



University of Colorado  
Boulder

## **[ECEN 5593]- Implementation and Analysis of Adaptive Histogram Equalization in PyCUDA**

**Date:** 12 May. 2017

### **Team Members:**

**Name:** Maitri Chattopadhyay

**Contact:** +1 720 755 7694

**Email:** mach5953@colorado.edu

**Name:** Subhradeep Dutta

**Contact:** +1 720 755 7799

**Email:** sudu7404@colorado.edu

**Objective**

# Advanced Computer Architecture

## Project Report

To design a parallel implementation of Adaptive Histogram Equalization (AHE) algorithm and compare its performance with and without GPU acceleration using Python and PyCUDA respectively as the development platform. Various kernel/windows sizes are put under observation to study their effects on execution time of the algorithm.

### Purpose of the Project

Adaptive Histogram Equalization can be extensively applied in medical imagery as well as many other fields. But it has been used in limited applications due to the fact that it is computationally intensive. By implementing the algorithm in GPU, the computation time can be reduced by a huge margin. This will not only be a great advantage for the existing applications but will also help broaden the horizon for the application of the algorithm.

### Project Environment Setup

The project environment is setup in Windows with the NVIDIA GeForce 960M GPU as the graphics hardware. The GPU used is the Nvidia GeForce 960M which has a compute capability of 5.0.

The following applications are to be installed for the next step

- NVIDIA CUDA toolkit
- Visual Studio Professional Edition 2013

DeviceQuery examples can be run to verify that all the installations are successful.

Next step is to set up PyCUDA on the system along with its libraries. The executable python Wheel library can be downloaded from <http://www.lfd.uci.edu/~gohlke/pythonlibs/#pycuda> and can be installed via the following command:

```
pip install <directory>\pycuda-2016.1.2+cuda8044-cp27-cp27m-win_amd64.whl
```

This executable file depends on the Python, CUDA and Windows versions. To verify that PyCUDA has been installed successfully, sample code from the website can be used.

Next, OpenCV installation has to be carried out. The system must have all the following prerequisites prior to installing OpenCV.

1. Python-2.7
2. Numpy.
3. Matplotlib

To verify the successful installation of OpenCV, the following commands can be run.

```
import cv2
print cv2.__version__
```

# Advanced Computer Architecture

## Project Report

### Related Work

Given the importance of the algorithm, some previous work has been done to make the algorithm run faster. One such attempt can be found at <http://ieeexplore.ieee.org/document/128965/>. Although this paper throws some light on how to make the algorithm faster, it does not use a GPU. It uses message passing between slave and master. Using GPU to make the algorithm faster is an interesting approach that has not been much explored.

Various applications of this algorithm have been discussed in <http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=1234014>. Various implementations for adaptive histogram equalization has been discussed in <http://dl.acm.org/citation.cfm?id=29046>.

### Algorithm Implementation

The adaptive histogram equalization (AHE) algorithm is a modified version of histogram equalization that is used for contrast enhancement in images. It differs from the traditional histogram equalization in the fact that it computes multiple histograms for multiple sections of the image. It is specially aimed at improving local contrast and enhancing edges in the image. Adaptive histogram equalization produces better results especially for those images which have specific regions that are more dark or bright as compared to the rest of the image.

An alternative implementation for this algorithm also exists which uses a sliding window approach to compute the new pixel values based on the values of the neighboring pixels. This implementation of the algorithm has been exploited in this project.

### Working and Methodology

The working of the algorithm is listed in sequential manner as follows:

1. Iterate through every pixel in the image
2. For every pixel  $a[i][j]$  extract a subset of the image based on the value provided by the user.
3. Now the pixel  $a[i][j]$  is ranked with respect to the other pixel values in the window
4. The new pixel value is then calculated based on the below formula
$$a*[i][j] = (\text{rank} * \text{maximum intensity value}) / (\text{window size} * \text{window size})$$

First the algorithm is developed in Python. OpenCV library is leveraged to read and write the images from the current working directory. The user is then prompted to enter the window size. The image is then reflected along the edges to the extent of the window size so that the loop can iterate till the last pixel of the original image. A duplicate array of the same dimensions is created which is used to store the new pixel values. At this point the timer is turned on before entering the actual logic block. Inside

## Advanced Computer Architecture

# Project Report

the nested loops, each pixel in the image is iterated through and a subset of the image is extracted which is defined by the user given window size. Now this subset of the image is converted to a 1D array and then sorted. The original pixel value in the sorted array is located and its index is used as rank. Then this value is fed into the formula to calculate the pixel value. To verify that the CPU and GPU output images are same, the difference between both the images is calculated. If both are identical, the result should be a perfect black image.

### Testing and Verification

Detailed testing and analysis of the algorithm was done and the images and data obtained is documented below.

Below is the original image used for analysis. The image is hazy and not very clear. Some details such as the device located in the rib area is difficult to make out.



**Figure1: the original image**

Below is the image obtained with window size 5. The image is sharper now with clear distinct boundaries between each bone. The details in the image can be further improved by increasing the window size.

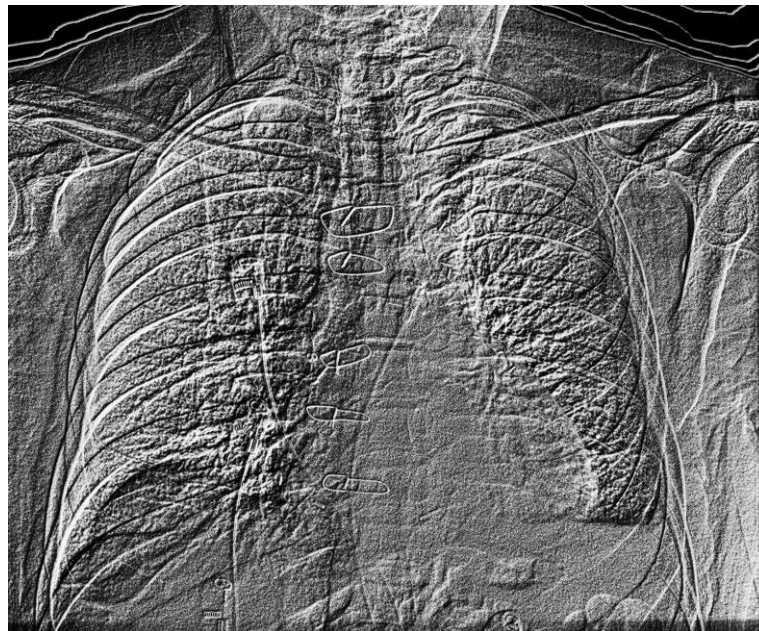
# Advanced Computer Architecture

## Project Report



**Figure 2: the image obtained after running the algorithm with window size 5.**

The below image is obtained after running the algorithm with window size of 10. It is notably clear and more sharper.

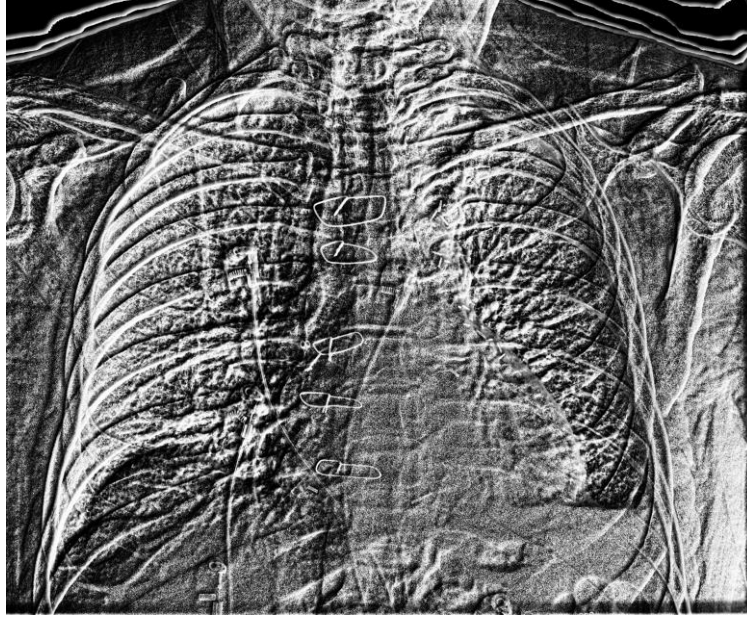


**Figure 3: the image obtained after running the algorithm with window size 10**

The below image is obtained after running the algorithm with window size of 20.

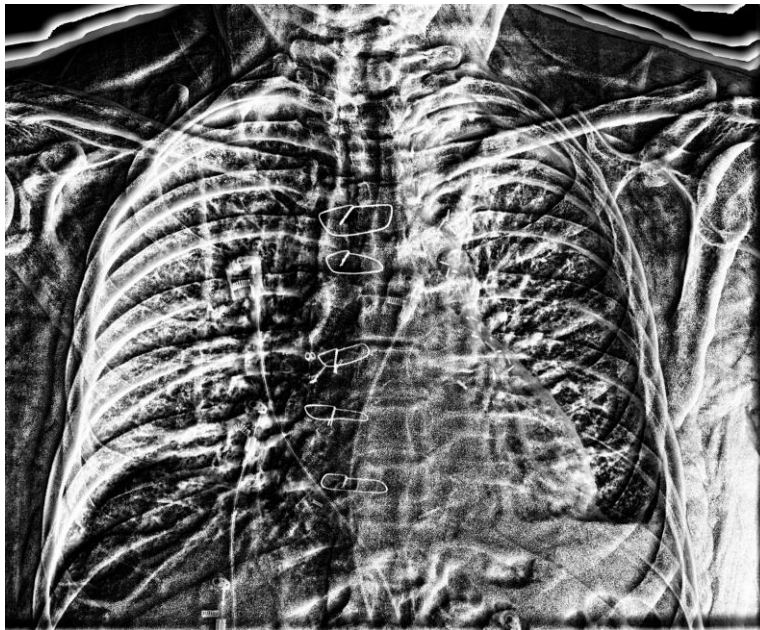
# Advanced Computer Architecture

## Project Report



**Figure 4: the image obtained after running the algorithm with window size 20**

The below image is obtained after running the algorithm with window size of 40.

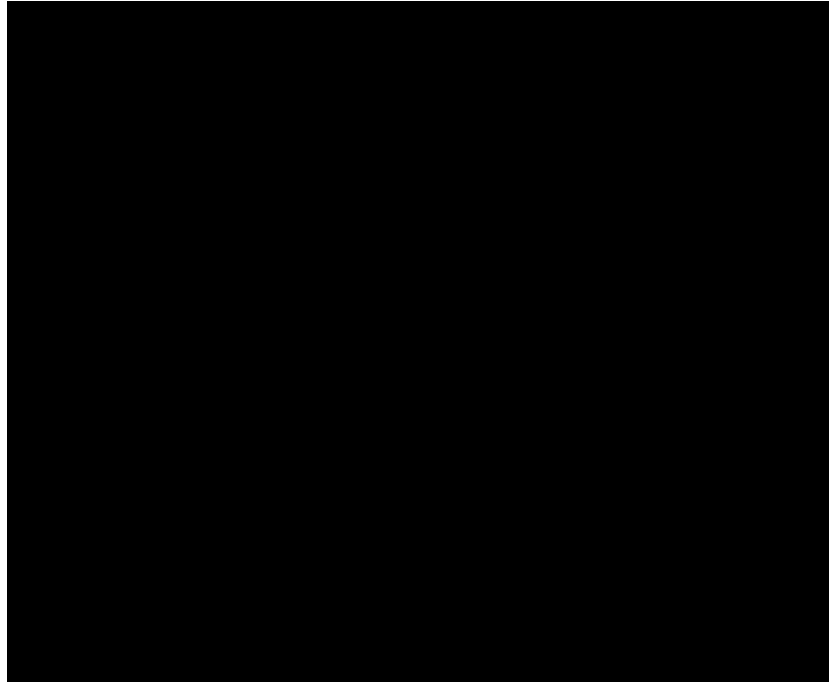


**Figure 5: the image obtained after running the algorithm with window size 40**

The image below is obtained after subtracting the CPU and GPU images. The perfect black image is a proof that the images are identical.

# Advanced Computer Architecture

## Project Report



**Figure 6: Black image obtained by subtracting the CPU and GPU**

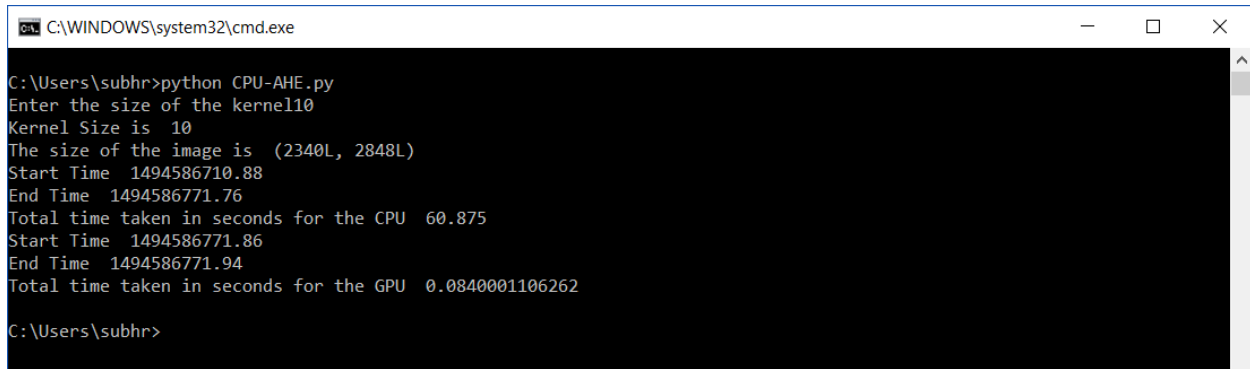
The four images below show the time taken by the algorithm to run in both CPU as well as GPU. It can be noted that the time taken for the GPU execution of the algorithm is almost 250 times faster.

```
C:\WINDOWS\system32\cmd.exe
C:\Users\subhr>python CPU-AHE.py
Enter the size of the kernel5
Kernel Size is 5
The size of the image is (2330L, 2838L)
Start Time 1494586619.36
End Time 1494586663.82
Total time taken in seconds for the CPU 44.4539999962
Start Time 1494586663.92
End Time 1494586663.95
Total time taken in seconds for the GPU 0.0320000648499
C:\Users\subhr>
```

**Figure 7: Execution time for window size 5**

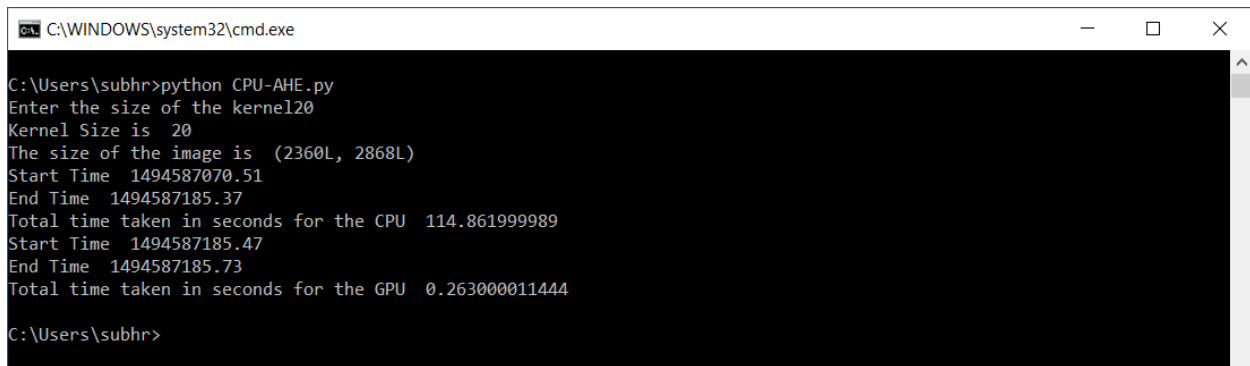
# Advanced Computer Architecture

## Project Report



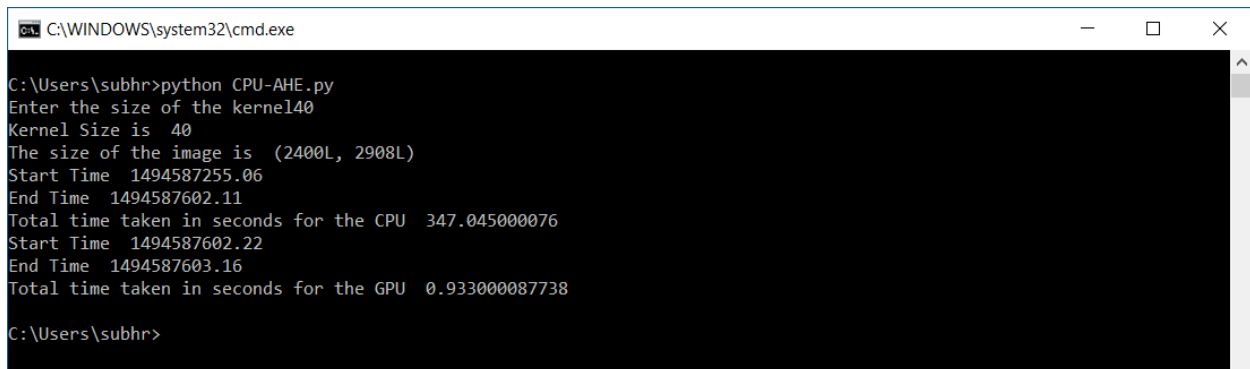
```
C:\WINDOWS\system32\cmd.exe
C:\Users\subhr>python CPU-AHE.py
Enter the size of the kernel10
Kernel Size is 10
The size of the image is (2340L, 2848L)
Start Time 1494586710.88
End Time 1494586771.76
Total time taken in seconds for the CPU 60.875
Start Time 1494586771.86
End Time 1494586771.94
Total time taken in seconds for the GPU 0.0840001106262
C:\Users\subhr>
```

**Figure 8: Execution time for window size 10**



```
C:\WINDOWS\system32\cmd.exe
C:\Users\subhr>python CPU-AHE.py
Enter the size of the kernel20
Kernel Size is 20
The size of the image is (2360L, 2868L)
Start Time 1494587070.51
End Time 1494587185.37
Total time taken in seconds for the CPU 114.861999989
Start Time 1494587185.47
End Time 1494587185.73
Total time taken in seconds for the GPU 0.263000011444
C:\Users\subhr>
```

**Figure 9: Execution time for window size 20**



```
C:\WINDOWS\system32\cmd.exe
C:\Users\subhr>python CPU-AHE.py
Enter the size of the kernel40
Kernel Size is 40
The size of the image is (2400L, 2908L)
Start Time 1494587255.06
End Time 1494587602.11
Total time taken in seconds for the CPU 347.045000076
Start Time 1494587602.22
End Time 1494587603.16
Total time taken in seconds for the GPU 0.933000087738
C:\Users\subhr>
```

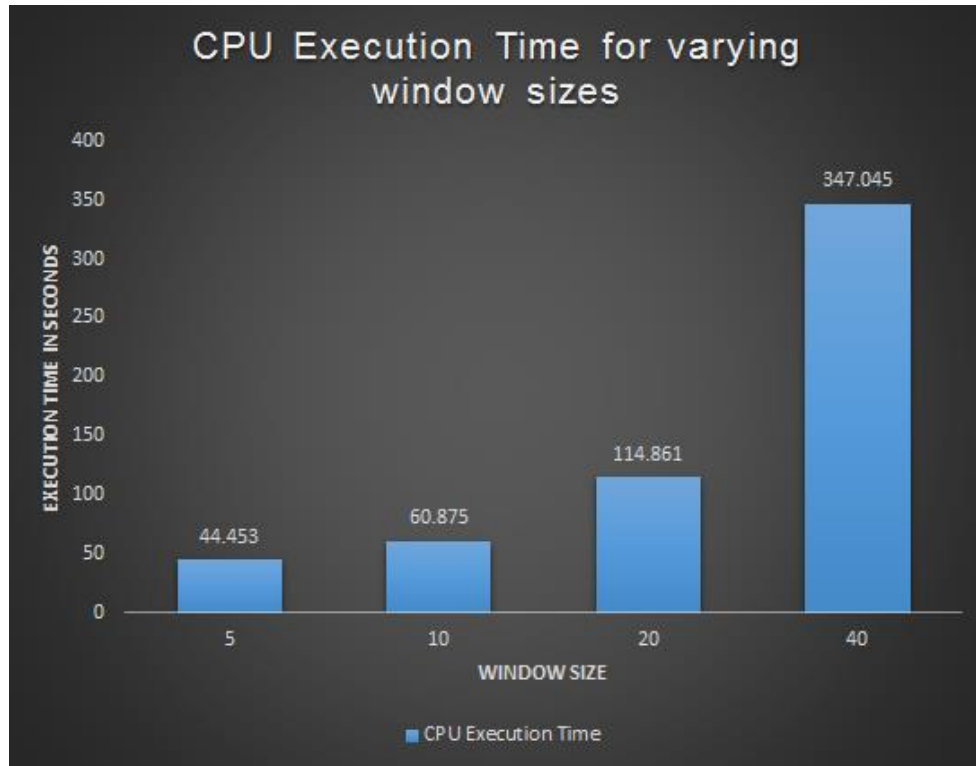
**Figure 10: Execution time for window size 40**



## Advanced Computer Architecture

# Project Report

The graph below shows the CPU execution time for various window sizes. As the window size increases, the time taken to execute also increases. Although the time taken increases with the window size, it doesn't get doubled when the window size is doubled. Hence the time taken is not directly proportional to the increase in window size.

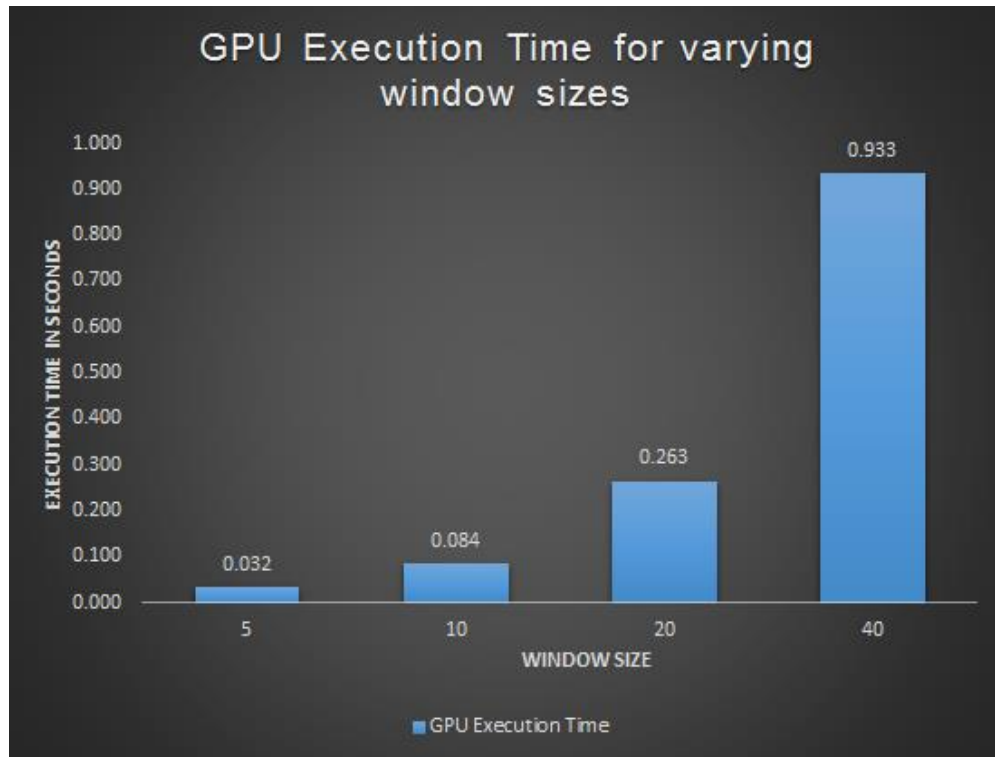


**Graph 1: CPU Execution time for varying window sizes**

The graph below shows the GPU execution time for various window sizes. The same trend in time in case of CPU execution is also observed in the GPU timings.

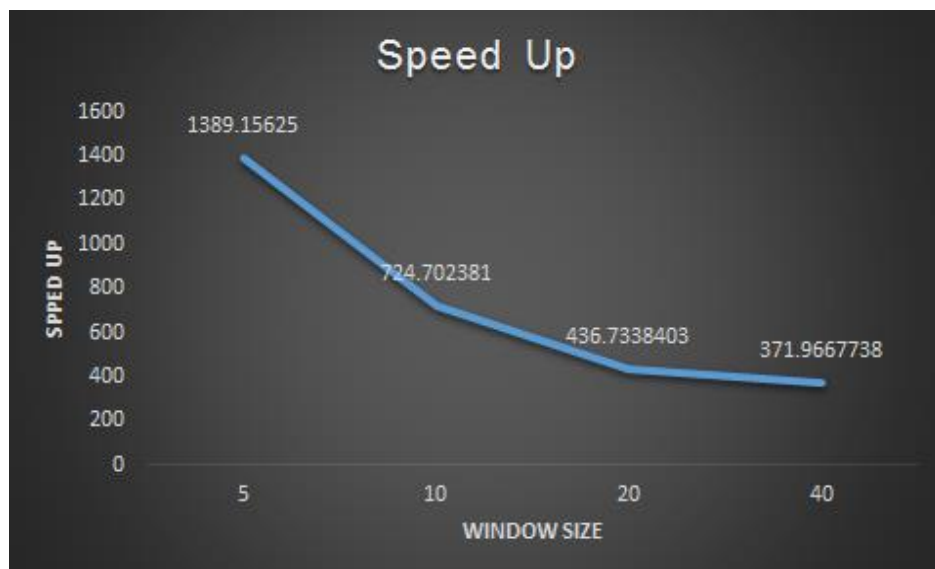
# Advanced Computer Architecture

## Project Report



**Graph 2: GPU Execution time for varying window sizes**

The following graph shows the speed up between the CPU and the GPU. Speedup is calculated by CPU Execution Time/GPU Execution Time

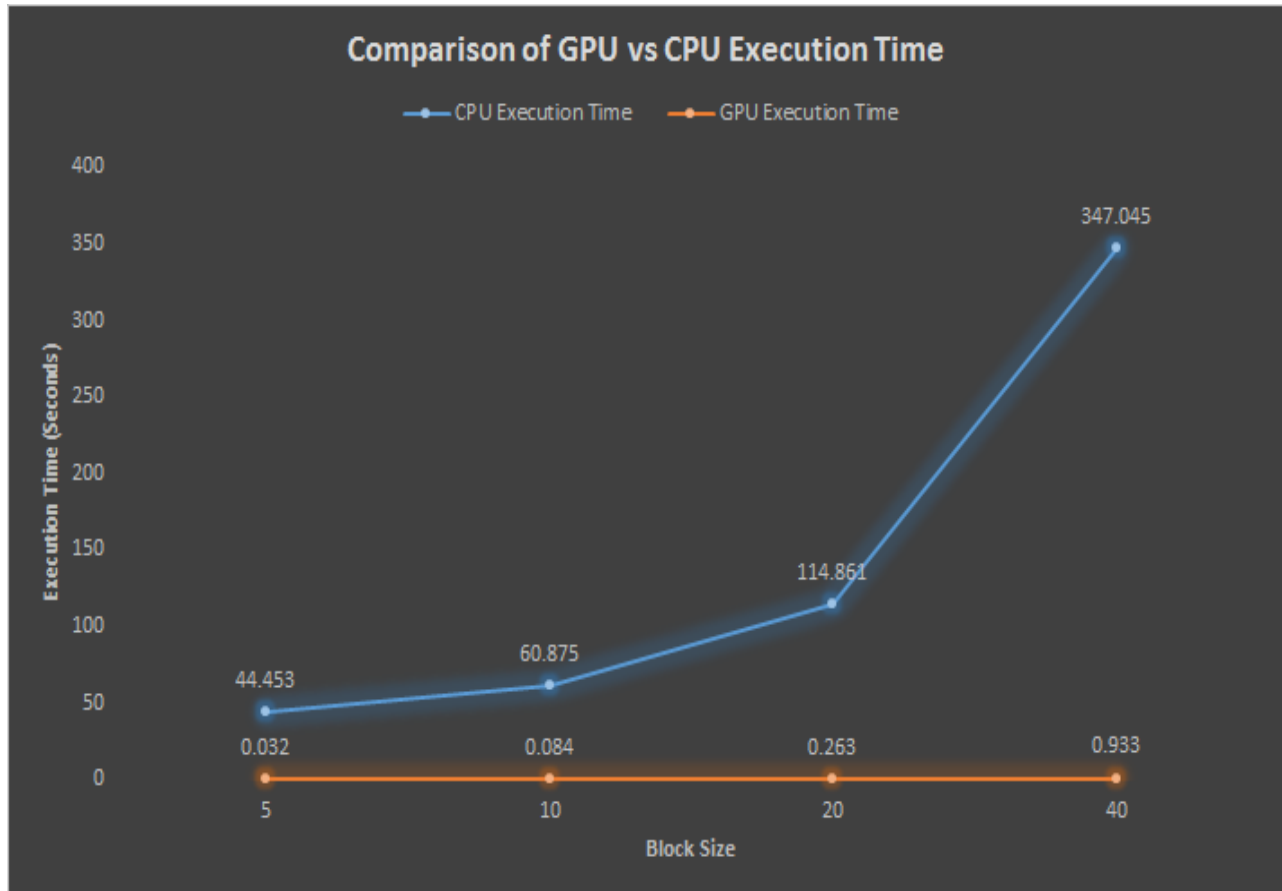


**Graph 3: Speed Up for varying window sizes**

The following graph shows the comparison between the CPU and GPU execution time.

# Advanced Computer Architecture

## Project Report



**Graph 4: CPU vs GPU execution time**

### Project Roadblocks / Difficulties

- When the algorithm was initially developed in python, it had 6 nested for loops to carry out the operations. This approach took a lot of time which made the implementation unrealistic to be used in real world.
- The algorithm was then modified to use python builtin functions to reduce the complexity. In the end, a more optimized code was produced with complexity of  $n^2$  and realistic execution time (which could be further reduced by a parallel implementation)
- There is very limited amount of resources as far as parallel computing using PyCUDA is concerned. Consequently, it was difficult translating the Python code to PyCuda.
- The CPU version of the program used Python's built in sorting function. Translating the same logic in CUDA was a challenge.

### Future scope

## Advanced Computer Architecture

# Project Report

This algorithm highlights essential and minute details that are of significance in medical imaging and might not be visible in the original image prior to enhancement. After the algorithm was implemented in GPU, almost 300x speedup was observed in the runtime of the algorithm. This algorithm can also be used for enhancing images from other medical devices such as specimens under the slide.

### References:

- Image obtained from <http://www.eurorad.org/eurorad/case.php?id=9074&lang=en&lang=en>
- J. Alex Stark, Adaptive Image Contrast Enhancement Using Generalizations of Histogram Equalization
- S Muniyappan, Dr.A.Allirani & S.Saraswathi, A Novel Approach for Image Enhancement by Using Contrast Limited Adaptive Histogram Equalization Method
- Charles W. Kurak Jr, Adaptive Histogram Equalization, a Parallel Implementation
- Victor T. Tom, Gregory J. Wolfe, Adaptive histogram equalization and its applications
- John B. Zimmerman, Stephen M. Pizer, Edward V. Staab, J. Randolph Perry, William McCartney, Bradley C. Brenton, An Evaluation of the Effectiveness of Adaptive Histogram Equalization for Contrast Enhancement