A Minor Project Report on

MEDIAN FILTERS IN IMAGE PROCESSING

undergone at

DEPARTMENT OF INFORMATION TECHNOLOGY

under the guidance of

Dr. B. Neelima

Submitted by

NADEER ALI	ROHAN BABLI	SULAIMAN MUHAMMAD
17IT224	17IT233	17IT143

V Sem B.Tech(IT)

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

Department of Information Technology

National Institute of Technology Karnataka,Surathkal.

November 2019

DECLARATION BY THE STUDENTS

We, **Nadeer Ali, Rohan Babli, Sulaiman Muhammad,** hereby declare that the project entitled "**Median Filters in Image Processing**" was carried out by us during the **5th semester** term of the academic year 2019 – 2020. We declare that this is our original work and has been completed successfully according to the direction of our guide Dr **B. Neelima** *(National Institute of Technology, Karnataka)* and as per the specifications of NITK Surathkal.

Place: NITK Surathkal

Date: 6/11/2019

_____

(Nadeer)

_____

(Rohan)

_____

(Sulaiman)

ABSTRACT

Image filtering is one of the most important parts in the image-processing. A fast median filtering algorithm is proposed for signal smoothing in image processing. The median filter is the well-known method in image processing because it preserves edges effectively while reducing the impulsive noise. However, it has a problem that execution speed is slow because it uses a sorting algorithm to obtain the median value. So, fast median filtering algorithms have been discussed. It takes much more time to perform the convolution in image filtering on CPU since the computation demanding of image filtering is massive. Contrast to CPU, GPU may be a good way to accelerate the image filtering. CUDA(Compute Unified Device Architecture) is a parallel computing architecture developed by NVIDIA. CUDA is highly suited for general purpose programming on GPU which is a programming interface to use the parallel architecture for general purpose computing. Compared with the traditional method of image filtering which was performed on CPU, the implementation of image filtering on GPU has a speedup of approximately 10 times. GPU has great potential as high-performance co-processor.

TABLE OF CONTENTS

LIST OF FIGURES

INTRODUCTION

Median filtering is a nonlinear filter used to remove salt or pepper noise of an image or signal. Since the median filter is at strength for eliminating noise while preserving edges, it is a well-known algorithm in image processing. It is used in applications requiring signal smoothing at the pre-processing step such as video edge detection and object segmentation. Noise removal at the pre-processing is essential in order to recognize objects or acquire information in the image. For example, it is necessary to eliminate the impulsive noise in order to reduce error rate when recognizing a QRcode. Median filters are excellent at removing the impulsive noise or sudden noise. Thus, these algorithms have been actively studied to be used in high-level applications. The median filter is a very conceptually simple method using the median value of the list, but it is a nonlinear filter and has a fatal problem that the processing time is proportional to the filter size. Thus, an algorithm that quickly executes the sorting algorithm to obtain the median value is necessary.

CUDA Programming Model:

CUDA is a hardware and software co-processing architecture for parallel computing that enables NVIDIA GPUs to execute programs written with C, C++, Fortran, OpenCL and other languages. By CUDA, programs are divided into CPU part and GPU part. Firstly some programs are executed on the CPU, then the data is copied to GPU and the programs go on computing on the GPU, finally the results are copied back to CPU. GPU has much bigger ram delay, smaller cache and poorer branch prediction than CPU, besides memory of GPU is rather limited. If the program doesn't have high concurrency degree, the effect of optimization is not good. A compiler generates executable code for CUDA device. The CPU regards a CUDA device as a multi-core co-processor. The CUDA design does not have memory restrictions of GPU. Programmers can access all memory available using CUDA with no restriction though the access time varies for different types of memory. The CUDA model is a collection of threads running in parallel shown in Fig.1.

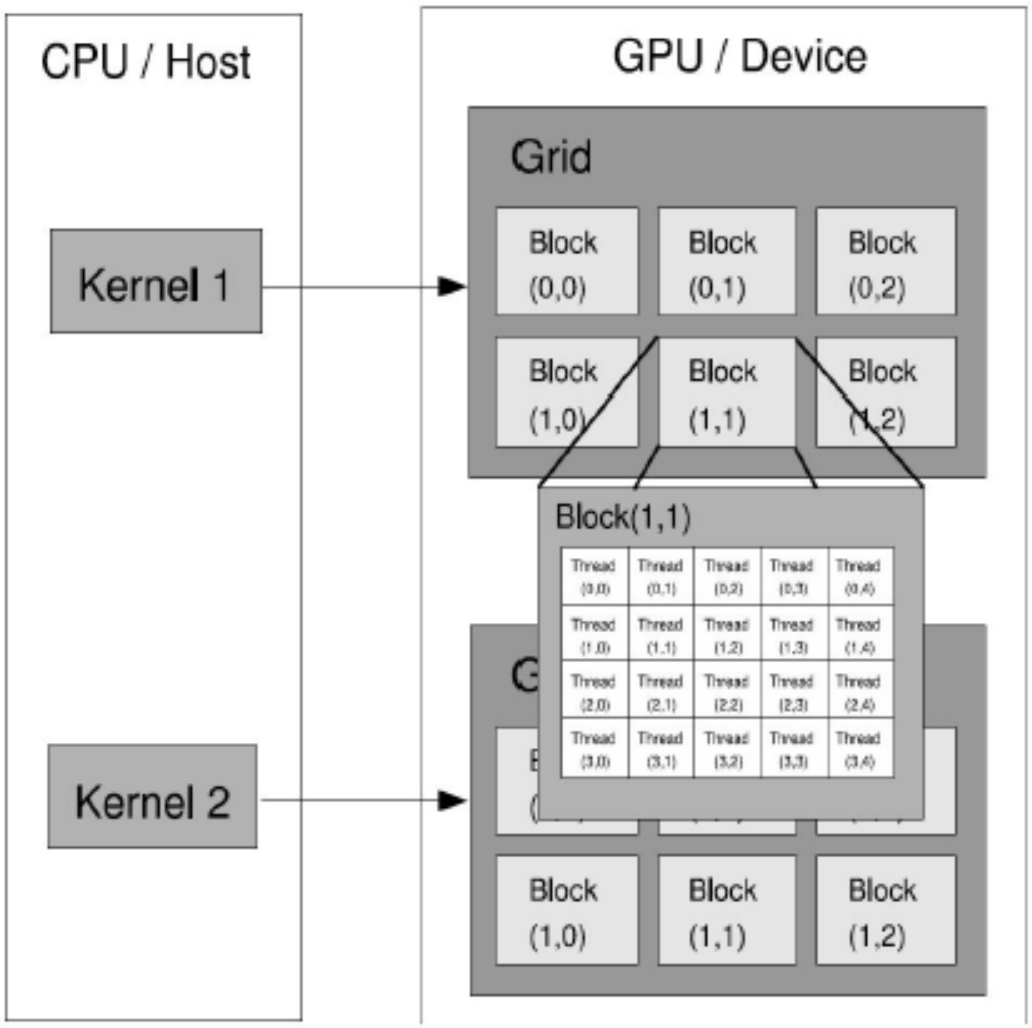


Fig.1. CUDA Programming Model

2. LITERATURE REVIEW

2.1 High Performance Median Filtering Algorithm Based on NVIDIA GPU Computing

AUTHORS: Placido Salvatore Battiato

This paper shows the effect of the Median Filtering and a comparison of the performance of the CPU and GPU in terms of response time. The Median filter is a nonlinear digital filtering technique often used to remove salt and pepper noise. In this paper, the first section explains the filtering problem and mathematical model. In CUDA section, they have presented the algorithm implemented using CUDA of median filter comparing the performance of CPU implementations of filtering effects, analysing one dimensional (1D) audio signal. The computational complexity (CC) defines the number of operations required for the extraction of the median from the window of N elements. The CC in an efficient sorting algorithm is estimated as $N*\ln(N)$. Instead in the counting algorithm CC is estimated as $N*N$ because each element mi of the window w is compared to all others and itself. Since N operations are executed in parallel in a GPU it could be said that the computational complexity is reduced to N. The simulation results demonstrate that it is possible to obtain gain in response time with an entry level GPU, allowing real-time image and audio filtering. However, the bottleneck of these systems is the PCI Express bus, for dedicated and direct bus through GPU/RAM
and CPU/ GPU the response time is reduced.

2.2 International Journal of Advanced Research in Computer Science and Software Engineering

AUTHORS: Ruchika Chandel, Gaurav Gupta

This paper mainly contains the five sections which describes the different algorithms and techniques. Section 1 describes the simple introduction about image filtering. Section II elaborates the mean filter for filtering the images. Then section III contains the various algorithms for image smoothing. In Section IV describes the various techniques for filtering the images. Section V contains the conclusion of this paper i.e. which algorithm is the best for removing the noise(filtering) from images. In this paper they have done the comparison between different image filtering algorithms. So, its concluded that median filtering approach is the best approach that can be easily implemented with the help of the image histograms. The median filter is demonstrably better than other algorithms at removing noise because it preserves edges for a given, fixed window size. So, median filtering is very widely used in digital image processing.

2.3 An Improved Median Filtering Algorithm for Image Noise Reduction

AUTHORS: Youlian Zhu, Cheng Huang

The paper proposed an improved median filtering algorithm for image noise reduction. It can adaptively resize the mask according to noise levels of the mask. Combined the median filtering with the average filtering, the improved algorithm can reduce the noise and retain the image details better. The statistical histogram is introduced to improve the searching speed of the median value and the correlation of image has been fully used. Thus, the complexity of the improved algorithm is decreased to O (N). Experimental results show that the improved algorithm can well do with the relationship between the effect of the noise reduction and the time complexity of the algorithm. Therefore, it has a good application prospect in image processing.

3. LACUNA IN THE EXISTING WORK

The following drawbacks were present in previously implemented median filtering algorithms:

- Majority of the computational effort and time is spent on calculating the median of each window
- Not efficient enough for larger signals such as images
- Since only the middle value in a list of numbers is required, selection algorithms could be much more efficient.
- More complex architecture which can improve computational efficiency has not been discussed

4. MOTIVATION

In the current generation of information technology where efficiency and usability take centre stage for any field of architectural design, it becomes transparent the need for optimization of any kind of useful implementation. With this mindset, it becomes evident that an algorithm that is used to filter out images, videos from the older generations, or even washed out photos becomes necessary for the current generation to fully realize their cultural and educational needs. The median filtering algorithm in an easy to use and understand algorithm which has not been fully realized to its maximum potential. In this report, steps are taken to maximize the optimization for this algorithm.

5. PROBLEM STATEMENT

The median filtering algorithm presents a simplistic way to drain out the salt and pepper in images, but the general naïve approach has significant overheads contributing to much computational costs and inefficient memory management. A schema is required where implementation of the algorithm can be established while incurring less overheads. Establishment of such a schema should involve architectures that aren't difficult to produce or procure and in essence, should be computationally and economically feasible.

6. METHODOLOGY

- General Approach

  The main idea of the median filter is to run through the signal entry by entry, replacing each entry with the median of neighbouring entries. The pattern of neighbours is called

the "window", which slides, entry by entry, over the entire signal. For 2D (or higher-dimensional) data the window must include all entries within a given radius or ellipsoidal-region

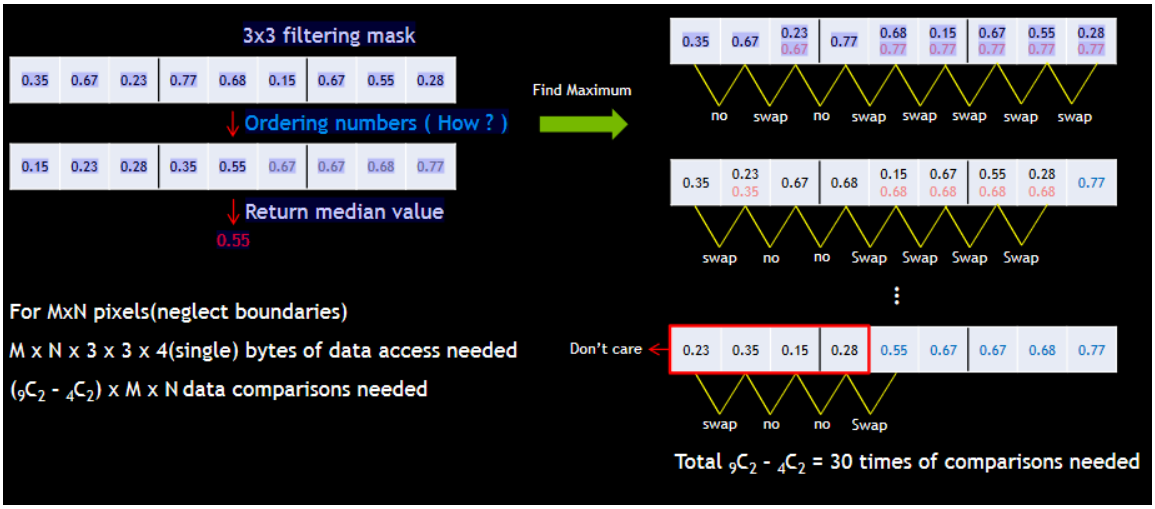


Fig2: Filtering idea

Fig3: Filtering result

- Parallelized CPU Approach

  We implement a parallelized approach following the CPU version by making use of multiple threads. This can be done by making use of OpenMP

- Naïve GPU Approach

  We implement the CUDA architecture to run the general algorithm to produce positive throughput. The program implemented in CPU is directly mapped to GPU without fully utilizing CUDA's capabilities and utilities.

- Further Optimization on Naïve GPU Approach

  The further optimization techniques could be utilized:

  - Reducing data access
  - Use the fastest memory as buffer: Use Shared memory or Register
  - Data Re-usability
  - Use a better sorting algorithm to reduce bottleneck issue
  - Reduce Latency

7. IMPLEMENTATION

A 3 x 3 median filtering algorithm is implemented to clean up a grayscale bitmap image. A Bitmap header file is used to perform load and save operations on bmp images and also to get and set pixel values of the image.

- General Approach Algorithm (CPU):

```
allocate outputPixelValue[image width][image height]
  allocate window[window width * window height]
  edgex := (window width / 2) rounded down
  edgey := (window height / 2) rounded down
  for x from edgex to image width - edgex
    for y from edgey to image height - edgey
      i = 0
      for fx from 0 to window width
        for fy from 0 to window height
          window[i] := inputPixelValue[x + fx - edgex][y + fy - edgey]
          i := i + 1
      sort entries in window[]
      outputPixelValue[x][y] := window[window width * window height / 2]
```

- Parallelized CPU Implementation (OpenMP)

We try to effectively parallelize the above implementation by letting different threads run the filtering algorithm in different rows of the MxN picture grid.

For CPU implementation, median filtering is performed by capturing neighbouring pixels around the current pixel, i.e. 9 pixel elements

including the current pixel element. Median of these 9 elements is calculated by sorting them and picking the middle element, which is used to set value of the current pixel in output image. This operation is carried out on all pixels in the image. For border

conditions, missing neighbouring pixels of the current pixel are replicated with a pixel of value 0.

- Naïve GPU Implementation

Fig4: GPU pseudo code

```
GPU version pseudo code

__global__ void compare(Dest,Src,mask) {
  int x = blockIdx.x*blockDim.x+threadIdx.x
  int y = blockIdx.y*blockDim.y+threadIdx.y
  int m_sq = mask*mask;
  float *buf = new float[m_sq];
  //Assign masked values to buf
  for(i = 0; i < m_sq; i++) {
    buf[i] = Src[(y-1+i/mask)*width+x-1+i%mask];
  }
  //calculate until we get the median value
  for(k = 0; k < (m_sq+1)/2; k++)
    for(l = 0; l < m_sq; l++)
      if ( buf[k] < buf[l] )
        swap(buf[k],buf[l]);
}
// width by height pixels image, 3x3 median filter
int main(...) {
  ...
  compare<<<grid,block,...>>>(output,input,3);
  ...
```

For GPU implementation, a 2D block with tile width as 16 is chosen (i.e. 16 x 16 block) for optimal utilization of all threads in a stream multiprocessor. Dimensions of the input image are used to determine the size of 2D grid, i.e. total number of blocks. For naïve GPU implementation, global memory is utilized where all 9 elements required to perform 3 x 3 median filtering on each pixel are loaded from global memory, i.e. each pixel is accessed 9 times from global memory.

- Optimized GPU approach

For GPU implementation with shared memory, an 18 x 18 block of shared memory is used for each block in which all 16 x 16 pixel elements corresponding to the current block of the input image are copied from global memory, along with the border elements around the 16 x 16 tile. These outlier elements are used to calculate median for the bordering elements of the 16 x 16 tile. Neighbouring elements of each pixel are accessed from shared memory for all pixel elements in the block.

8. EXPERIMENTAL SETUP

- CUDA Version 10.1 using Visual Studio 2019 has been used for implementing the GPU approach
- For GPU implementation (both with global memory and with shared memory), a 2D block with tile width as 16 is chosen (i.e. 16 x 16 block) for optimal utilization of all threads in a stream multiprocessor. Dimensions of the input image are used to determine the size of 2D grid, i.e. total number of blocks.
- For GPU implementation with shared memory, an 18 x 18 block of shared memory is used for each block in which all 16 x 16 pixel elements corresponding to the current block of the input image are copied from global memory, along with the border elements around the 16 x 16 tile.
- OpenMP with g++ compiler was used to implement parallelized CPU version

9. OBSERVATIONS

Computation times of all three implementations are measured along computation and speedup of GPU implementations with global memory and with shared memory, against CPU implementation-both single and multiple threads implementations(OpenMP).

This program is tested on 3 images (Lenna.bmp, RIT.bmp, milkyway.bmp) of different sizes and the corresponding results are tabulated as below using 2 different sorting algorithms(Bubble and insertion sort). From the results, it can be observed that both GPU implementations give better performance than the CPU implementations and also from the tables shown below, it can be seen that speedup of both GPU implementations increases as the image size increases.

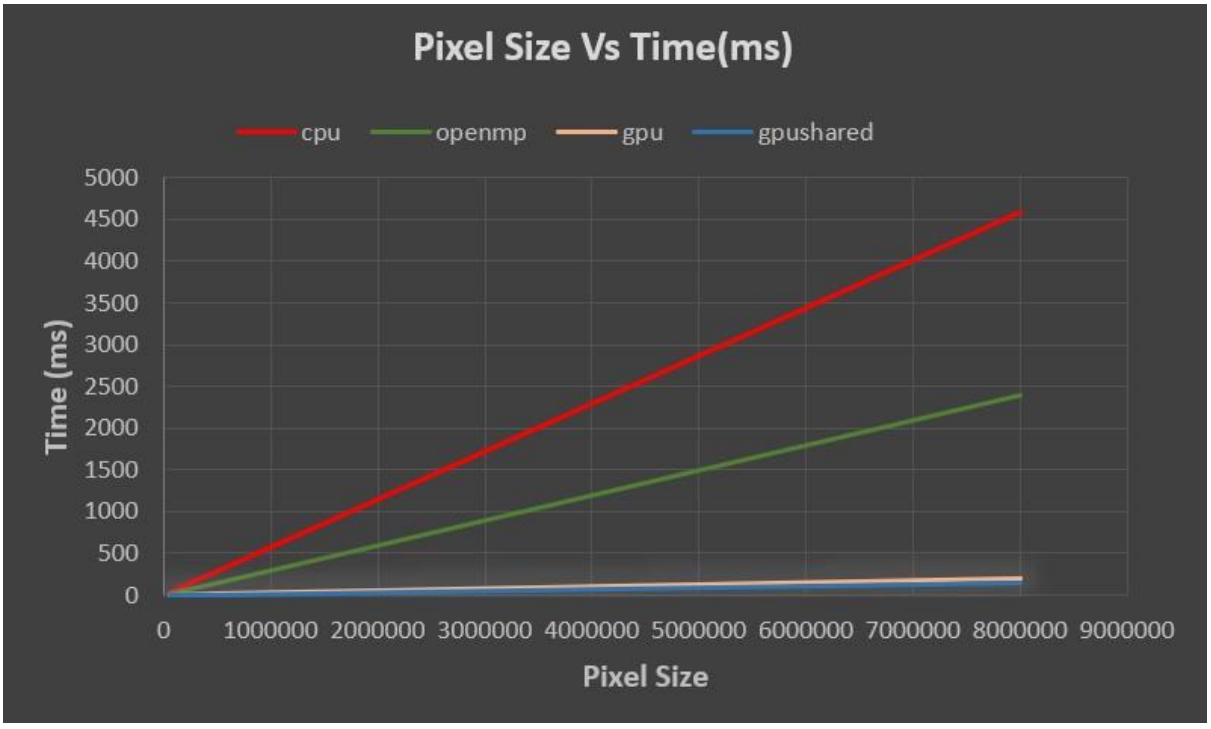
IMAGE	SIZE	CPU	OPENMP	GPU	GPU-SHARED	SPEEDUP-OPENMP	SPEEDUP-GPU	SPEEDUP-GPUSHARED
Lenna	252 X 205	30ms	14.54ms	2.11ms	1.93ms	2.06	14.21	15.54
RIT	946 X 532	291.14ms	144.4ms	13.51 ms	10.256ms	2.01	21.54	28.40
Milky way	4000 X 2000	4580.3ms	2390.48ms	196.4 ms	147.5ms	1.91	23.32	31.05

Table 1: Results Summary using bubble sort

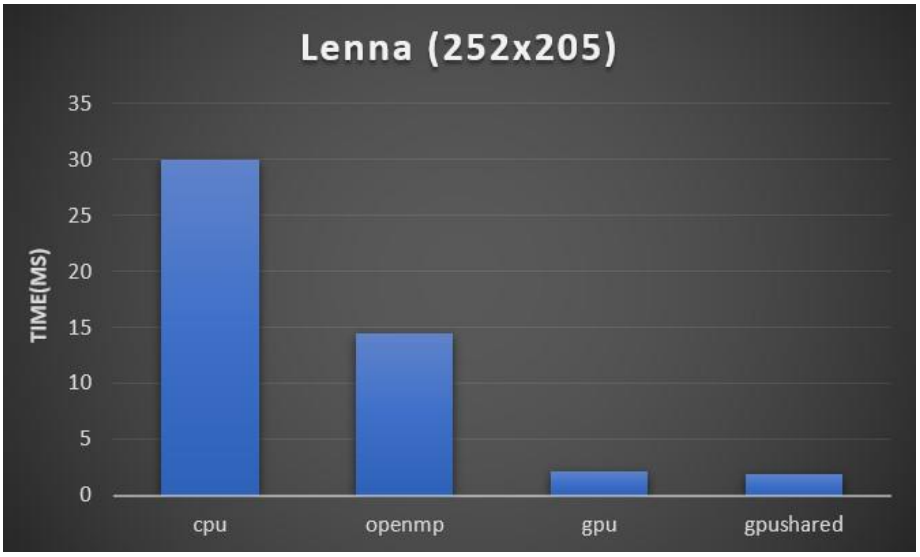
IMAGE	CPU	OPENMP	GPU	GPU-SHARED	SPEEDUP-OPENMP	SPEEDUP-GPU	SPEEDUP-GPUSHARED
LENNA	17.2ms	7.92ms	1.69ms	1.19ms	2.15	10.17	48.40
RIT	166.17ms	82.57ms	9.14ms	5.73ms	2.01	18.16	28.95
MILKY WAY	2844.24ms	1344.66ms	127.02ms	78.81ms	2.11	22.29	36.08

Table 2: Results summary with insertion sort

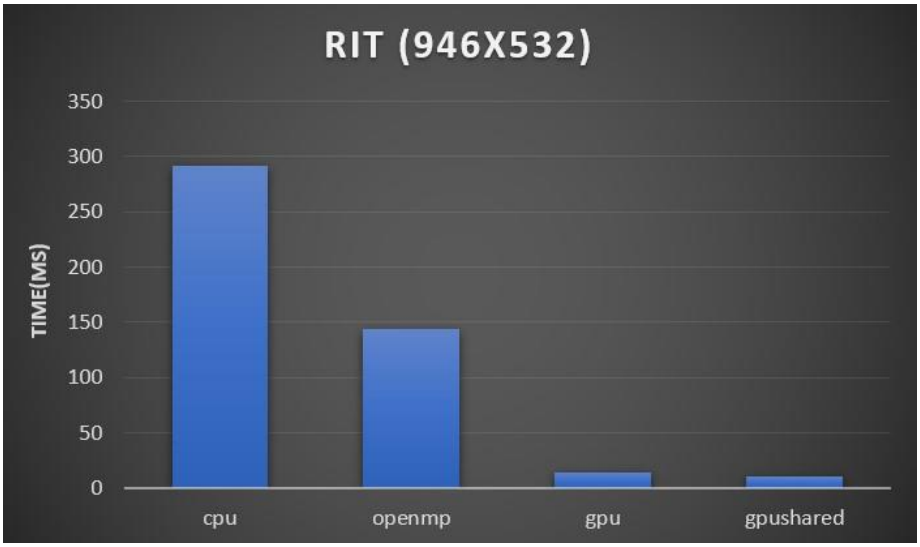
Graph1:Pixelsize vs Time(Bubble Sort)



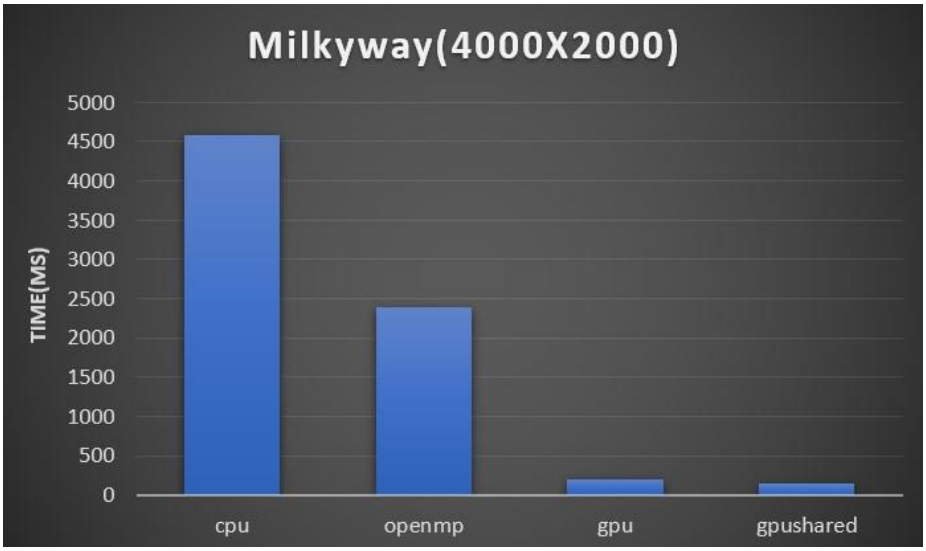
Graph2: Time for Lenna(bubble sort)



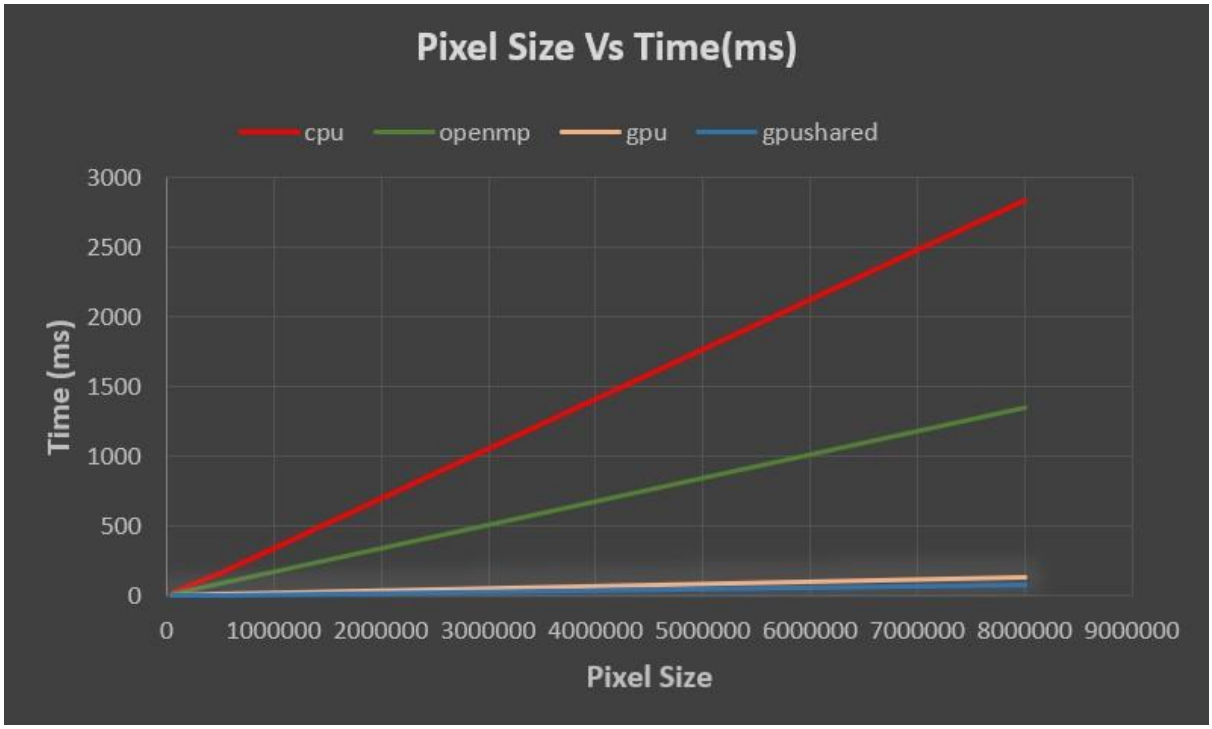
Graph3: Time for RIT(bubble sort)



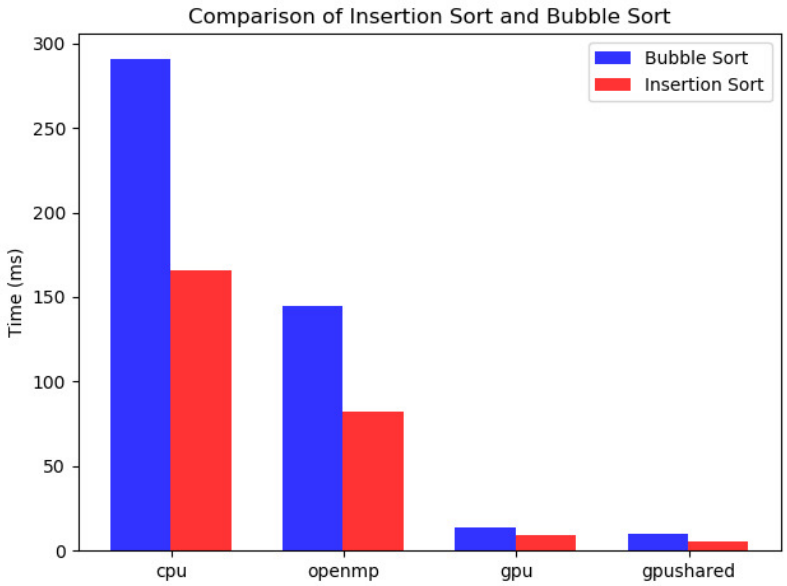
Graph4: Time for Milky Way(bubble sort)



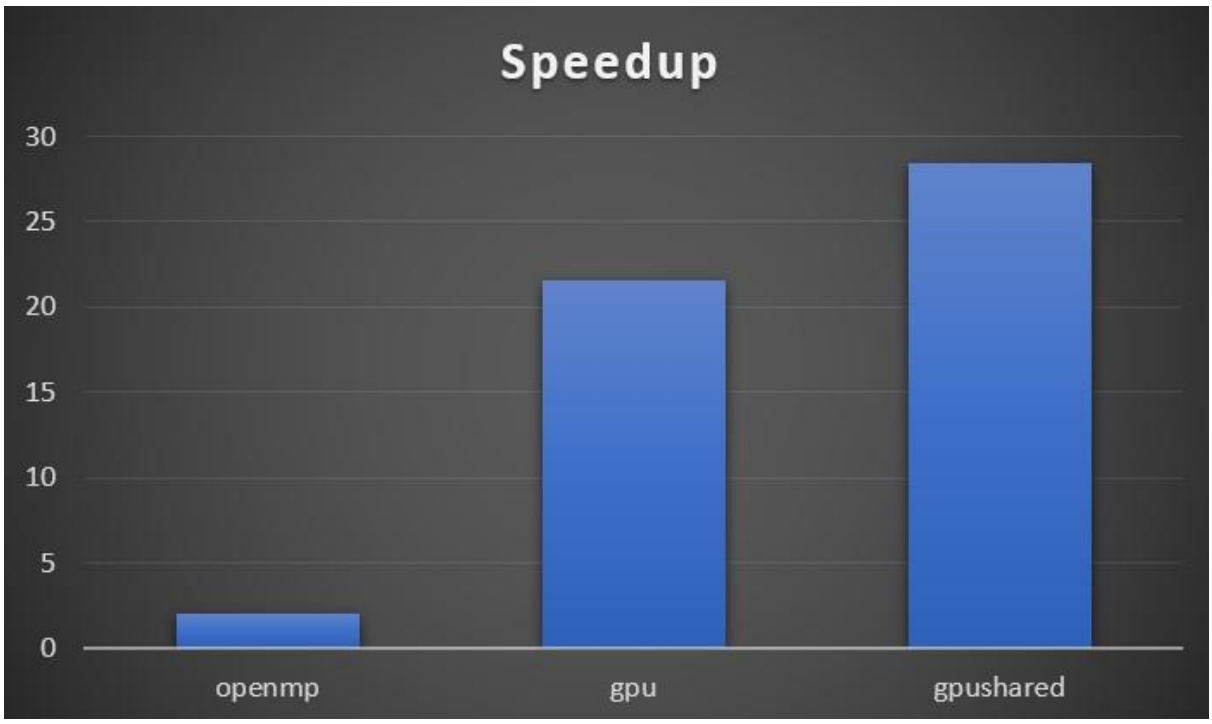
Graph5: Pixel size vs Time(insertion sort)



Graph6: Bubble and Insertion sort comparison



Graph7: Speedup comparison



10. RESULTS

The following results were obtained while testing the algorithm:

Fig5: Lenna original

Fig 6: Lenna after filtering

Fig7: RIT original

Fig8: RIT after filtering

Fig9: Milky Way original

Fig10: Milky way after filtering

- GPU implementations are seen to be computed 20 times faster than general CPU approaches
- OpenMP computes 2 times faster when compared to CPU implementation
- Speedup increases as size of the image increases
- Shared memory implementation of GPU computes 1.6 times faster when compared to global memory implementation
- Insertion sorting algorithm computes 1.5 times faster when compared to bubble sort

11. CHALLENGES FACED

The following challenges were faced while implementing the algorithm:

- Installing CUDA
- Learning and Implementing the GPU version in CUDA
- Getting appropriate hardware
- Figuring out Optimization techniques

12. CONCLUSION

The following conclusions were made after implementing and testing the algorithm:

- In many cases of GPU computation, accessing is most important part for optimal performance. Therefore memory reducing a GPU global memory access and using highest memory performance is the key to GPU code optimization

- Consequently, finding a optimal point between the memory operations and data accesses is the most important factor to get the best performance

13. REFERENCES

[1] High Performance Median Filtering Algorithm Based on NVIDIA GPU Computing-Placido Salvatore Battiato

[2] M. Juhola, J. Katajainen, and T. Raita, “Comparison of algorithms forstandard median filtering,”IEEE Transactions on Signal Processing,vol. 39, no. 1, pp. 204–208, Jan 1991

[3] C. Nvidia, “Nvidia cuda compute unified device architecture program-ming guide,”NVIDIA Corporation, 2007.

[4] K. Oistamo, Q . Liu, M. Grundstrom, and Y . Neuvo, "Weighted vector median operation for filtering multispectral data," in Systems Engineering, 1992., IEEE International Conference on, 1992, pp. 16-19

[5] Weiss B (2006) Fast median and bilateral filtering. In: ACM SIGGRAPH 2006 Papers, ACM, New York, NY, USA, SIGGRAPH ’06, pp 519–526

[6] Gökrem, L., andEngino, S. (2018). Different applied median filter in salt and pepper noise, pg 1–10.