International Conference on Cloud of Things and Wearable Technologies 2018 [ICCOTWT 2018]

ISBN	978-93-88122-00-9	**VOL**	01
Website	iccotwt.com	**eMail**	iccotwt@asdf.res.in
Received	14 – May– 2018	**Accepted**	02 - June – 2018
Article ID	ICCOTWT011	**eAID**	ICCOTWT.2018.011

Image Filters Using CUDA on NVIDIA

Tedros Berhane[1] and Namit Nagpal[2]

[1,2]School of Engineering and Computer Science, Oakland University, Rochester, Michigan, USA

Abstract— *This paper explains our team project on GPU Accelerated Parallel programing using CUDA and OpenCV open source available development tool and was completed at Oakland University, Rochester, MI Engineering labs. The project compared two commonly known as image filters, Median and Bilateral. The two filters were also compared with basic Gaussian filter, which has normal distribution, the team applied sorting algorithm methods in CUDA programming to perform parallel computing and optimization*

Keywords: Computer Vision (CV), OpenCV, CUDA, Median Filter, Bilateral Filter, Algorithm, Benchmarks, Shared Memory, Results.

INTRODUCTION

The document is final project report on GPU Accelerated Computing, in which two graduate students at Oakland University developed optimization and speed up results using CUDA and OpenCV. Parallel computing in this project has been achieved using CUDA programing and was perform in NVIDIA computing platforms environment. Parallel computing techniques recently has been practiced in the academia and industries to solve complex problems through thread and kernel launches. NVIDIA has Computer Vision (CV) open source and customized libraries where user can custom with their application from facial recognitions, contour of the physical image appearances, or smoothing blur image outcome [4].

To achieve this result, we accessed the wide-ranging of memory management, and parallelly optimization techniques of CUDA, the concepts and approaches of shared memory was used. In addition, thread scheduling by using tiling approaches and Kernel launches from CPU was enhanced as per performance results shows in next sections.

The image filtering was reached using Central Processing Unit (CPU) and Graphical Processing Unit (GPU) in OpenCV to compare and see the speed performance related to serial and parallel programming methods. In the program code, higher programming languages C/++ that are oftentimes compatible with CUDA and OpenCV were used to run the algorithms developed by the team. In this case, C++ was used in filtering images. Most importantly OpenCV Application Peripherals Interfaces (API) and struct class type of data organization approach was precisely practice in importing image matrix and outputting data results from OpenCV. Over all, the Massive parallel programming and computing procedure for median and bilateral filters differs on the pseudo code of the functions, as the result different blurring intensity has been displayed. The following four images portrays median and Bilateral filtering. Two figures below are sample of Median and Bilateral filter results ruined by the team.

Cite this article as: Tedros Berhane and Namit Nagpal. "Image Filters Using CUDA on NVIDIA". *International Conference on Cloud of Things and Wearable Technologies 2018*: 06-11. Print.

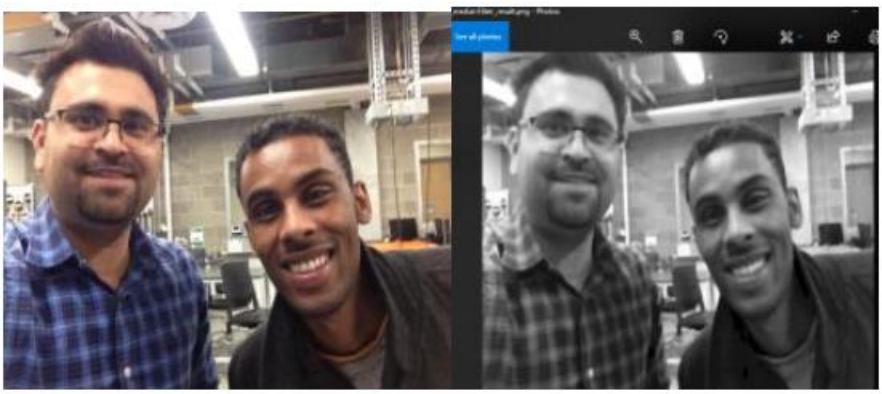


Figure 1: Median Filter

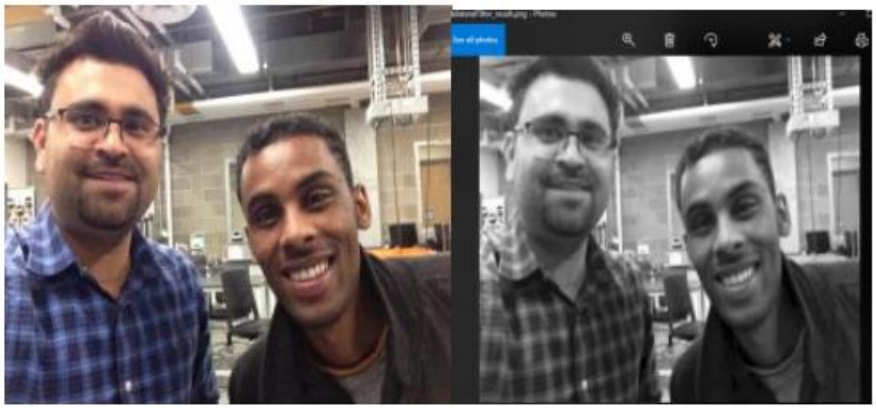


Figure 2: Bilateral Filter

COMPUTER VISION (CV)

Computer Vision (CV) is modern computing software that enables and helps in detection, classification, and recognition of objects by the body size, shapes, and appearances [6]. Nowadays, Computer vision are using heavily in autonomous or self-driving cars, where the it detects the appearances of stop lights, types of cars, and buildings. This computer software helps in guiding self-driving cars through image and video surveillances cameras. Computer visions can also detect and classify human faces and body parts. The team used Computer vision to blur two types of image filters.

OPENCV

OpenCV is open sources software supported by NVIDIA CUDA and has higher level functional Application Peripheral Interfaces (APIs), where the user or developer can customize inputs and outputs of their source's files [4]. Thereafter, the team inserted 2-Dimensional (2D) image through CV strands. OpenCV, can be programmed with C, C++, Python, and Java and has 250 built functions. When CUDA is used with OpenCV the memory models can give 5x-100x times speed faster.

CUDA

The final thing the team approached better optimization and enhance good image blurring product was implementing the sequential code at first hand and allied it with parallel code. Naïve approach of host code was compatible with OpenCV libraries and functions, but to reach better optimization and speed up tiling approaches and shared memory access of basic parallel massive programming method was utilized. The speed performances of the host and device has been displayed below in result section.

A. Median Filter

Median Filter is one type of many image or signal processing filters that commonly used in graphics to remove pixel intensity, while preserving the image edges. Pixel intensity removal is done by sorting instead of Gaussian normal distributions, the neighboring

Cite this article as: Tedros Berhane and Namit Nagpal. "Image Filters Using CUDA on NVIDIA". *International Conference on Cloud of Things and Wearable Technologies 2018*: 06-11. Print.

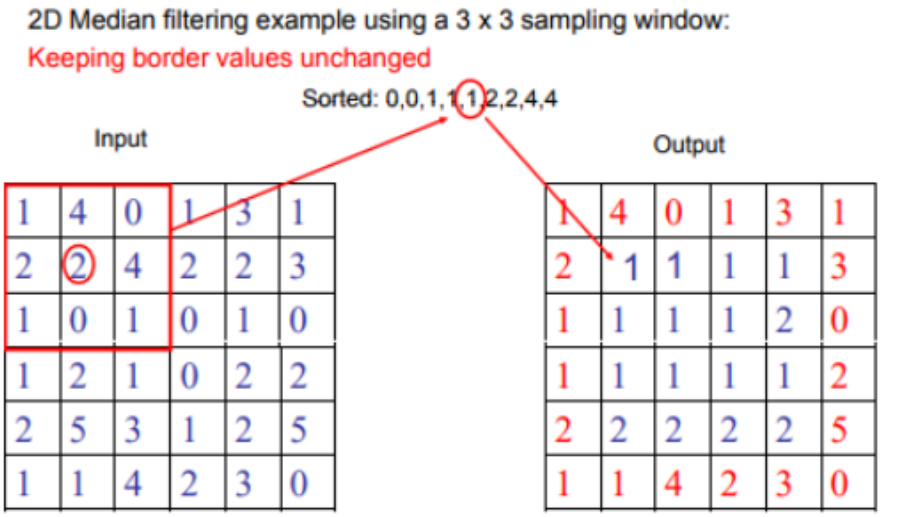
elements within sub matrix gets the median distribution and the median kernel is then replaces the element size [2]. As non-linear type of function, Median filter has cons that when sorting of median distribution is applied the average value doesn't sum up or make the whole pixel value. However, this gives better speed up performance when implemented in programming due to algorithm or logic nature of it.

B. Bilateral Filter

Bilateral is another type nonlinear image filtering model, where filter or blurring image is done by the weight average of intensity value from neighing elements, while preserving sharp edge of the pixel [3]. Bilateral has two filters, spatial, and local. Spatial filter measures the geometric closeness between the current point and nearby point, while Local filter measures the photometric similarity between the pixel value of the current point and nearby point [3]. Bilateral image blurring has higher latency due to the functional mode of the filter and the weighted average gives better distorting upshot.

C. Algorithm

Median filter algorithm used in the project by moving the image pixel by pixel, replacing each pixel with the median value of neighboring pixels. We used median of nine neighboring elements in the project. The pattern of neighbors is called the "window", which slides, pixel by pixel over the entire image pixel. The median is calculated by bubble sort all the pixel values from the window to numerical order, and then substituting the pixel actuality considered with the median pixel value for example.




```
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]
```

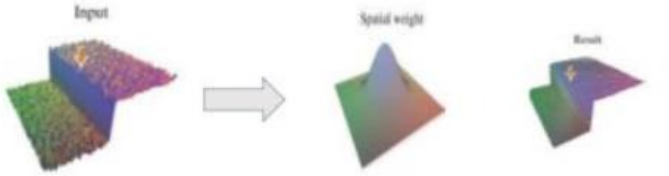
Bilateral filter algorithm used in the project by moving the image pixel by pixel the intensity of each pixel in the image is substituted by a weighted average of intensity values nine nearby pixels. This weight can be constructed on a Gaussian distribution for instance, the range variance like the color intensity.

Cite this article as: Tedros Berhane and Namit Nagpal. "Image Filters Using CUDA on NVIDIA". *International Conference on Cloud of Things and Wearable Technologies 2018*: 06-11. Print.

```
3. For each pixel p in the input_image
      p_value = 0;
      weight = 0;

      for each pixel q inside the range of spacial kernel
         p_value += Gaussian_spacial(|p-q|)*Gaussain_range(|Ip - Iq|)*Iq;
         weight += Gaussian_spacial(|p-q|)*Gaussain_range(|Ip - Iq|);
      end

      //normalize
      p_value /= weight;
```



D. Results

we tried using different type of sorting techniques but got the same result then for the final programming we use only bubble sort.

```
-bash-4.2$ ./build/filters_gpu
GPU Accelerated Median Filter took 1.11111 ms.
CPU Accelerated Median Filter took 37.7778 ms.          Linear sort
GPU Accelerated Bilateral Filter took 2.22222 ms.
CPU Accelerated Bilateral Filter took 114.444 ms.
-bash-4.2$ make
nvcc -I. -arch=sm_52 -c src/median_gpu.cu -o build/median_gpu.o
nvcc -I. -arch=sm_52 -c src/bilateral_gpu.cu -o build/bilateral_gpu.o
g++ -o build/filters_gpu src/main.cpp build/median_gpu.o build/bilateral_
 opencv` `pkg-config --libs opencv` -L/usr/local/cuda/lib64 -lcudart
-bash-4.2$ ./build/filters_gpu
GPU Accelerated Median Filter took 1.11111 ms.
CPU Accelerated Median Filter took 36.6667 ms.          Bubble sort
GPU Accelerated Bilateral Filter took 2.22222 ms.
CPU Accelerated Bilateral Filter took 114.444 ms.
```

Result with Image size 450 x 300

```
-bash-4.2$ ./build/FinalProject
GPU : Time taken by GPU Median Filter 2.22222 ms.
CPU:  Time taken by CPU Median Filter 10 ms.
GPU : Time taken by GPU Bilateral 2.22222 ms.
CPU : Time taken by CPU Bilateral 25.5556 ms.
```

Result with Image size 1450 x 1300

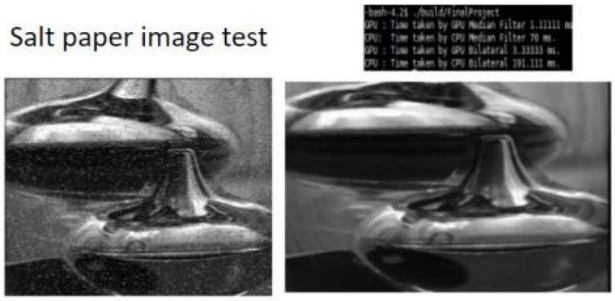
```
GPU : Time taken by GPU Median Filter 4.44444 ms.
CPU:  Time taken by CPU Median Filter 125.556 ms.
GPU : Time taken by GPU Bilateral 3.33333 ms.
CPU : Time taken by CPU Bilateral 357.778 ms.
```

Result with Image size 700 x 400

```
-bash-4.2$ ./build/FinalProject
GPU : Time taken by GPU Median Filter 2.22222 ms.
CPU:  Time taken by CPU Median Filter 20 ms.
GPU : Time taken by GPU Bilateral 1.11111 ms.
CPU : Time taken by CPU Bilateral 54.4444 ms.
```

Median filtering is a widely used image in improvement method for eliminating salt and pepper noise. Since this filtering is not as much of sensitive than linear techniques to dangerous changes in pixel values, it can eliminate salt and pepper noise without knowingly reducing the sharpness of an image.

Cite this article as: Tedros Berhane and Namit Nagpal. "Image Filters Using CUDA on NVIDIA". *International Conference on Cloud of Things and Wearable Technologies 2018*: 06-11. Print.

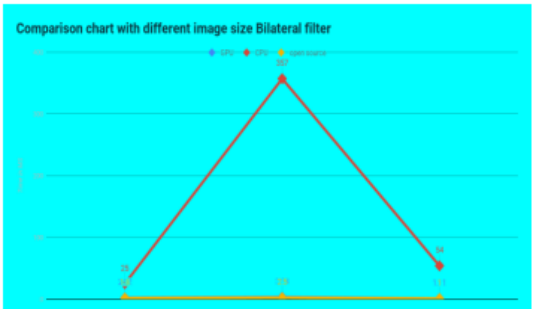


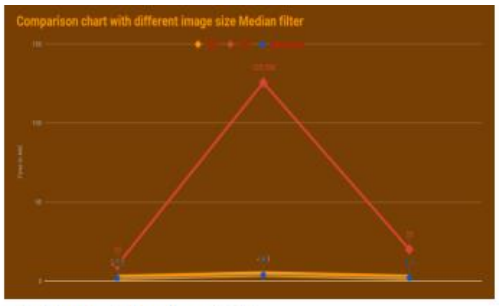
PROFILING RESULTS

Profiling results shows the usage of memory and performance of the application. It also tells us that which api used how much time and number of times it called.

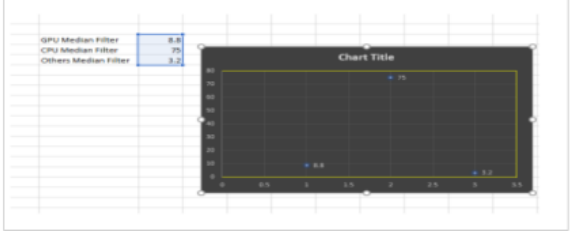
```
==9264== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 51.32%  5.7067ms        20  285.33us  285.12us  285.51us  [CUDA memcpy HtoD]
 48.68%  5.4128ms        20  270.64us  270.56us  270.69us  [CUDA memcpy DtoH]

==9264== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 95.52%  506.56ms        40  12.664ms  412.62us  487.73ms  cudaMalloc
  2.29%  12.164ms        40  304.09us  132.08us  1.1035ms  cudaMemcpy
  1.07%  5.6801ms        20  284.00us  259.57us  292.29us  cudaDeviceSynchronize
  0.94%  4.9942ms        40  124.85us  100.47us  177.78us  cudaFree
  0.08%  415.61us        91  4.5670us     341ns  160.71us  cuDeviceGetAttribute
  0.07%  378.18us         1  378.18us  378.18us  378.18us  cuDeviceTotalMem
  0.01%  41.628us        80     520ns     288ns  6.2120us  cudaSetupArgument
  0.01%  32.665us         1  32.665us  32.665us  32.665us  cuDeviceGetName
  0.00%  24.082us        20  1.2040us     870ns  3.7600us  cudaConfigureCall
  0.00%  22.857us        20  1.1420us     926ns  1.7480us  cudaLaunch
  0.00%  6.1370us         3  2.0450us     476ns  4.4310us  cuDeviceGetCount
  0.00%  2.3980us         3     799ns     499ns  1.3110us  cuDeviceGet
-bash-4.2$
```





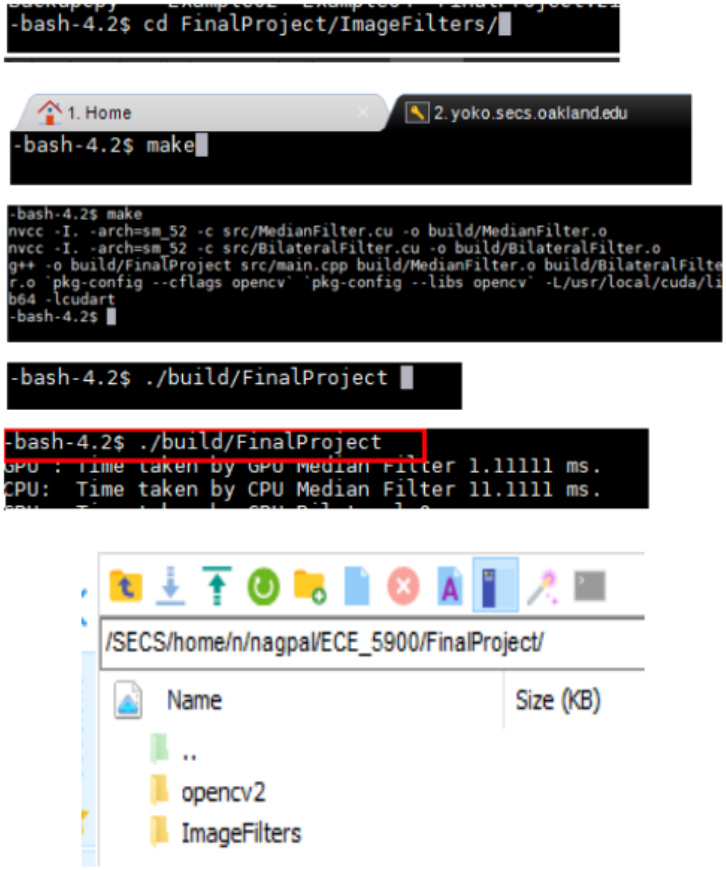
BenchMark Median filter



Cite this article as: Tedros Berhane and Namit Nagpal. "Image Filters Using CUDA on NVIDIA". *International Conference on Cloud of Things and Wearable Technologies 2018*: 06-11. Print.

How to Build the Project

OpenCV 3.3.1 is used in the project/Go to the project Directory/Type make Then. /build/Final Project/Folder Structure



Performance and Optimization (Shared Memory)

Shared Memory used to calculate the median of window in the median filter and change of the intensity in the Bilateral filter. Device constant Used to define the window size

CONCLUSIONS

Overall, we learned how to use both sequential and parallel programs through image filtering and processing in OpenCV. The most important part of the learning process was access OpenCV available libraries and functions to set up an image in in C++ programming language and resizing it. We learned how to customize freely available of both NVIDIA CUDA and OpenCV. We learned how to compare performance results with core processors and Graphical Processors. Eventually, the team has learned how to approach 2D image through tiling and shared memory accesses.

REFERENCES

1. Yipin Zhou (zyp), cs129/hdr. [Online]. Available: http://cs.brown.edu/courses/cs129/results/proj5/zyp/. [Accessed: 07-Dec-2017].
2. Median filter Wikipedia, 05-Nov-2017. [Online]. Available: https://en.wikipedia.org/wiki/Median_filter. [Accessed: 07-Dec-2017].
3. Bilateral filter, Wikipedia, 29-Oct-2017. [Online]. Available: https://en.wikipedia.org/wiki/Bilateral_filter. [Accessed: 07- Dec-2017].
4. OpenCV, NVIDIA Developer, 27-Apr-2017. [Online]. Available: https://developer.nvidia.com/opencv [Accessed: 08-Dec-2017].
5. Parallel computing, Wikipedia, 06-Dec-2017. [Online]. Available: https://en.wikipedia.org/wiki/Parallel_computing. [Accessed: 08-Dec-2017].
6. Computer vision, Wikipedia, 09-Oct-2017. [Online]. Available: https://en.wikipedia.org/wiki/Computervision. [Accessed: 08-Dec-2017]

Cite this article as: Tedros Berhane and Namit Nagpal. "Image Filters Using CUDA on NVIDIA". *International Conference on Cloud of Things and Wearable Technologies 2018*: 06-11. Print.