

# ITCS443 Parallel and Distributed Systems

## Project Report

CUDA - BGR2GRAY and Image Addition

## Members

Kasidit Ruaydee 5988117

Pasin Pubpateeravanich 5988186

Patthanayu Rueangdej 5988195

Nuwat Tantbirojn 5988240

Presented to

Dr. Putt Sakdhnagool

Faculty of Information and Communication Technology

Mahidol University

2018



# 1 Introduction

## 1.1 Project goal

To successfully applied CUDA to python in image processing program to see if using CUDA is able to make a program process faster than serial implementation.

## 1.2 Why is this project interesting / worth to do?

We need to prove that if we able to use CUDA to apply parallel processing in any kind of program, the process of a program will be faster than serial implementation program with sequential processing. It's worth doing because if it can guarantees a program to have faster processing, it would be a good thing for us to be able to use it with other codes.

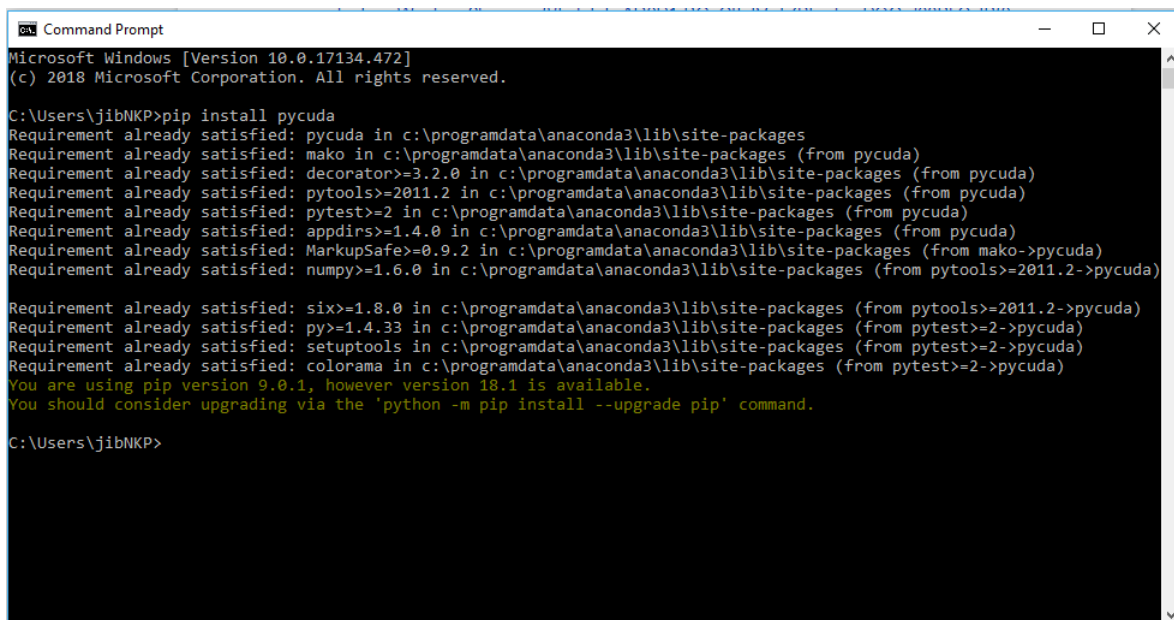
## 2 Methodology / Algorithm

### **\*\*Requirement for Pycuda\*\***

In this project, we implement the source code using Pycharm with Python language. In order to apply CUDA with Python, first we have to install library called Pycuda in the machine which is available on the website below.

[www.ibm.com/developerworks/community/blogs/jfp/entry/Installing\\_PyCUDA\\_On\\_Anaconda\\_For\\_Windows?lang=en&fbclid=IwAROP1-BGoCtI\\_tVeT7BbndyprPOG-d08BkCstBf8P\\_WQOUwkeAMrH1TaGDo](http://www.ibm.com/developerworks/community/blogs/jfp/entry/Installing_PyCUDA_On_Anaconda_For_Windows?lang=en&fbclid=IwAROP1-BGoCtI_tVeT7BbndyprPOG-d08BkCstBf8P_WQOUwkeAMrH1TaGDo)

After pycuda is downloaded, continue to install pycuda on the machine through Anaconda prompt/ command prompt by typing the command line “pip install pycuda”[1].



```

Microsoft Windows [Version 10.0.17134.472]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\jibNKP>pip install pycuda
Requirement already satisfied: pycuda in c:\programdata\anaconda3\lib\site-packages
Requirement already satisfied: mako in c:\programdata\anaconda3\lib\site-packages (from pycuda)
Requirement already satisfied: decorator>=3.2.0 in c:\programdata\anaconda3\lib\site-packages (from pycuda)
Requirement already satisfied: pytools>=2011.2 in c:\programdata\anaconda3\lib\site-packages (from pycuda)
Requirement already satisfied: pytest>=2 in c:\programdata\anaconda3\lib\site-packages (from pycuda)
Requirement already satisfied: appdirs>=1.4.0 in c:\programdata\anaconda3\lib\site-packages (from pycuda)
Requirement already satisfied: MarkupSafe>=0.9.2 in c:\programdata\anaconda3\lib\site-packages (from mako->pycuda)
Requirement already satisfied: numpy>=1.6.0 in c:\programdata\anaconda3\lib\site-packages (from pytools>=2011.2->pycuda)
Requirement already satisfied: six>=1.8.0 in c:\programdata\anaconda3\lib\site-packages (from pytools>=2011.2->pycuda)
Requirement already satisfied: py>=1.4.33 in c:\programdata\anaconda3\lib\site-packages (from pytest>=2->pycuda)
Requirement already satisfied: setuptools in c:\programdata\anaconda3\lib\site-packages (from pytest>=2->pycuda)
Requirement already satisfied: colorama in c:\programdata\anaconda3\lib\site-packages (from pytest>=2->pycuda)
You are using pip version 9.0.1, however version 18.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.

C:\Users\jibNKP>

```

Figure 1: Pycuda Installation

After the installation is completed, we have to create paths for pycuda to be able to compile on Pycharm.

Path1:C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.0\bin  
 Path2:C:\Program Files (x86)\Microsoft Visual Studio 14.0\VC\bin

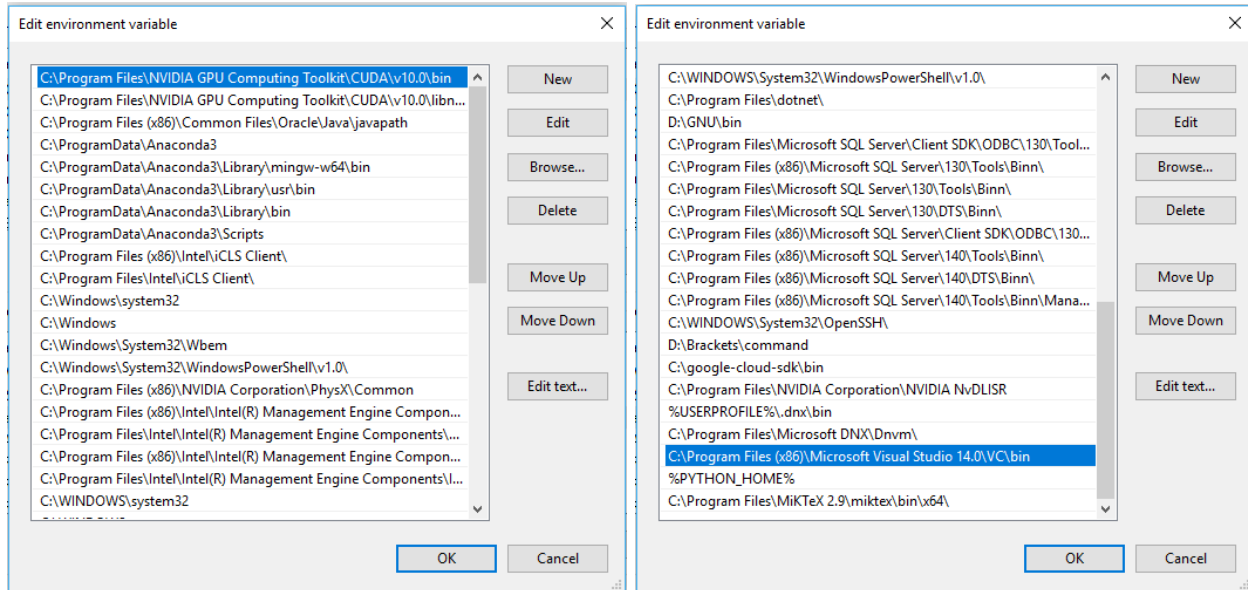


Figure 2: Creating paths on the machine

## 2.1 High-level description of the project / algorithm

### BGR2GRAY

The first program is the image conversion from BGR to Grayscale for 512\*512 px image. We create a program that allow user to convert the image from normal image with color to grayscale image. The result of a program will provide the original image and grayscale image.

### Image addition

The second program is the image addition for 512\*512 px image. We create a program that allow user to create a new image by combining two images together. For the obvious result, user might have to find one image that have transparent background to see the combination between two images. The result of a program will provide 2 of the original images and image that come from the combination of two images.

## 2.2 Detailed explanation of algorithm

In both program, we implemented the code that allow us to know the execution time for each code for the comparison and experimental results[2].

### 2.2.1 Pycuda\_BGR2GRAY.py

#### Kernel function

We created kernel function name's "bgr2gray". At the kernel function, we implement a formula to calculate the values of color in each pixel into grayscale. The position of each pixel on the image will be represented by using variable a to represent row and b to represent column[3][4].

```
#define INDEX(a, b) a*512+b

__global__ void bgr2gray(float *result, float *bb, float *gg, float *rr){
    unsigned int i = threadIdx.x + (blockIdx.x * (blockDim.x*blockDim.y));
    unsigned int a = i/512;
    unsigned int b = i%512;
    result[INDEX(a, b)] = (0.299*rr[INDEX(a, b)] + 0.587*gg[INDEX(a, b)] + 0.114*bb[INDEX(a, b)]);
}
```

Figure 3: Kernel function "bgr2gray"

#### Main function

A program will read image and then separate into 3 shades which are blue, green, and red with the same size (262144 comes from 512\*512). The variable "result" will copy the array size of one of three shades above and use it to keep the output from kernel.

```
# read the image
img = cv2.imread("doggy.jpg",1)

# Separate shades (262144 = 512*512)
b_img = img[:, :, 0].reshape(262144).astype(np.float32)
g_img = img[:, :, 1].reshape(262144).astype(np.float32)
r_img = img[:, :, 2].reshape(262144).astype(np.float32)
result = g_img
```

Figure 4: Read image and separate shades

After that, program will call kernel function name's "bgr2gray" to convert the image into grayscale by passing pointer of b\_img, g\_img, r\_img, and result. After kernel function is finished, program will display two output which are the original image and grayscale image[5].

```
# to kernel
func = mod.get_function("bgr2gray")
func(cuda.Out(result), cuda.In(b_img), cuda.In(g_img), cuda.In(r_img), block_=(1024, 1, 1), grid_=(256, 1, 1))

# Show the result
result = np.reshape(result, (512, 512)).astype(np.uint8)
cv2.imshow("Original",img)
cv2.imshow("Grayscale_CUDA",result)
```

Figure 5: Call kernel function "bgr2gray" and display output

### 2.2.2 Pycuda\_image\_addition.py

#### Kernel function

We created kernel function name's "addNum". At the kernel function, we implement a formula to calculate summation of values in each pixel on the first and second image to combine them together. The position of each pixel on the image will be represented by using variable a to represent row and b to represent column.

```
global void addNum(float *result, float *a, float *b, int n){
    int tid = threadIdx.x + blockDim.x * blockIdx.x;
    while(tid < n){
        result[tid] = a[tid] + b[tid];
        if(result[tid] > 511) result[tid] = 511;
        tid = tid + blockDim.x * gridDim.x;
    }
}
```

Figure 6: Kernel function "addNum"

#### Main function

A program will read two images as grayscale. After that, program will create variables "newimg1" and "newimg2" to receive size of the image which are the same, and variable "n" will be referred to width of the image. The variable "result" will copy the array size of one of newimg above and use it to keep the output from kernel.

```
# read images as grayscale
img1 = cv2.imread("doggy.jpg", 0)
img2 = cv2.imread("Illuminati.png", 0)

# Size (262144 = 512*512)
newimg1 = img1.reshape(262144).astype(np.float32)
newimg2 = img2.reshape(262144).astype(np.float32)
n = newimg1.size

# make the result to have same size as newimg1
result = newimg1
```

Figure 7: Read images and define size

After that, program will call kernel function name's "addNum" to calculate for the summation values of two image and combine them together. After kernel function is finished, program will display three output which are two of the original images and combined image[6].

```
# to kernel
func = mod.get_function("addNum")
func(cuda.Out(result), cuda.In(newimg1), cuda.In(newimg2), np.uint32(n), block = (1024, 1, 1), grid = (64, 1, 1))

# Show the result
result = np.reshape(result, (512, 512)).astype(np.uint8)
cv2.imshow("Dog", img1)
cv2.imshow("Illuminati", img2)
cv2.imshow("Added image_CUDA", result)
```

Figure 8: Call kernel function "addNum" and display output

### 3 Experiment and results

#### CUDA

We've created BGR2GRAY and Image addition with normal python language in order to compare with Pycuda source code and get the experimental result.

```
import cv2
import time

start_time = time.time()

img = cv2.imread("doggy.jpg")

gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

cv2.imshow("Original", img)
cv2.imshow("Grayscale", gray)
print("time elapsed: {:.4f}s".format(time.time() - start_time))
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Figure 9: Python version of BGR2GRAY

```
import cv2
import time

start_time = time.time()
img1 = cv2.imread("doggy.jpg", 0)
img2 = cv2.imread("Illuminati.png", 0)

img3 = img1 + img2

cv2.imshow("Image 1", img1)
cv2.imshow("Image 2", img2)
cv2.imshow("Added image", img3)
print("time elapsed: {:.4f}s".format(time.time() - start_time))
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Figure 10: Python version of Image addition

For the image processing result, both programs have the same output which mean that programs that has CUDA applied in it provide the accurate and correct output.

### Result for BGR2GRAY

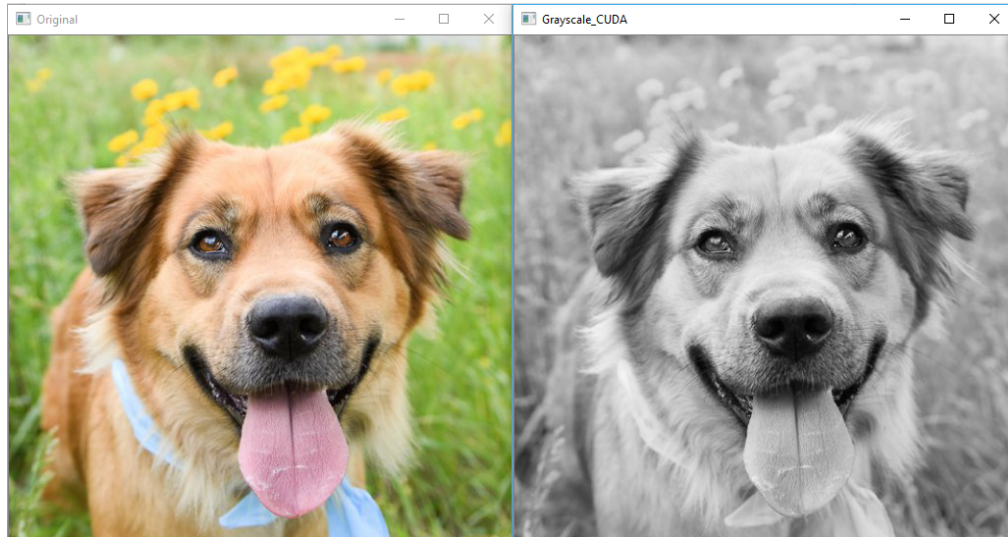


Figure 11: Output of Pycuda\_BGR2GRAY

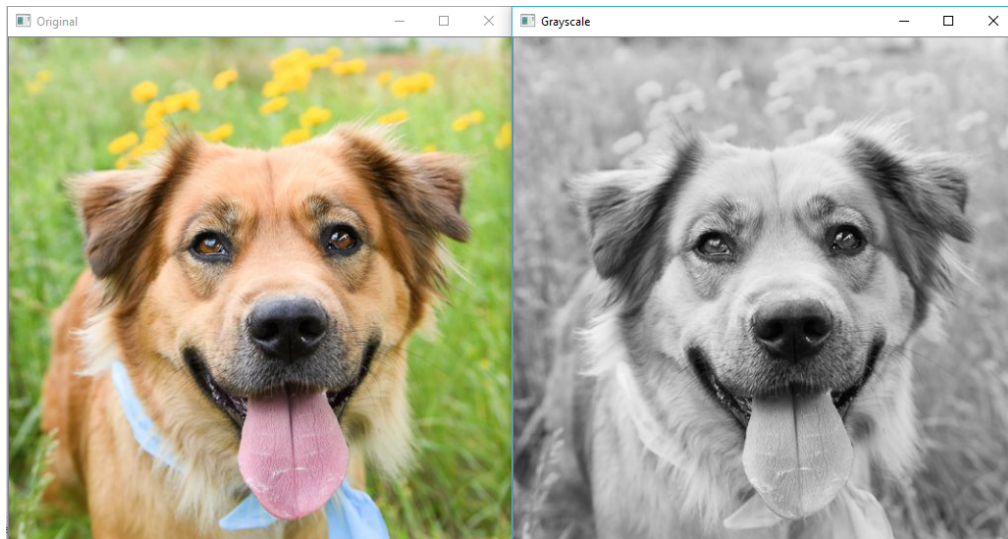


Figure 12: Output of BGR2GRAY



## Result for Image addition

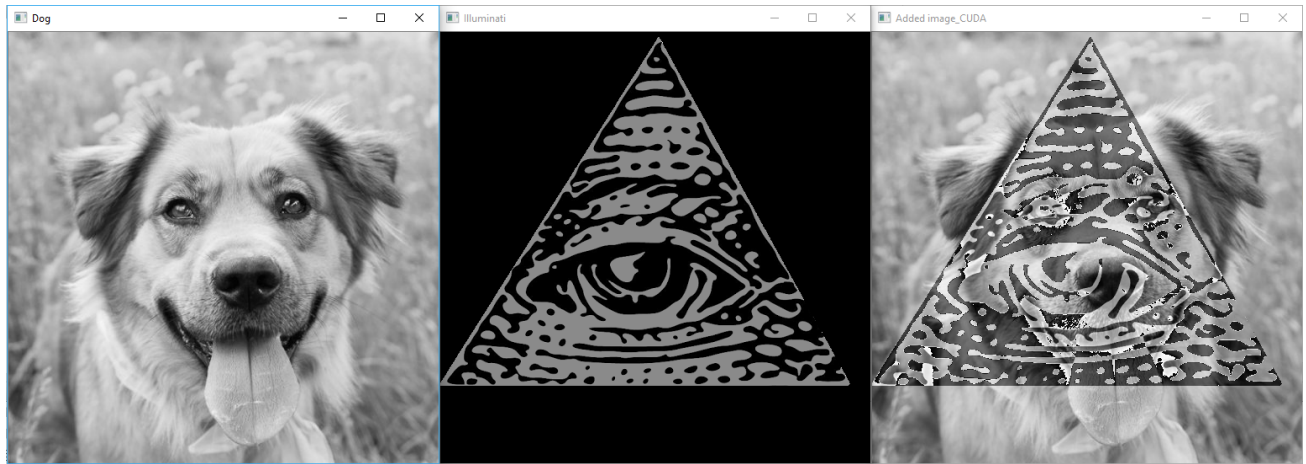


Figure 13: Output of Pycuda\_image\_addition

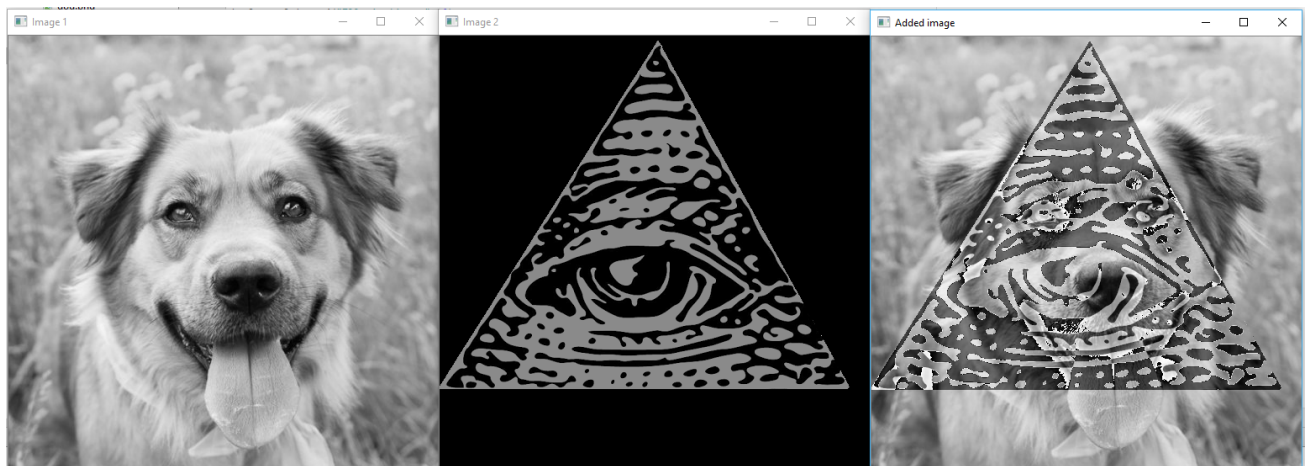


Figure 14: Output of Image addition

On the other hand, the average execution time of Pycuda version and python version almost have the same result. Sometimes Pycuda is faster than Python, and sometimes Python is faster than Pycuda, with little number of different.

### Execution time for BGR2GRAY

```
C:\ProgramData\Anaconda3\python.exe C:/Users/jibNKP/PycharmProjects/test/Pycuda_BGR2GRAY
time elapsed: 0.0618s

Process finished with exit code 0
```

Figure 15: Execution time of Pycuda\_BGR2GRAY

```
C:\ProgramData\Anaconda3\python.exe C:/Users/jibNKP/PycharmProjects/test/BGR2GRAY
time elapsed: 0.0549s

Process finished with exit code 0
```

Figure 16: Execution time of BGR2GRAY

### Execution time for Image addition

```
C:\ProgramData\Anaconda3\python.exe C:/Users/jibNKP/PycharmProjects/test/Pycuda_image_addition
time elapsed: 0.0718s

Process finished with exit code 0
```

Figure 17: Execution time of Pycuda\_image\_addition

---

```
C:\ProgramData\Anaconda3\python.exe "C:/Users/jibNKP/PycharmProjects/test/Image addition"
time elapsed: 0.0608s

Process finished with exit code 0
```

Figure 18: Execution time of Image addition

## 4 Discussion and conclusion

### 4.1 Discuss the results of your work

After the discussion about the result, first we have the same hypothesis of the result for CUDA that it able to make a process faster than normal python program, but the output told us a different story. Due to the curiosity, we've done research more about Pycuda and found that python program that use numpy in the code is already speed up the code. Pycuda is good for accelerate expensive operations. The execution time might not have much different, it usually depend on the size of the images and program flow. There is latency involved in passing the data back and forth across the PCI bus that is only made up for with large data sizes. In conclusion, we can applied cuda to use with python to implement code with accurate output and program execution time between both pycuda and python might not have much different[7].

### 4.2 How to further improve the work

If we have more time and opportunity to improve the source code for the better result, we would use the advantages of shared memory to apply in the project. Using shared memory could improve a program to be able to work on several images at the same time in multiple thread blocks which might reduce the execution time of a program.

## References

- [1] Jean Francois Puget. (2016). *Installing PyCUDA On Anaconda For Windows (IT Best Kept Secret Is Optimization)*. [online] Available at:  
[https://www.ibm.com/developerworks/community/blogs/jfp/entry/Installing\\_PyCUDA\\_On\\_Anaconda\\_For\\_Windows?lang=en&fbclid=IwAROP1-BGoCtI\\_tVeT7BbndyprP0G-d08BkCstBf8P\\_WQOUwkeAMrH1TaGDo](https://www.ibm.com/developerworks/community/blogs/jfp/entry/Installing_PyCUDA_On_Anaconda_For_Windows?lang=en&fbclid=IwAROP1-BGoCtI_tVeT7BbndyprP0G-d08BkCstBf8P_WQOUwkeAMrH1TaGDo)
- [2] KarolisR. (2016). *How do I time script execution time in PyCharm without adding code every time?*. [online] Stack Overflow. Available at:  
<https://stackoverflow.com/questions/35656239/how-do-i-time-script-execution-time-in-pycharm-without-adding-code-every-time>
- [3] waspinator. (2012). *How can I convert an RGB image into grayscale in Python?*. [online] Stack Overflow. Available at:  
<https://stackoverflow.com/questions/12201577/how-can-i-convert-an-rgb-image-into-grayscale-in-python>
- [4] En.wikipedia.org. (2018). *Grayscale*. [online] Available at:  
[https://en.wikipedia.org/wiki/Grayscale#Converting\\_color\\_to\\_grayscale](https://en.wikipedia.org/wiki/Grayscale#Converting_color_to_grayscale)
- [5] Ashwin Ashok. (2014). *PyCuda/Examples/SimpleRGB2Gray - Andreas Klöckner's wiki*. [online] Available at:  
<https://wiki.tiker.net/PyCuda/Examples/SimpleRGB2Gray>
- [6] Alexander Mordvintsev & Abid K. (2013). *Arithmetic Operations on Images — OpenCV-Python Tutorials 1 documentation*. [online] Available at:  
[https://opencv-python-tutroals.readthedocs.io/en/latest/py\\_tutorials/py\\_core/py\\_image\\_arithmetics/py\\_image\\_arithmetics.html](https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_core/py_image_arithmetics/py_image_arithmetics.html)
- [7] ardiyu07. (2011). *processing an image using CUDA implementation, python (pycuda) or C++?*. [online] Stack Overflow. Available at:  
[https://stackoverflow.com/questions/4970580/processing-an-image-using-cuda-implementation-python-pycuda-or-c?fbclid=IwAR0Z4tug1Hs7E\\_3iVMNtSOQKQbKzTU71LDaEufVjruk8j\\_zVrQhk1t2Iug](https://stackoverflow.com/questions/4970580/processing-an-image-using-cuda-implementation-python-pycuda-or-c?fbclid=IwAR0Z4tug1Hs7E_3iVMNtSOQKQbKzTU71LDaEufVjruk8j_zVrQhk1t2Iug)

## Appendix

### Pycuda\_BGR2GRAY.py

```
# Image conversion to grayscale using Pycuda

import pycuda.driver as cuda
import pycuda.autoinit
from pycuda.compiler import SourceModule
import numpy as np
import cv2
import time # Import for execution time calculation

mod = SourceModule\
(
    """
    #define INDEX(a, b) a*512+b

    __global__ void bgr2gray(float *result, float *bb, float *gg, float *rr){
        unsigned int i = threadIdx.x + (blockIdx.x * (blockDim.x*blockDim.y));
        unsigned int a = i/512;
        unsigned int b = i%512;
        result[INDEX(a, b)] = (0.299*rr[INDEX(a, b)] + 0.587*gg[INDEX(a, b)] + 0.114*bb[INDEX(a, b)]);
    }
    """
)

# Start the timer
start_time = time.time()

# read the image
img = cv2.imread("doggy.jpg",1)

# Separate shades (262144 = 512*512)
b_img = img[:, :, 0].reshape(262144).astype(np.float32)
g_img = img[:, :, 1].reshape(262144).astype(np.float32)
r_img = img[:, :, 2].reshape(262144).astype(np.float32)
result = g_img

# to kernel
func = mod.get_function("bgr2gray")
func(cuda.Out(result), cuda.In(b_img), cuda.In(g_img), cuda.In(r_img), block=(1024,1,1), grid=(256,1,1))

# Show the result
result = np.reshape(result, (512, 512)).astype(np.uint8)
cv2.imshow("Original",img)
cv2.imshow("Grayscale_CUDA",result)

# Print execution time result
print ("time elapsed: {:.4f}s".format(time.time() - start_time))
cv2.waitKey(0)
cv2.destroyAllWindows()
```

## Pycuda\_image\_addition.py

```
# Image addition using Pycuda

import pycuda.driver as cuda
import pycuda.autoinit
from pycuda.compiler import SourceModule
import numpy as np
import cv2
import time # Import for execution time calculation

mod = SourceModule("""
    __global__ void addNum(float *result, float *a, float *b, int n){
        int tid = threadIdx.x + blockDim.x * blockIdx.x;
        while(tid < n){
            result[tid] = a[tid] + b[tid];
            if(result[tid] > 511) result[tid] = 511;
            tid = tid + blockDim.x * gridDim.x;
        }
    }
""")

# Start the timer
start_time = time.time()

# read images as grayscale
img1 = cv2.imread("doggy.jpg", 0)
img2 = cv2.imread("Illuminati.png", 0)

# Size (262144 = 512*512)
newimg1 = img1.reshape(262144).astype(np.float32)
newimg2 = img2.reshape(262144).astype(np.float32)
n = newimg1.size

# make the result to have same size as newimg1
result = newimg1

# to kernel
func = mod.get_function("addNum")
func(cuda.Out(result), cuda.In(newimg1), cuda.In(newimg2), np.uint32(n), block=(1024,1,1), grid=(64,1,1))

# Show the result
result = np.reshape(result, (512, 512)).astype(np.uint8)
cv2.imshow("Dog", img1)
cv2.imshow("Illuminati", img2)
cv2.imshow("Added image_CUDA", result)

# Print execution time result
print ("time elapsed: {:.4f}s".format(time.time() - start_time))

cv2.waitKey(0)
cv2.destroyAllWindows()
```

## BGR2GRAY.py

```
import cv2
import time

start_time = time.time()

img = cv2.imread("doggy.jpg")

gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

cv2.imshow("Original", img)
cv2.imshow("Grayscale", gray)
print ("time elapsed: {:.4f}s".format(time.time() - start_time))
cv2.waitKey(0)
cv2.destroyAllWindows()
```

## Image addition.py

```
import cv2
import time

start_time = time.time()
img1 = cv2.imread("doggy.jpg", 0)
img2 = cv2.imread("Illuminati.png", 0)

img3 = img1 + img2

cv2.imshow("Image 1", img1)
cv2.imshow("Image 2", img2)
cv2.imshow("Added image", img3)
print ("time elapsed: {:.4f}s".format(time.time() - start_time))
cv2.waitKey(0)
cv2.destroyAllWindows()
```