CPU and GPU Computing Evaluation

By Aria Moradi

In this project we aim to compare execution performance of CPU and GPU for simple computation problems like matrix multiplication and image processing.

There are 2 scenarios, the first one is multiplication of two large random matrices:

```
""" The Explanation of the Code:
In this code, we start by defining the size of the square matrices to
be multiplied (matrix_size).
We then generate two random matrices (matrix_a and matrix_b) using the
np.random.rand function.

We define two functions: cpu_matrix_multiplication and
gpu_matrix_multiplication,
which perform matrix multiplication using CPU and GPU, respectively.

In the CPU function, we use the NumPy np.dot function for matrix
multiplication.
In the GPU function, we use the cp.asarray function from the cupy
library
to transfer the matrices to the GPU, perform matrix multiplication
using cp.dot,
and then transfer the result back to the CPU using cp.asnumpy.

After running the CPU and GPU matrix multiplication functions,
we can optionally compare the results and print the execution times.

Please note that for the GPU matrix multiplication to work,
you need to have the cupy library installed, which provides
a NumPy-compatible interface for GPU computations.
Also, make sure you have a compatible GPU and CUDA drivers installed.

Feel free to adjust the matrix size and modify the code according to
your specific requirements.
"""

import numpy as np
import cupy as cp
import tensorflow as tf
import torch
import time

# Define the matrix sizes
matrix_size = 1000
test_count = 100
```

```
matrix_pairs = []

for i in range(test_count):
    matrix_a = np.random.rand(matrix_size, matrix_size)
    matrix_b = np.random.rand(matrix_size, matrix_size)
    matrix_pairs.append([matrix_a, matrix_b])


# CPU Matrix Multiplication
def cpu_matrix_multiplication(matrix_a, matrix_b):
    start_time = time.time()

    # Perform matrix multiplication using CPU
    cpu_result = np.dot(matrix_a, matrix_b)

    end_time = time.time()
    execution_time = end_time - start_time
    return cpu_result, execution_time


# GPU Matrix Multiplication
def gpu_matrix_multiplication_cupy(matrix_a, matrix_b):
    start_time = time.time()

    # Perform matrix multiplication using GPU
    gpu_matrix_a = cp.asarray(matrix_a)
    gpu_matrix_b = cp.asarray(matrix_b)
    gpu_result = cp.dot(gpu_matrix_a, gpu_matrix_b)
    cpu_result = cp.asnumpy(gpu_result)

    end_time = time.time()
    execution_time = end_time - start_time
    return cpu_result, execution_time


def gpu_matrix_multiplication_tensorflow(matrix_a, matrix_b):
    start_time = time.time()

    with tf.device('/GPU:0'):
      gpu_result = tf.matmul(matrix_a, matrix_b)

    end_time = time.time()
    execution_time = end_time - start_time
    return gpu_result, execution_time


def gpu_matrix_multiplication_pytorch(matrix_a, matrix_b):
    start_time = time.time()

    gpu_matrix_a = torch.from_numpy(matrix_a)
    gpu_matrix_b = torch.from_numpy(matrix_b)
```

```
    gpu_result = torch.matmul(gpu_matrix_a, gpu_matrix_b)
    gpu_result = gpu_result.numpy()

    end_time = time.time()
    execution_time = end_time - start_time
    return gpu_result, execution_time


def cpu_test():
    test_result = []

    for i in range(test_count):
        result, execution_time = cpu_matrix_multiplication(matrix_a,
matrix_b)
        # test_result.append([result, execution_time])
        test_result.append(execution_time)

    return test_result


def gpu_test(mul_function):
    test_result = []

    for i in range(test_count):
        result, execution_time = mul_function(matrix_a, matrix_b)
        # test_result.append([result, execution_time])
        test_result.append(execution_time)

    return test_result


# Run CPU Matrix Multiplication
cpu_test_result = cpu_test()

# Run GPU Matrix Multiplication
gpu_test_result_cupy = gpu_test(gpu_matrix_multiplication_cupy)
gpu_test_result_tensorflow =
gpu_test(gpu_matrix_multiplication_tensorflow)
gpu_test_result_pytorch = gpu_test(gpu_matrix_multiplication_pytorch)

# Compare the results (optional)
# print("CPU Result:")
# print(cpu_result)
# print("GPU Result:")
# print(gpu_result)

# Print the execution times
print("CPU Execution Time:              ", cpu_test_result, "seconds")
print("GPU Execution Time (CuPy):       ", gpu_test_result_cupy,
"seconds")
print("GPU Execution Time (TensorFlow): ", gpu_test_result_tensorflow,
```

```
"seconds")
print("GPU Execution Time (PyTorch):    ", gpu_test_result_pytorch,
"seconds")

print("\n")

print("CPU Execution Time Average:              ",
sum(cpu_test_result) / test_count, "seconds")
print("GPU Execution Time Average (CuPy):       ",
sum(gpu_test_result_cupy) / test_count, "seconds")
print("GPU Execution Time Average (TensorFlow): ",
sum(gpu_test_result_tensorflow) / test_count, "seconds")
print("GPU Execution Time Average (PyTorch):    ",
sum(gpu_test_result_pytorch) / test_count, "seconds")
```

```
CPU Execution Time:               [0.09871101379394531,
0.09735655784606934, 0.0996243953704834, 0.09606766700744629,
0.0982978343963623, 0.09641170501708984, 0.09520339965820312,
0.09458136558532715, 0.12127375602722168, 0.09517240524291992,
0.10121512413024902, 0.10441279411315918, 0.09888458251953125,
0.09730553627014160, 0.0970005989074707, 0.09534716606140137,
0.10003113746643066, 0.10153770446777344, 0.0951070785522461,
0.10240030288696289, 0.09217095375061035, 0.11425399780273438,
0.1124114990234375, 0.10115885734558105, 0.08875465393066406,
0.07398223876953125, 0.05490469932556152, 0.052301645278930664,
0.05430030822753906, 0.053090572357177734, 0.05406379699707031,
0.05340075492858887, 0.07061004638671875, 0.051386117935180664,
0.0543673038482666, 0.05651116371154785, 0.05031251907348633,
0.050954580307006836, 0.05486464500427246, 0.06138300895690918,
0.05218815803527832, 0.052088022232055664, 0.05606245994567871,
0.049783945083618164, 0.05771970748901367, 0.05292630195617676,
0.051192522048950195, 0.05141711235046387, 0.05353116989135742,
0.05182957649230957, 0.06685066223144531, 0.07346367835998535,
0.05269336700439453, 0.05279660224914551, 0.052069902420043945,
0.05192756652832031, 0.0525212287902832, 0.05348086357116699,
0.05128073692321777, 0.05064249038696289, 0.055908918380737305,
0.05377769470214844, 0.05416679382324219, 0.05125260353088379,
0.05208420753479004, 0.05173301696777344, 0.05198168754577637,
0.05154776573181152, 0.055028676986694336, 0.04983377456665039,
0.07240080833435059, 0.052542924880981445, 0.053179264068603516,
0.050135135650634766, 0.052718162536621094, 0.05669736862182617,
0.05466508865356445, 0.05357241630554199, 0.05841350555419922,
0.057471513748168945, 0.06479382514953613, 0.05843758583068848,
0.057464599609375, 0.0560300350189209, 0.05282950401306152,
0.053609609603881836, 0.05494832992553711, 0.05363798141479492,
0.06661415100097656, 0.06563711166381836, 0.05481529235839844,
0.051236867904663086, 0.05315113067626953, 0.055211544036865234,
0.05288839340209961, 0.05156588554382324, 0.05143260955810547,
0.05405879020690918, 0.05315256118774414, 0.051189422607421875]
seconds
```

```
GPU Execution Time (CuPy):        [0.042804718017578125,
0.033241987228393555, 0.033051490783691406, 0.033190250396728516,
0.032753705978393555, 0.03343462944030762, 0.03326010704040527,
0.026762008666992188, 0.017055511474609375, 0.016964435577392578,
0.016832590103149414, 0.017238616943359375, 0.01753401756286621,
0.016980409622192383, 0.016764402389526367, 0.017097949981689453,
0.017140626907348633, 0.017071247100830078, 0.01688408851623535,
0.015445470809936523, 0.01486968994140625, 0.014822959899902344,
0.01530313491821289, 0.015447854995727539, 0.015207529067993164,
0.014931678771972656, 0.01484823226928711, 0.014886856079101562,
0.015043973922729492, 0.015076637268066406, 0.01495051383972168,
0.015128135681152344, 0.01469278335571289, 0.015440940856933594,
0.014940500259399414, 0.014953136444091797, 0.014711856842041016,
0.014770269393920898, 0.014489412307739258, 0.01451420783996582,
0.014552593231201172, 0.014675378799438477, 0.014486312866210938,
0.014538764953613281, 0.014719247817993164, 0.01507258415222168,
0.015231609344482422, 0.015084266662597656, 0.014899253845214844,
0.014899253845214844, 0.014706134796142578, 0.014657974243164062,
0.01473689079284668, 0.014575719833374023, 0.014798879623413086,
0.015411853790283203, 0.015560388565063477, 0.01528477668762207,
0.015327215194702148, 0.015248775482177734, 0.015096187591552734,
0.015106916427612305, 0.014928579330444336, 0.01481318473815918,
0.014848709106445312, 0.015042304992675781, 0.014884233474731445,
0.014640092849731445, 0.014782905578613281, 0.014458179473876953,
0.015065193176269531, 0.014825820922851562, 0.014746665954589844,
0.014700651168823242, 0.01458740234375, 0.014841794967651367,
0.014896869659423828, 0.014835357666015625, 0.014754295349121094,
0.015032529830932617, 0.015990734100341797, 0.016225576400756836,
0.01595473289489746, 0.015274763107299805, 0.015071392059326172,
0.015390157699584961, 0.014857769012451172, 0.015058517456054688,
0.014956474304199219, 0.014894962310791016, 0.015061616897583008,
0.014874458312988281, 0.01565265655517578, 0.015540838241577148,
0.015067100524902344, 0.015424489974975586, 0.015395641326904297,
0.015102386474609375, 0.015080690383911133, 0.01568436622619629]
seconds
GPU Execution Time (TensorFlow):  [0.005807161331176758,
0.01381230354309082, 0.013703346252441406, 0.01866936683654785,
0.013746976852416992, 0.013975858688354492, 0.013963460922241211,
0.013779401779174805, 0.013877391815185547, 0.013791084289550781,
0.013789892196655273, 0.013750553131103516, 0.01394510269165039,
0.013712644577026367, 0.013931989669799805, 0.013762474060058594,
0.013773679733276367, 0.013709545135498047, 0.013818740844726562,
0.01365804672241211, 0.013600349426269531, 0.013701438903808594,
0.01385045051574707, 0.013889074325561523, 0.013744354248046875,
0.013877630233764648, 0.013780593872070312, 0.013663053512573242,
0.014180183410644531, 0.013736248016357422, 0.013816595077514648,
0.013824701309204102, 0.01412510871887207, 0.0138092041015625,
0.014119386672973633, 0.013772726058959961, 0.013794183731079102,
0.013720273971557617, 0.013864517211914062, 0.013880252838134766,
```

0.013857603073120117, 0.013728857040405273, 0.01387643814086914,
0.013786077499389648, 0.013795614242553711, 0.017501115798950195,
0.014232397079467773, 0.013961076736450195, 0.013878107070922852,
0.013758182525634766, 0.01365351676940918, 0.013819456100463867,
0.013816595077514648, 0.013982534408569336, 0.013819456100463867,
0.013858795166015625, 0.014267921447753906, 0.015448331832885742,
0.014399528503417969, 0.013683319091796875, 0.014411687850952148,
0.01430201530456543, 0.013245344161987305, 0.01395559310913086,
0.014102697372436523, 0.013968467712402344, 0.01371002197265625,
0.013868093490600586, 0.013796806335449219, 0.013895273208618164,
0.014408111572265625, 0.01366281509399414, 0.013686180114746094,
0.013735055923461914, 0.013670921325683594, 0.013705730438232422,
0.013814687728881836, 0.01369476318359375, 0.01380014419555664,
0.013755083084106445, 0.013725996017456055, 0.013673782348632812,
0.013602256774902344, 0.014431238174438477, 0.013613462448120117,
0.013881206512451172, 0.014000177383422852, 0.013843297958374023,
0.013802289962768555, 0.013910531997680664, 0.013854026794433594,
0.013915300369262695, 0.0138702392578125, 0.01372075080871582,
0.013943195343017578, 0.013887405395507812, 0.013889789581298828,
0.013735294342041016, 0.013895273208618164, 0.013802051544189453]
seconds
GPU Execution Time (PyTorch):   [0.04921913146972656,
0.056212663650512695, 0.05361008644104004, 0.05332636833190918,
0.049344778060913086, 0.04946327209472656, 0.04994463920593262,
0.05076909065246582, 0.05856966972351074, 0.05015230178833008,
0.05054020881652832, 0.05264163017272949, 0.05198979377746582,
0.05014944076538086, 0.05142521858215332, 0.05246448516845703,
0.049672842025756836, 0.04937338829040527, 0.0521395206451416,
0.05150580406188965, 0.051079750061035156, 0.05110430717468262,
0.05008411407470703, 0.04995226860046387, 0.05141854286193848,
0.051390647888183594, 0.0527651309967041, 0.05373358726501465,
0.05605626106262207, 0.0486757755279541, 0.05248832702636719,
0.05005478858947754, 0.04876255989074707, 0.04910421371459961,
0.05103921890258789, 0.05182981491088867, 0.04980826377868652,
0.05074477195739746, 0.04904675483703613, 0.04895734786987305,
0.050015926361083984, 0.05112195014953613, 0.05232501029968262,
0.051727294921875, 0.05028510093688965, 0.050016164779663086,
0.051328182220458984, 0.05077052116394043, 0.05818057060241699,
0.049356698989868164, 0.05160045623779297, 0.05099630355834961,
0.050093889236450195, 0.04850196838378906, 0.0486299991607666,
0.06988930702209473, 0.08226847648620605, 0.09239411354064941,
0.09291625022888184, 0.09224963188171387, 0.09090209007263184,
0.0959620475769043, 0.09961652755737305, 0.09189653396606445,
0.09258413314819336, 0.09265923500061035, 0.0969705581665039,
0.0931086540222168, 0.09415578842163086, 0.09208536148071289,
0.08769440650939941, 0.08885407447814941, 0.09016776084899902,
0.08594059944152832, 0.0924386978149414, 0.08917975425720215,
0.0900413990020752, 0.09216499328613281, 0.09037065505981445,
0.09015846252441406, 0.09164834022521973, 0.09635090827941895,

```
0.08942937850952148, 0.09043741226196289, 0.08413434028625488,
0.09426593780517578, 0.08781266212463379, 0.08922672271728516,
0.09241461753845215, 0.0893704891204834, 0.08874130249023438,
0.09084749221801758, 0.0906991958618164, 0.08790326118469238,
0.08696913719177246, 0.08961081504821777, 0.08996391296386719,
0.08700037002563477, 0.08161187171936035, 0.06493115425109863] seconds


CPU Execution Time Average:                0.0662940812110901 seconds
GPU Execution Time Average (CuPy):         0.016724536418914793 seconds
GPU Execution Time Average (TensorFlow):   0.013870413303375245 seconds
GPU Execution Time Average (PyTorch):      0.0684559965133667 seconds
```

In this scenario we'll compare exection time of CPU (based on numpy), and 3 GPU accelarated methods based on CuPy, TensorFlow and PyTorch for matix multiplication.

For a matrix muliplication of two matrices of size N x N, it is required to perform an order of N^3 operations, in this test N = 1000 so we expect to perform an order of 10^9 operations, we also perform 100 tests to average out execution times.

As the output shows that exection times on the GPU is generally faster because it is desigend to accelarate vector operations and runs many cores to maximize parallization.

Also we observe that:

- TensorFlow is fastest as it runs all the executions on the GPU without relying on system RAM.
- CuPy and NumPy results are close as CuPy implements NumPy on CUDA but still uses system RAM.
- PyTorch is faster than CuPy but not by much because this library doesn't directly support numpy arrays, we have to convert back and fourth numpy memory to it's "torch" matrix type.

```
""" Code Definition to use:
In this code, we start by loading an image for processing using the
cv2.imread function.
Then, we define two functions: cpu_image_processing and
gpu_image_processing,
which perform image processing operations using CPU and GPU,
respectively.

In the example, we convert the image to grayscale using the
cv2.cvtColor function
for both CPU and GPU processing.

The cpu_image_processing function measures the execution time using
the time module.
Similarly, the gpu_image_processing function measures the execution
time
but with GPU-accelerated operations using the OpenCV CUDA module.
```

```
After running the CPU and GPU image processing functions,
the results are displayed using cv2.imshow, and the execution times
are printed.

Please note that for this code to work, you need to have OpenCV
installed with CUDA support.
Additionally, you may need to modify the image processing operations
based on
your specific requirements.

Remember to replace "path_to_your_image.jpg" with the actual path to
your image file
"""

import cv2
import numpy as np
import time
import cupy as cp

# Load an image for processing
image_paths = [
    "drive/MyDrive/3.jpg",
    "drive/MyDrive/1.jpg",
    "drive/MyDrive/bigImage.png"
]

test_count = 50

# CPU Image Processing
def cpu_image_processing(image):
    start_time = time.time()

    # Perform image processing operations using CPU
    # Example: Convert the image to grayscale
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    end_time = time.time()
    execution_time = end_time - start_time
    return gray_image, execution_time

def cpu_image_processing_numpy(image):
    start_time = time.time()

    second = np.array([0.299, 0.587, 0.114])
    gray_image = np.dot(image, second)


    end_time = time.time()

    # end_time = time.time()
```

```
    execution_time = end_time - start_time
    return gray_image, execution_time

# GPU Image Processing
def gpu_image_processing_cv2_cuda(image):
    start_time = time.time()

    # Perform image processing operations using GPU
    # Example: Convert the image to grayscale
    gpu_image = cv2.cuda_GpuMat()
    gpu_image.upload(image)
    gpu_gray_image = cv2.cuda.cvtColor(gpu_image, cv2.COLOR_BGR2GRAY)
    gray_image = gpu_gray_image.download()

    end_time = time.time()
    execution_time = end_time - start_time
    return gray_image, execution_time


def gpu_image_processing_cupy(image):
    # start_time = time.time()

    # gray_image = np.dot(image, [0.2989, 0.5870,
0.1140]).astype(np.uint8)
    gpu_image = cp.asarray(image)
    second = cp.asarray([0.299, 0.587, 0.114])

    start_time = time.time()


    gpu_result = cp.dot(gpu_image, second)
    x = gpu_result[0][0]

    end_time = time.time()


    gray_image = cp.asnumpy(gpu_result)

    # end_time = time.time()

    execution_time = end_time - start_time
    return gray_image, execution_time


def cpu_test(image):
    test_result = []

    for i in range(test_count):
        result, execution_time = cpu_image_processing(image)
        test_result.append(execution_time)
```

```
    return test_result

def cpu_test_numpy(image):
    test_result = []

    for i in range(test_count):
        result, execution_time = cpu_image_processing_numpy(image)
        test_result.append(execution_time)

    return test_result

def gpu_test_cupy(image):
    test_result = []

    for i in range(test_count):
        result, execution_time = gpu_image_processing_cupy(image)
        test_result.append(execution_time)

    return test_result

def gpu_test_cuda(image):
    test_result = []

    for i in range(test_count):
        result, execution_time = gpu_image_processing_cv2_cuda(image)
        test_result.append(execution_time)

    return test_result

for image_path in image_paths:
  print("Testing image ", image_path)

  image = cv2.imread(image_path)

  print("Image size is ", image.shape)

  # Run CPU Image Processing
  # cpu_result, cpu_execution_time = cpu_image_processing(image)
  cpu_test_result = cpu_test(image)
  cpu_test_result_numpy = cpu_test_numpy(image)

  # Run GPU Image Processing
  # gpu_result, gpu_execution_time = gpu_image_processing_2(image)
  gpu_test_result_cupy = gpu_test_cupy(image)
  # gpu_test_result_cuda = gpu_test_cuda(image)

  # print(cpu_result)
  # print(gpu_result)

  # Display the results and execution times
```

```
  # cv2.imshow("CPU Result", cpu_result)
  # cv2.imshow("GPU Result", gpu_result)
  # cv2.waitKey(0)

  print("CPU Execution Time:              ", cpu_test_result,
"seconds")
  print("CPU Execution Time (numPy):      ", cpu_test_result_numpy,
"seconds")
  print("GPU Execution Time (CuPy):       ", gpu_test_result_cupy,
"seconds")
  # print("GPU Execution Time (CV2_CUDA):   ", gpu_test_result_cuda,
"seconds")

  print("\n")

  print("CPU Execution Time Average:              ",
sum(cpu_test_result) / test_count, "seconds")
  print("CPU Execution Time Average (NumPy):      ",
sum(cpu_test_result_numpy) / test_count, "seconds")
  print("GPU Execution Time Average (CuPy):       ",
sum(gpu_test_result_cupy) / test_count, "seconds")
  # print("GPU Execution Time Average (CUDA):       ",
sum(gpu_test_result_cuda) / test_count, "seconds")

  print("##################\n\n\n")


Testing image  drive/MyDrive/3.jpg
Image size is  (960, 640, 3)
CPU Execution Time:               [0.0004837512969970703,
0.00024080276489257812, 0.00025463104248046875, 0.0002357959747314453,
0.00023317337036132812, 0.00023293495178222656,
0.00025081634521484375, 0.000240325927734375, 0.00026822090014892578,
0.00026082992553710938, 0.00026082992553710938, 0.00024867057800292969,
0.00024914741516113281, 0.00025868415832519531, 0.00024580955505371094,
0.00023365020751953125, 0.00023365020751953125,
0.00023245811462402344, 0.00024461746215820312, 0.00025463104248046875,
0.00029754638671875, 0.00027680397033691406, 0.00028395652770996094,
0.00028276443481445312, 0.00027680397033691406, 0.00027608871459960938,
0.00031208992004394531, 0.00028467178344726562, 0.00026655197143554688,
0.00027394294738769531, 0.00026750564575195312, 0.00027394294738769531,
0.00028800964355468750, 0.00028681755065917969, 0.00027751922607421875,
0.00027537345886230469, 0.00028729438781738281, 0.00030517578125,
0.00027489662170410156, 0.00026535987854003906,
0.00027370452880859375, 0.00026631355285644531, 0.00026512145996093750,
0.00025677680969238281, 0.00024986267089843750, 0.00024890899658203125,
0.00024962425231933594, 0.00026035308837890625, 0.00028610229492187500,
0.00024795532226562500] seconds
CPU Execution Time (numPy):       [0.018322467803955078,
0.015343189239501953, 0.013334512710571289, 0.015308618545532227,
```

```
0.014032363891601562, 0.013492822647094727, 0.01359248161315918,
0.013393402099609375, 0.014092683792114258, 0.012954235076904297,
0.013316631317138672, 0.017778873443603516, 0.013138294219970703,
0.013826131820678711, 0.013110160827636719, 0.013442754745483398,
0.013451099395751953, 0.013214826583862305, 0.013269662857055664,
0.012900829315185547, 0.012982368469238281, 0.013188838958740234,
0.01352691650390625, 0.01562952995300293, 0.013599157333374023,
0.013109922409057617, 0.0128326416015625, 0.012619733810424805,
0.01246500015258789, 0.012707710266113281, 0.013146400451660156,
0.012461662292480469, 0.012413978576660156, 0.012712240219116211,
0.013653993606567383, 0.013010740280151367, 0.012578248977661133,
0.012991905212402344, 0.013141155242919922, 0.01353597640991211,
0.01307225227355957, 0.013234853744506836, 0.01325225830078125,
0.013077974319458008, 0.01265716552734375, 0.012362957000732422,
0.012907266616821289, 0.013239145278930664, 0.012555837631225586,
0.013072967529296875] seconds
GPU Execution Time (CuPy):        [0.0002467632293701172,
0.00017499923706054688, 0.0001533031463623047, 0.00011658668518066406,
0.00013136863708496094, 0.0001633167266845703, 0.00012969970703125,
0.00012183189392089844, 0.00016427040100097656, 0.00018429756164550781,
0.00016880035400390625, 0.00018310546875, 0.00012469291687011719,
0.00017356872558593750, 0.00011968612670898438, 0.00011563301086425781,
0.00012683868408203125, 0.00013256072998046875,
0.00016808509826660156, 0.00012993812561035156,
0.00013065338134765625, 0.00015163421630859375, 0.00016021728515625,
0.00013566017150878906, 0.00014948844909667969, 0.00015521049499511719,
0.00012636184692382812, 0.00014400482177734375,
0.00013637542724609375, 0.00014138221740722656,
0.00012135505676269531, 0.00013208389282226562, 0.00013089179992675781,
0.00011229515075683594, 0.00011181831359863281,
0.00011467933654785156, 0.00011897087097167969,
0.00011157989501953125, 0.00011873245239257812, 0.00012707710266113281,
0.00011563301086425781, 0.00015044212341308594,
0.00011467933654785156, 0.00011229515075683594,
0.00012874603271484375, 0.00011849403381347656,
0.00012803077697753906, 0.00011396408081054688, 0.00011253356933593750,
0.00011134147644042969] seconds


CPU Execution Time Average:                0.00026802539825439455
seconds
CPU Execution Time Average (NumPy):        0.013461136817932129 seconds
GPU Execution Time Average (CuPy):         0.00013731956481933595
seconds
##################



Testing image  drive/MyDrive/1.jpg
Image size is  (6000, 4000, 3)
```

```
CPU Execution Time:               [0.010320425033569336,
0.01352548599243164, 0.009873390197753906, 0.009677410125732422,
0.009523153305053711, 0.009596824645996094, 0.009598970413208008,
0.009918451309204102, 0.009839534759521484, 0.009718894958496094,
0.009520530700683594, 0.009611129760742188, 0.009360074996948242,
0.00956273078918457, 0.009704351425170898, 0.009519338607788086,
0.009520530700683594, 0.009753227233886719, 0.00941920280456543,
0.009848594665527344, 0.013386964797973633, 0.010449647903442383,
0.012134313583374023, 0.012701034545898438, 0.012241601943969727,
0.0161898136138916, 0.009701728820800781, 0.0096282958984375,
0.009606361389160156, 0.010565757751464844, 0.009540557861328125,
0.009436845779418945, 0.010094165802001953, 0.009314775466918945,
0.00950002670288086, 0.009337663650512695, 0.009387731552124023,
0.009239673614501953, 0.009673833847045898, 0.009301900863647461,
0.009242534637451172, 0.009071826934814453, 0.00924229621887207,
0.009454488754272461, 0.009346723556518555, 0.009688854217529297,
0.009435653686523438, 0.009684562683105469, 0.010188579559326172,
0.00973653793334961] seconds
CPU Execution Time (numPy):       [1.2404065132141113,
1.2052619457244873, 1.1965868473052979, 0.6943144798278809,
0.6387655735015869, 0.6546657085418701, 0.6407787799835205,
0.6319684982299805, 0.634521484375, 0.6425790786743164,
0.6312870979309082, 0.6578996181488037, 0.6438045501708984,
0.6283817291259766, 0.6390872001647949, 0.6306643486022949,
0.6496305465698242, 0.6389386653900146, 0.7007262706756592,
1.1730105876922607, 1.1816496849060059, 1.1843676567077637,
0.67336106300354, 0.64725661277771, 0.6435916423797607,
0.6339406967163086, 0.6369912624359131, 0.6388318538665771,
0.635810375213623, 0.6395580768585205, 0.6352343559265137,
0.6370601654052734, 0.6455004215240479, 0.6296508312225342,
0.6375281810760498, 0.6524209976196289, 0.6349365711212158,
0.6910498142242432, 1.2017083168029785, 1.1969304084777832,
1.22328782081604, 0.6698343753814697, 0.6390764713287354,
0.6295530796051025, 0.6355280876159668, 0.6325819492340088,
0.6486170291900635, 0.6393246650695801, 0.6280150413513184,
0.6476376056671143] seconds
GPU Execution Time (CuPy):        [0.0002815723419189453,
0.0002593994140625, 0.0002474784851074219, 0.0002498626708984375,
0.0002357959747314453, 0.0002579689025878906, 0.0002694129943847656,
0.0002450942993164062, 0.0002293586730957031, 0.0002913475036621094,
0.0002541542053222656, 0.0002315044403076172, 0.0002233982086181641,
0.0002076625823974609, 0.0002324581146240234,
0.0002386569976806641, 0.0002260208129882812,
0.0002398490905761719, 0.0002381801605224609,
0.0002748966217041016, 0.0002350807189941406, 0.0002651214599609375,
0.0002892017364501953, 0.0002233982086181641, 0.0002732276916503906,
0.0002882480621337891, 0.0003032684326171875, 0.0003063678741455078,
0.0002739429473876953, 0.0002994537353515625, 0.0003399848937988281,
0.0002906322479248047, 0.00029754638671875, 0.0003392696380615234,
```

```
0.0003330707550048828, 0.0003132820129394531, 0.00036406517028808594,
0.0003182888080310058594, 0.00031113624572753906, 0.0003204345703125,
0.0003483295440673828, 0.0002586841583251953, 0.0002658367156982422,
0.0002343654632568359, 0.0002238750457763672, 0.0002951622009277344,
0.0002307891845703125, 0.0002505779266357422, 0.0002522468566894531,
0.0002567768096923828] seconds


CPU Execution Time Average:                0.010078740119934083 seconds
CPU Execution Time Average (NumPy):        0.7448822927474975 seconds
GPU Execution Time Average (CuPy):         0.0002707147598266601 6
seconds
#################



Testing image  drive/MyDrive/bigImage.png
Image size is  (10000, 10000, 3)
CPU Execution Time:                [0.054454803466796875,
0.053730010986328125, 0.05409812927246094, 0.05743837356567383,
0.05375075340270996, 0.05382728576660156, 0.05749368667602539,
0.05714297294616699, 0.05431842803955078, 0.0718533992767334,
0.06469297409057617, 0.05421090126037598, 0.054338932037353516,
0.055709123611450195, 0.05343437194824219, 0.05357766151428223,
0.05344390869140625, 0.05347037315368652, 0.053441762924194336,
0.054021596908569336, 0.05364274978637695, 0.05433201789855957,
0.05309748649597168, 0.05507826805114746, 0.05672645568847656,
0.05377960205078125, 0.05336761474609375, 0.05356407165527344,
0.05459856986999512, 0.06407880783081055, 0.05537295341491699,
0.05391240119934082, 0.05476975440979004, 0.05446815490722656,
0.05388283729553223, 0.053054094314575195, 0.053350210189819336,
0.05393099784851074, 0.05413937568664551, 0.054221153259277344,
0.05417752265930176, 0.05462360382080078, 0.057858943939208984,
0.05472946166992187 5, 0.05555415153503418, 0.05489301681518555,
0.05391287803649902, 0.05612540245056152, 0.05967378616333008,
0.05393838882446289] seconds
CPU Execution Time (numPy):        [2.636632204055786,
3.6241862773895264, 3.304150342941284, 2.6714730262756348,
2.656409978866577, 2.628725051879883, 4.665302991867065,
2.624660015106201, 2.6395668983459473, 2.6524064540863037,
3.4009242057800293, 3.528449773788452, 2.625739812850952,
2.669175386428833, 2.622297525405884, 4.610863447189331,
2.661916494369507, 2.625025510787964, 2.6114580631256104,
3.0124826431274414, 3.9231226444244385, 2.646136999130249,
2.617927312850952, 2.6517996788024902, 4.232942819595337,
2.7640609741210938, 2.6517484188079834, 2.6518166065216064,
2.6242470741271973, 4.339846849441528, 2.6915066242218018,
2.6542580127716064, 2.6395046710968018, 3.8545444011688232,
3.0828258991241455, 2.6558010578155518, 2.625636100769043,
2.6293606758117676, 4.561779022216797, 2.653968572616577,
```

```
2.6371259689331055, 2.651585340499878, 3.561891555786133,
3.3900444507598877, 2.643742322921753, 2.6546828746795654,
2.668896198272705, 4.6415557861328125, 2.6661808490753174,
2.6749796867370605] seconds
GPU Execution Time (CuPy):        [0.002232789993286133,
0.0002627372741699219, 0.0002672672271728515 6, 0.0002391338348388672,
0.0002777576446533203, 0.0002789497375488281, 0.0002825260162353515 6,
0.0003163814544677734 4, 0.0002844333648681640 6,
0.0002977848052978515 6, 0.0004167556762695312 5,
0.0002496242523193359 4, 0.0002481937408447265 6,
0.0002946853637695312 5, 0.0003139972686767578, 0.0002760887145996094,
0.0002396106719970703, 0.0002536773681640625, 0.00025177001953125,
0.0003345012664794922, 0.0003519058227539062 5, 0.0002591609954833984 4,
0.0003132820129394531, 0.0002574920654296875, 0.0003023147583007812 5,
0.0002949237823486328, 0.0003459453582763672, 0.0003561973571777344,
0.0002663135528564453, 0.0002737045288085937 5, 0.0002703666687011719,
0.0002508163452148437 5, 0.0003180503845214844, 0.0002655982971191406,
0.0002567768096923828, 0.0002701282501220703, 0.000240325927734375,
0.0002653598785400390 6, 0.0002405643463134765 6, 0.0002760887145996094,
0.0002686976385498046 9, 0.0002410411834716797, 0.0003111362457275390 6,
0.0003244876861572265 6, 0.0002815723419189453, 0.0003199577331542969,
0.0002372264862060547, 0.0002367496490478515 6, 0.0002462863922119140 6,
0.00029754638671875] seconds


CPU Execution Time Average:               0.05534608364105224 seconds
CPU Execution Time Average (NumPy):       3.0363073110580445 seconds
GPU Execution Time Average (CuPy):        0.0003211736679077148
seconds
#################
```

In this scenario we'll compare exection time of CPU (based on OpenCV), and 2 GPU accelarated methods based on OpenCV on CUDA and Cupy for transformation of RGB image to Grayscale.

Note that the readme for this scenario proposed to use the PIL but as it stands this library doesn't support GPU accelarion. Also OpenCV on CUDA doesn't work on google colab and I wasn't able to compile it for my personal system either.

According to OpenCV documentaion, transforming RGB images to Grayscale can be done by simply calculating the sum of multiplication of an scaler into color each channel for every single pixel.

As the output shows that exection times on the GPU is generally faster because it is desigend to accelarate vector operations and runs many cores to maximize parallization, but as noted in the previous scenario CuPy execution times isn't signifactly faster than CPU execution times.

Note that according to internet research running the same operation via OpenCV on CUDA is abou 300% faster than CPU.