoising noisy pixels is repeated MxN times to obtain the final denoised image. Since pixels are denoised sequentially, as the size of the image increases, the entire process becomes time consuming. Figure 1 depicts the complex iterative nature of NLM algorithm.

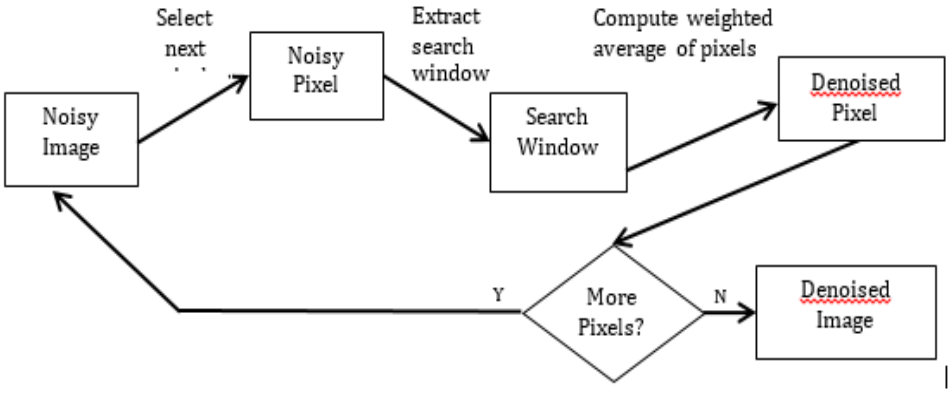

Fig. 1 Block diagram of sequential NLM algorithm

6. CUDA Implementation of NLM Algorithm

As discussed in previous section, self-similarity concept is the backbone of NLM algorithm. From the implementation point of view, to denoise a particular pixel, its neighborhood similarity to other pixel neighborhoods within a search window is considered. The search window is extracted such that the pixel to be denoised is at its center [A. Buades *et al*., (2005)]. In order to denoise border pixels in an image, symmetric padding is carried out, if required.

In serial implementation, search window acts as a sliding window which slides after denoising each pixel. For each pixel to be denoised, initially a search window with reference patch at its center is considered. The current patches are then extracted and for each sliding current patch, similarity with reference patch is found and weight is computed. Finally, weighted average of all pixels in the search window is obtained as the denoised pixel value. This procedure is repeated serially for all pixels in the noisy image. Hence, the time required to denoise a noisy pixel is directly proportional to the size of an image [A. Buades *et al*., (2005)].

In this work, the goal is to replace serial processing of noisy pixels by incorporating the features of CUDA programming model. The CUDA kernel function is designed in such a way that it performs NLM computation to denoise a single noisy pixel. This is achieved by threads in a thread block located in the device memory. Each thread can read pixel value corresponding to its index either from global or shared memory. As denoising of a pixel requires all other pixel neighborhoods in its search wondow for NLM algorithm, the thread which read the noisy pixel needs to communicate with all threads associated with the pixels in the search window. For this situation, data access via shared memory is a better alternative than the same from global memory. In order to access pixels within a search window from shared memory, thread blocks are designed in such a way that it has access to all pixels in a search window. The index of the central thread in a thread block points to the pixel to be denoised and performs NLM compution while all the other threads remain idle.

The initial step is to create a large buffer which can store search windows of each pixel in the noisy image. This buffer is passed to CUDA function with grid dimension that depends on the size of the image and block dimension fixed to the size of the search window. For the proposed algorithm, the aim is to denoise 64x64 pixels in an image in parallel. For this purpose, search window buffer of these pixels along with its grid and block dimensions are to be passed to kernel function from host memory. Kernel function will be invoked by each thread from device memory. Hence, block dimension is set to 21x21, size of search window and grid dimension to 64x64, number of search windows processed in parallel. The CUDA function is designed in such a way that the kernel function can be called four times with different inputs. Therefore, the search window buffer of input noisy image is divided into search window blocks of 128x128 pixels which can be further divided into 64x64 as the four inputs for CUDA function. Each kernel call returns 64x64 denoised outputs which are returned to the host function. The four denoised components are merged to obtain a denoised block of size 128x128. This procedure is repeated for the entire image. Figure 2 depicts the block diagram of proposed algorithm.

7. Experimental Set up and Performance Evaluation Matrices

This work is implemented using MATLAB R2016a with CUDA 7.5. Generally, MATLAB implementation of a CUDA program contains a header file (.h), a cuda file (.cu) and a mex file (.cpp). Cuda file contains definition for grid and block dimensions, memory allocation for device variables to copy variables from host memory, kernel function call, and copying of results from kernel function back to host memory. nvcc is the NVIDIA CUDA compiler driver used to compile cuda file and in turn create object file (.obj) [Suh & Kim, (2013)].

Mex file contains a mex function whose parameters are the number of output variables, pointers to output variables, number of input variables and pointers to input variables. mxGetData() is used to read data from input/output pointer variables and mxCreateNumericMatrix() is used to define output pointer variables. Then, CUDA function is called. Mex compiler is used to compile mex file and link with previously created object file to obtain the complied machine code mexw64 which can be executed directly from MATLAB running under windows 64 bit setup [Suh & Kim, (2013)].

The resultant denoised images from serial and parallel NLM algorithm implementation are evaluated using performance evaluation metrics such as Peak Signal to Noise Ratio (PSNR), Structural Similarity Index (SSIM) and Feature Similarity Index (FSIM) measures. PSNR represents the ratio between maximum powers of a signal to the noise which degrades an image[Horé & Ziou, (2010)].

$$PSNR = 10\, log_{10}\left\{\frac{L^2}{MSE}\right\} \tag{6}$$

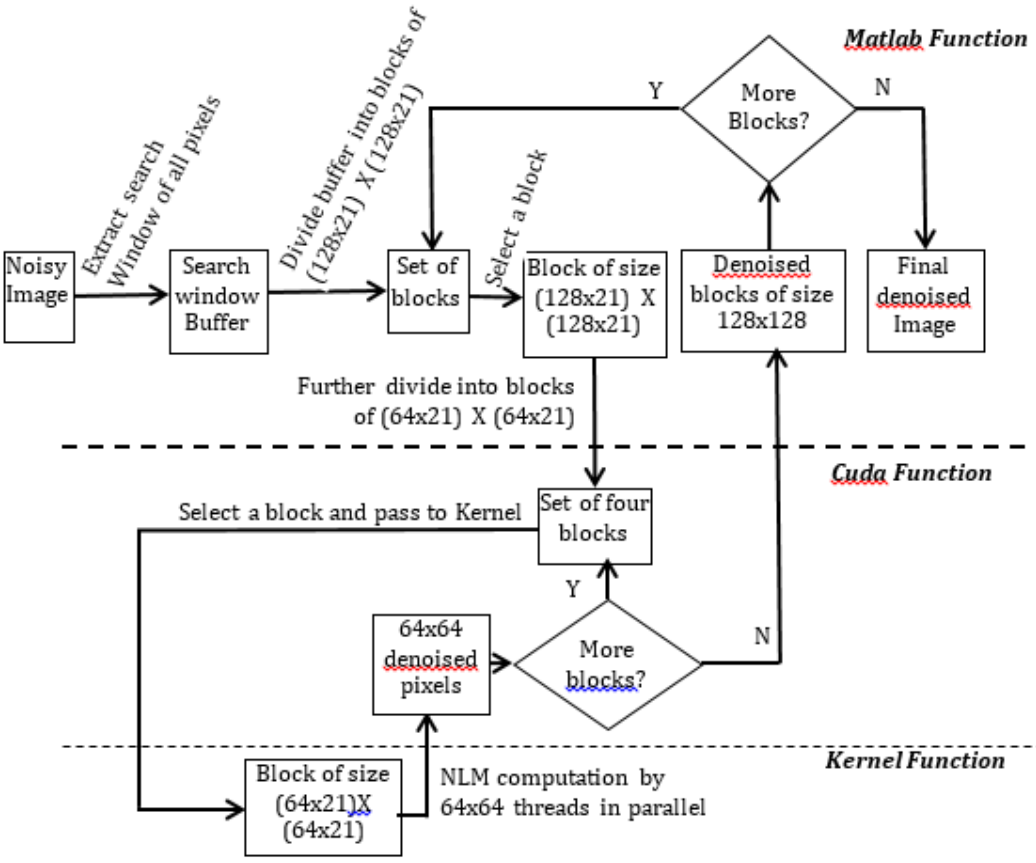

Fig. 2 Block Diagram of proposed algorithm

where

$$MSE = \frac{1}{MN}\sum_{i=0}^{M}\sum_{j=0}^{N}\left(u_{ij} - v_{ij}\right)^2 \tag{7}$$

Here, L is the maximum possible intensity of an image, u and v represents the original image and the denoised image.

SSIM is used to find the similarity between structures of two images. It is measured as

$$SSIM = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2\mu_y^2 + c_1)(\sigma_x^2\sigma_y^2 + c_2)} \tag{8}$$

where x and y are two windows of identical sizes, μ_x and μ_y are the average of x and y, σ_x^2 and σ_y^2 are the variance of x and y, σ_{xy} is the covariance. $c_1 = (k_1 L)^2$, $c_2 = (k_2 L)^2$, $k_1 \ll 1, k_2 \ll 1, L$ is the dynamic range of pixel values [Z. Wang *et al*., (2004)].

FSIM focuses on human visual system which understands an image according to its low level features. The primary feature in FSIM is phase congruency [Zhang *et al*., (2011)].

Each image under consideration is denoised n times using the algorithm and the elapsed time for each execution is recorded and its average is computed. This average value denotes the execution time for a particular image. Similarly, the Cumulative Average Time (CAT) is computed for selected set of images of a particular size. This process is repeated for all the available image sizes. CAT value can be computed as follows.

$$CAT = \frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n} T_{ij} \tag{9}$$

where T_{ij} is the elapsed time of i^{th} image in j^{th} execution, m and n indicates the number of images in a particular size and the number of executions for each image respectively.

In order to assess the time improvement of proposed parallel NLM algorithm over basic NLM algorithm, speed up factor is computed. The speed up factor is calculated as follows.

$$Speed\ up\ factor = \frac{CAT\ value\ of\ basic\ NLM\ Algorithm}{CAT\ value\ of\ Proposed\ Parallel\ NLM\ Algorithm} \tag{10}$$

Basic NLM algorithm has a time complexity of $O(N^2 \text{ x } S^2 \text{ x } P^2)$, where N^2, S^2 and P^2 indicate the total number of pixels in an image, search window size and patch size respectively. The time complexity of parallelized version of basic NLM algorithm is given as $O(K \text{ x } L \text{ x } S^2 \text{ x } P^2) + O(K \text{ x } Q^2 \text{ x } (S^2 + 1))$where $O(K \text{ x } Q^2 \text{ x } (S^2 + 1))$ is the data transfer time. Here, K indicates the total number of search window blocks of 128x128 pixels in an image of size N^2, $Q = 128$ is the size of image block denoised at a time, $L = 4$, is the total number of kernel calls for denoising a 128x128 image block. The proposed parallelized algorithm becomes effective when $K \text{ x } L \ll N^2$.

8. Results and Discussions

The denoising performance analysis of basic NLM algorithm and the proposed parallelized version of NLM algorithm are carried out with a set of natural images obtained from University of Southern California- Signal and Image Processing Institute (USC-SIPI) image database. It is a collection of digitized images which is divided into volumes based on the basic character of the images such as textures, aerials, etc. Images in each volume are of various sizes as 256x256, 512x512 and 1024x1024 pixels [*SIPI Image Database*, n.d.]. Also, natural images from a standard image database with a collection of images used by famous authors like Gonzales, Woods and Eddins in their books on image processing is considered. There are standard test images, light microscopy images, NASA planetary images, etc. in the database [*Image Databases*, n.d.]. The algorithm is also tested on medical images from Simulated Brain Database (SBD), also known as BrainWeb. It is a collection of set of Magnetic Resonance Imaging (MRI) images to evaluate the performance of vatrious analysis methods. Slice thickness, noise, intensity non-uniformity and modality are the various parameters that is to be specified according to our application [*BrainWeb: Simulated Brain Database*, (2019)].

For the experimental analysis of the current study, a total of 30 natural images from SIPI database and the standard image database are selected. Also, 30 slices of MRI images from BrainWeb database with T1 modality, 3mm slice thickness, 0% noise and 20% intensity non-uniformity are considered.

The images chosen for the experiments originally contains negligible noise. In order to evaluate the efficacy of the proposed algoirhm, noise is explicitly introduced according to application requirement. Hence, the test images are corrupted with additive white Gaussian noise (AWGN) with different noise standard deviations. For the implementation of NLM algorithm, a search window of size 21x21 and a patch of size 7x7 are selected. The filtering parameter h is chosen based on trial and error scheme for each noise standard deviations (σ). Same paramters are used for both sequential and parallel versions.

As this work is implemented to reduce the time complexity of basic NLM algorithm, experiments are carried out with images of four different dimensions - 128x128, 256x256, 512x512 and 1024x1024. In order to validate the results of both serial and parallel implementation, images corrupted with AWGN of various noise standard deviations such as 10, 20, 30 and 40 are executed independently on Geforce 940M GPU enabled high performance computing system. The efficacies of the denoised results are analysed quantitatively using PSNR [Horé & Ziou, (2010)], SSIM [Z. Wang et al., (2004)] and FSIM [Zhang et al., (2011)] performance evaluation measures.

Table 1 and Table 2 gives the denoised results of NLM algorithm obtained for five randomly selected natural images from SIPI database/a standard image database for the serial and parallel implementation respectively. From these tables, it is clear that results based on the perfromance evaluation measures nearly remain the same. This proves that the proposed parallelized version of NLM algorithm does not alter the basic NLM working

principle. The slight variation in the results is due to the fact that the addition of noise is done independently for both serial and parallel implentation.

The denoising results of NLM algorithm using five MRI medical image slices from BrainWeb database for serial and parallel implementations are given in Table 3 and Table 4 respectively. Here also, based on the performance evaluation measures, the results for both the versions nearly remain the same, thereby validating the proposed algorithm.

Table 1 Basic NLM Algorithm- denoising results of natural images corrupted with additive white Gaussian noise with standard deviation (σ = 10)

Sl #	Image	128x128			256x256			512x512			1024x1024		
		PSNR	SSIM	FSIM	PSNR	SSIM	FSIM	PSNR	SSIM	FSIM	PSNR	SSIM	FSIM
1	Barbara	29.9	0.913	0.946	29.5	0.888	0.949	33.0	0.966	0.979	31.8	0.985	0.991
2	Lena	31.8	0.918	0.950	32.1	0.890	0.949	33.7	0.954	0.975	35.5	0.981	0.989
3	Man	30.6	0.906	0.949	30.1	0.869	0.949	31.4	0.950	0.974	34.4	0.980	0.990
4	Cat	31.0	0.882	0.9367	31.7	0.867	0.940	32.5	0.953	0.971	35.0	0.983	0.989
5	Mandril	29.8	0.881	0.929	28.1	0.886	0.940	28.9	0.952	0.971	32.7	0.984	0.991

Table 2 Proposed Parallelized NLM Algorithm- Denoising results of natural images corrupted with additive white Gaussian noise with standard deviation (σ = 10)

Sl #	Image	128x128			256x256			512x512			1024x1024		
		PSNR	SSIM	FSIM	PSNR	SSIM	FSIM	PSNR	SSIM	FSIM	PSNR	SSIM	FSIM
1	Barbara	30.4	0.914	0.949	30.3	0.894	0.944	32.9	0.965	0.979	31.8	0.985	0.991
2	Lena	31.6	0.919	0.953	32.2	0.889	0.950	33.7	0.954	0.976	35.3	0.981	0.990
3	Man	30.6	0.991	0.949	30.3	0.878	0.946	31.9	0.953	0.976	34.2	0.981	0.991
4	Cat	31.7	0.888	0.945	31.2	0.871	0.936	32.7	0.955	0.973	34.8	0.983	0.990
5	Mandril	30.2	0.886	0.935	28.5	0.894	0.943	29.8	0.957	0.975	32.6	0.985	0.991

Table 3 Basic NLM Algorithm - Denoising results of medical images corrupted with additive white Gaussian noise with standard deviation (σ = 10)

Sl #	Image	128x128			256x256			512x512			1024x1024		
		PSNR	SSIM	FSIM	PSNR	SSIM	FSIM	PSNR	SSIM	FSIM	PSNR	SSIM	FSIM
1	Slice 1	35.4	0.869	0.931	36.7	0.883	0.932	38.3	0.949	0.968	39.6	0.981	0.988
2	Slice 2	35.3	0.873	0.926	36.6	0.880	0.931	38.3	0.950	0.967	39.5	0.980	0.988
3	Slice 3	35.5	0.878	0.934	36.7	0.884	0.933	38.2	0.949	0.967	39.5	0.981	0.988
4	Slice 4	35.4	0.876	0.932	36.6	0.884	0.931	38.2	0.949	0.967	39.5	0.981	0.988
5	Slice 5	35.3	0.872	0.929	36.6	0.883	0.932	38.2	0.947	0.967	39.4	0.980	0.988

Table 4 Proposed Parallelized NLM Algorithm - Denoising results of medical images corrupted with additive white Gaussian noise with standard deviation (σ = 10)

Sl #	Image	128x128			256x256			512x512			1024x1024		
		PSNR	SSIM	FSIM	PSNR	SSIM	FSIM	PSNR	SSIM	FSIM	PSNR	SSIM	FSIM
1	Slice 1	35.4	0.875	0.932	36.6	0.881	0.932	38.3	0.949	0.968	39.5	0.981	0.988
2	Slice 2	35.5	0.877	0.931	36.7	0.885	0.932	38.3	0.949	0.967	39.6	0.982	0.986
3	Slice 3	35.5	0.878	0.933	36.7	0.883	0.933	38.2	0.947	0.966	39.5	0.981	0.988
4	Slice 4	35.5	0.879	0.933	36.7	0.883	0.932	38.3	0.950	0.967	39.5	0.981	0.988
5	Slice 5	35.3	0.875	0.930	36.6	0.882	0.932	38.2	0.947	0.967	39.4	0.980	0.988

Fig. 3 and Fig. 4 depicts the denoising results of both versions of NLM algorithm on sample natural and medical image respectively.

For computing the CAT value, each image is executed 5 times. Hence, a total of 300 (*60 images* x *5 times*) executions are carried out. Table 5 gives the CAT value obtained for NLM algorithm and its proposed parallel version. The overhead of data transfer time is also included in the CAT value computation of parallelized version of the NLM algorithm.

Fig. 3 Denoising results of both versions of NLM algorithm on *Lena Image* : (a) Original Image (b) Noisy Image (c) Serial NLM (d) Parallel NLM

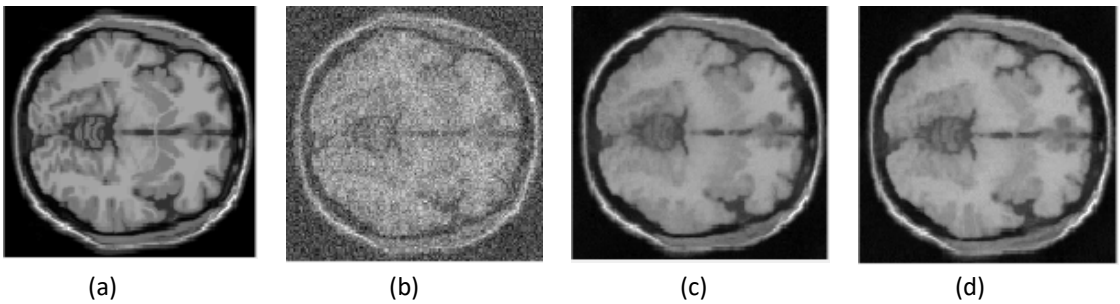

Fig. 4 Denoising results of both versions of NLM algorithm on *Slice 1 Brain Image* : (a) Original Image (b) Noisy Image (c) Serial NLM (d) Parallel NLM

Table 5 CAT value obtained for NLM algorithm and its proposed parallel version

Images	Algorithms	CAT Values (in sec.)			
		128x128	256x256	512x512	1024x1024
Natural Images	NLM	14.03	56.46	238.17	989.17
	Parallelized NLM	6.35	25.90	116.75	462.23
Medical Images	NLM	14.82	61.57	249.17	971.20
	Parallelized NLM	6.10	26.77	118.65	451.72

Based on the CAT values given in Table 5, the speed up factors for natural and medical images of various sizes is computed and is shown in Table 6 and Table 7 respectively.

Table 6 Speed up factor for natural images of different sizes

Sl#	Image size	Speed up factor
1	128x128	2.21x
2	256x256	2.18x
3	512x512	2.04x
4	1024x1024	2.14x

Table 7 Speed up factor for medical images of different sizes

Sl#	Image size	Speed up factor
1	128x128	2.43x
2	256x256	2.30x
3	512x512	2.10x
4	1024x1024	2.15x

Based on speed up factor, it is clear that for each image size, performance of proposed parallelized NLM algorithm is at least twice as fast as basic NLM algorithm for both natural and medical images. This time improvement is achieved without any alteration on the results obtained during the quantitative analysis using PSNR, SSIM and FSIM measures.

9. Conclusion

In this paper, a parallelized version of basic NLM algorithm using CUDA architecture is proposed. The algorithm is designed such that 64x64 pixels are denoised in parallel. Search window of 128x128 pixels are passed to CUDA by dividing it into four equal parts. Kernel function is invoked four times sequentially by each part. In order to assess the denoising performance of the proposed parallelized version of NLM algorithm, basic NLM algorithm is also implemented. The quantitative analysis using different measures have shown that the denoising performance of parallel version is same as basic NLM algorithm which substantiates the correctness of the proposed parallelized version of NLM algorithm. Based on speed up factor, it is evident that the proposed algorithm is twice as fast as basic NLM algorithm for all images with various sizes. The case study can be extended to any GPU processor architecture and the new functions available may improve the speed up factor.

REFERENCES

[1] Al-Ayyoub, M., Abu-Dalo, A. M., Jararweh, Y., Jarrah, M., & Al Sa'D, M. (2015). A GPU-based implementations of the fuzzy C-means algorithms for medical image segmentation. Journal of Supercomputing. https://doi.org/10.1007/s11227-015-1431-y
[2] Bahri, H., Sayadi, F., Khemiri, R., Chouchene, M., & Atri, M. (2017). Image feature extraction algorithm based on CUDA architecture: case study GFD and GCFD. IET Computers & Digital Techniques. https://doi.org/10.1049/iet-cdt.2016.0135
[3] BrainWeb: Simulated Brain Database. (2019). https://brainweb.bic.mni.mcgill.ca/
[4] Buades, A., Coll, B., & Morel, J. M. (2005). A Review of Image Denoising Algorithms, with a New One. Multiscale Modeling & Simulation. https://doi.org/10.1137/040616024
[5] Buades, Antoni, Coll, B., & Morel, J. M. (2005). A non-local algorithm for image denoising. Proceedings - 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2005. https://doi.org/10.1109/CVPR.2005.38
[6] Chakrabarti, G., Grover, V., Aarts, B., Kong, X., Kudlur, M., Lin, Y., Marathe, J., Murphy, M., & Wang, J. Z. (2012). CUDA: Compiling and optimizing for a GPU platform. Procedia Computer Science. https://doi.org/10.1016/j.procs.2012.04.209
[7] Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaffer, J. W., & Skadron, K. (2008). A performance study of general-purpose applications on graphics processors using CUDA. Journal of Parallel and Distributed Computing. https://doi.org/10.1016/j.jpdc.2008.05.014
[8] Contributors, W. (2019). Maxwell (microarchitecture). Wikipedia, The Free Encyclopedia.
[9] cuda_tutorial. (2018). http://geco.mines.edu/tesla/cuda_tutorial_mio/
[10] cuda-c-basics. (2018). https://www.nvidia.com/docs/IO/116711/sc11-cuda-c-basics.pdf
[11] CUDA. (2019). https://en.wikipedia.org/wiki/CUDA
[12] CUDA C programming Guide. (2019). http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf

[13] Farber, R. (2011). CUDA Application Design and Development. In CUDA Application Design and Development. https://doi.org/10.1016/C2010-0-69090-0
[14] Ferreiro, A. M., García, J. A., López-Salas, J. G., & Vázquez, C. (2013). An efficient implementation of parallel simulated annealing algorithm in GPUs. Journal of Global Optimization. https://doi.org/10.1007/s10898-012-9979-z
[15] GeForce_GTX_680. (2018). http://la.nvidia.com/content/PDF/ prod uct-specifications/GeForce_GTX_680_ Whitepaper_ FINAL.pdf.
[16] GeForce-GTX-750-Ti. (2018). https://international.download.nvidia.com/geforce-com/international/pdfs/GeForce-GTX-750-Ti-Whitepaper.pdf.
[17] GeForce _GTX_980. (2018). https://www.techpowerup.com/gpu-specs/docs/nvidia-gtx-980.pdf
[18] Gonzalez, R. C., & Woods, R. E. (2006). Digital Image Processing (3rd Edition). In Prentice-Hall, Inc. Upper Saddle River, NJ, USA ©2006. https://doi.org/10.1117/1.3115362
[19] Gulo, C. A. S. J., de Arruda, H. F., de Araujo, A. F., Sementille, A. C., & Tavares, J. M. R. S. (2016). Efficient parallelization on GPU of an image smoothing method based on a variational model. Journal of Real-Time Image Processing. https://doi.org/10.1007/s11554-016-0623-x
[20] He, B., Tang, L., Xie, J., Wang, X., & Song, A. (2015). Parallel numerical simulations of three-dimensional electromagnetic radiation with MPI-CUDA paradigms. Mathematical Problems in Engineering. https://doi.org/10.1155/2015/823426
[21] Hernández, D. E., Olague, G., Hernández, B., & Clemente, E. (2017). CUDA-based parallelization of a bio-inspired model for fast object classification. Neural Computing and Applications. https://doi.org/10.1007/s00521-017-2873-3
[22] Horé, A., & Ziou, D. (2010). Image quality metrics: PSNR vs. SSIM. Proceedings - International Conference on Pattern Recognition. https://doi.org/10.1109/ICPR.2010.579
[23] Image Databases. (n.d.). Retrieved May 9, 2019, from http://imageprocessingplace.com/root_files_V3/image_databases.htm
[24] Kadah, Y. M., Abd-Elmoniem, K. Z., & Farag, A. A. (2011). Parallel computation in medical imaging applications. International Journal of Biomedical Imaging. https://doi.org/10.1155/2011/840181
[25] Kang, D. K., Kim, C. W., & Yang, H. I. (2014). GPU-based parallel computation for structural dynamic response analysis with CUDA. Journal of Mechanical Science and Technology. https://doi.org/10.1007/s12206-014-0928-2
[26] Nickolls, J., Buck, I., Garland, M., & Skadron, K. (2016). Scalable parallel programming. 2008 IEEE Hot Chips 20 Symposium, HCS 2008. https://doi.org/10.1109/HOTCHIPS.2008.7476525
[27] NVIDIA Fermi. (2019). https://www.nvidia.com/ cont ent/PDF/fermi_white_papers/NVIDIA_ Fermi _Compute_Architecture_Whitepaper.pdf
[28] NVIDIA GeForce 940M. (2019).
[29] NVIDIA Quadro. (2019). http://www.nvidia.in/ content/PDF/data-sheet/ DS _NV_ Quadro_ K600_ OCT13_ NV_US_lr.pdf
[30] Park, S., Lee, J., Lee, H., Shin, J., Seo, J., Lee, K. H., Shin, Y. G., & Kim, B. (2014). Parallelized seeded region growing using CUDA. Computational and Mathematical Methods in Medicine. https://doi.org/10.1155/2014/856453
[31] Potluri, S., Fasih, A., Vutukuru, L. K., MacHot, F. Al, & Kyamakya, K. (2011). CNN based high performance computing for real time image processing on GPU. Studies in Computational Intelligence. https://doi.org/10.1007/978-3-642-24806-1_20
[32] Raju, G., Wahid, F. F., & Shareekhath, K. P. (2015). Modified non-local means filtering. 2015 IEEE International Conference on Signal Processing, Informatics, Communication and Energy Systems, SPICES 2015. https://doi.org/10.1109/SPICES.2015.7091552
[33] Sanders, J., & Kandrot, E. (2011). Cuda By Example. In Review Literature And Arts Of The Americas. https://doi.org/10.1073/pnas.1010880108/-/DCSupplemental.www.pnas.org/cgi/
[34] SIPI Image Database. (n.d.). Retrieved May 9, 2019, from http://sipi.usc.edu/database/
[35] Su, H., Wen, M., Wu, N., Ren, J., & Zhang, C. (2014). Efficient parallel video processing techniques on GPU: From framework to implementation. The Scientific World Journal. https://doi.org/10.1155/2014/716020
[36] Suh, J. W., & Kim, Y. (2013). Accelerating MATLAB with GPU Computing: A Primer with Examples. In Accelerating MATLAB with GPU Computing: A Primer with Examples. https://doi.org/10.1016/C2012-0-06517-9
[37] Upadhya, A. H. K., Talawar, B., & Rajan, J. (2016). GPU implementation of non-local maximum likelihood estimation method for denoising magnetic resonance images. Journal of Real-Time Image Processing, 13(1), 181–192.
[38] Wang, J., Ma, X., Zhu, Y., & Sun, J. (2014). Efficient parallel implementation of active appearance model fitting algorithm on GPU. The Scientific World Journal. https://doi.org/10.1155/2014/528080
[39] Wang, L., Li, S., Zhang, G., Ma, Z., & Zhang, L. (2013). A GPU-based parallel procedure for nonlinear analysis of complex structures using a coupled FEM/DEM approach. Mathematical Problems in Engineering. https://doi.org/10.1155/2013/618980
[40] Wang, Y. K., & Huang, W. Bin. (2014). A CUDA-enabled parallel algorithm for accelerating retinex. Journal of Real-Time Image Processing. https://doi.org/10.1007/s11554-012-0301-6
[41] Wang, Z., Bovik, A. C., Sheikh, H. R., & Simoncelli, E. P. (2004). Image quality assessment: From error visibility to structural similarity. IEEE Transactions on Image Processing. https://doi.org/10.1109/TIP.2003.819861
[42] Wilt, N. (2013). Cuda handbook. In Climate Change 2013 - The Physical Science Basis. https://doi.org/10.1007/s13398-014-0173-7.2
[43] X, Z., S, M., W, C., & Z, W. (2016). Exploiting parallelism in the simulation of general purpose graphics processing unit program. Journal of Shanghai Jiaotong University (Science), 21(3), 280–288.
[44] Zhang, L., Zhang, L., Mou, X., & Zhang, D. (2011). FSIM: A feature similarity index for image quality assessment. IEEE Transactions on Image Processing. https://doi.org/10.1109/TIP.2011.2109730
[45] Zhiyi, Y., Yating, Z., & Yong, P. (2008). Parallel image processing based on CUDA. Proceedings - International Conference on Computer Science and Software Engineering, CSSE 2008. https://doi.org/10.1109/CSSE.2008.1448