# GPU Lab

AbdelRahman Adel AbdelFattah - 17012296

**Colab Link**

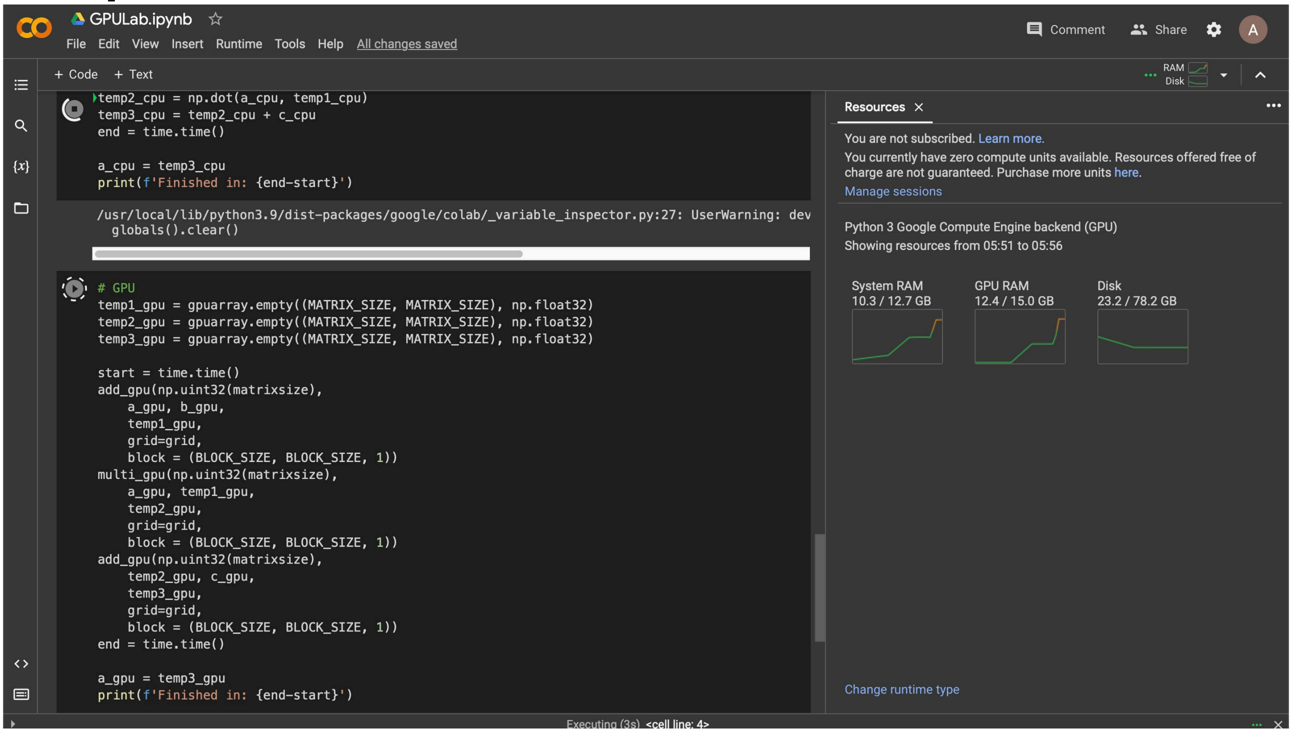
## Code

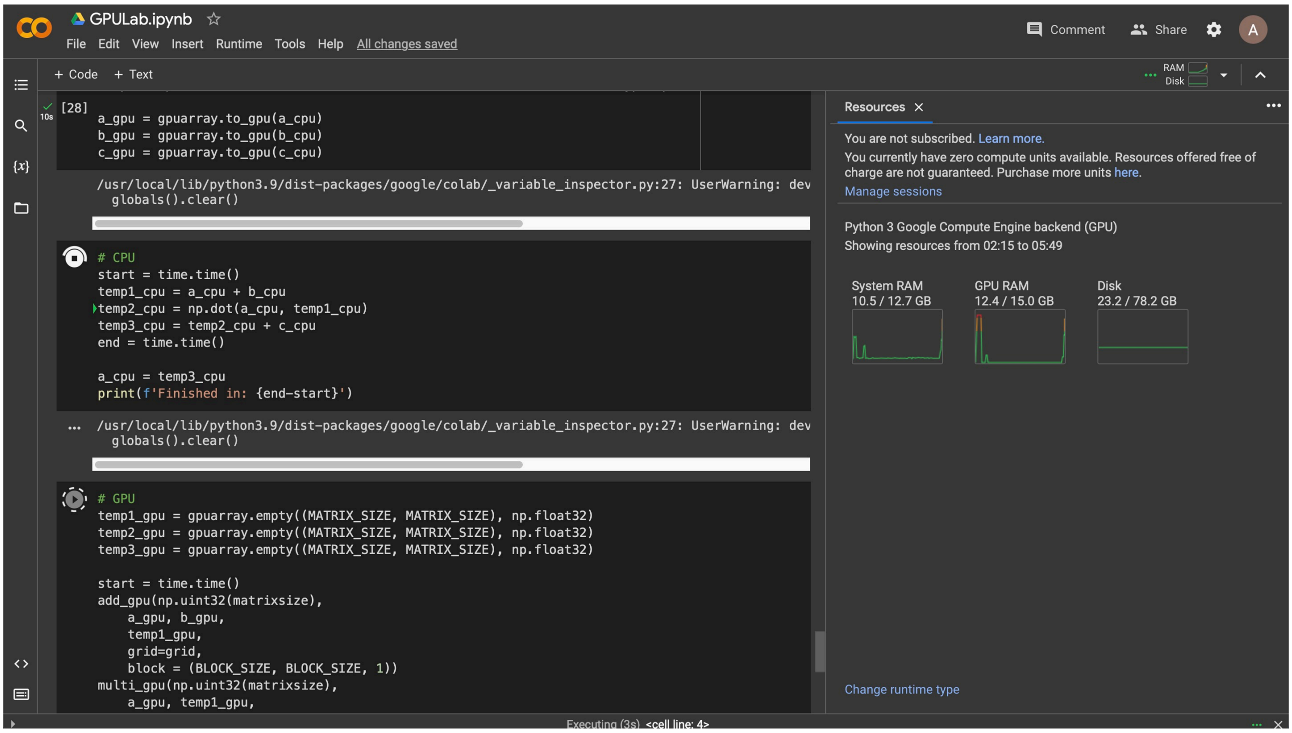
```
import numpy as np
from pycuda import compiler, gpuarray, tools
import pycuda.driver as cuda
import pycuda.autoinit
import time
kernel_code = """
__global__ void multi_gpu(int matrixsize,float *a, float *b, float
*c)
{
    int tx = blockDim.x*blockIdx.x + threadIdx.x;
    int ty = blockDim.y*blockIdx.y + threadIdx.y;

    if((ty <matrixsize) && (tx < matrixsize)){
      float Pvalue = 0;
      for(int k=0; k<matrixsize;++k){
        float Aelement = a[ty*matrixsize +k];
        float Belement = b[k*matrixsize +tx];
        Pvalue += Aelement * Belement;
      }
      c[ty * matrixsize + tx] = Pvalue;
    }
}

__global__ void add_gpu(int matrixsize,float *a, float *b, float
*c)
{
    int tx = blockDim.x*blockIdx.x + threadIdx.x;
    int ty = blockDim.y*blockIdx.y + threadIdx.y;

    if((ty <matrixsize) && (tx < matrixsize)){
      float Aelement = a[ty * matrixsize + tx];
      float Belement = b[ty * matrixsize + tx];
      c[ty * matrixsize + tx] = Aelement + Belement;
    }
}
"""
def add_cpu(matrix_a, matrix_b):
  result = np.zeros((matrix_a.shape[0], matrix_a.shape[1]))
  for i in range(matrix_a.shape[0]):
    for j in range(matrix_a.shape[1]):
      result[i][j] = matrix_a[i][j] + matrix_b[i][j]
  return result
```

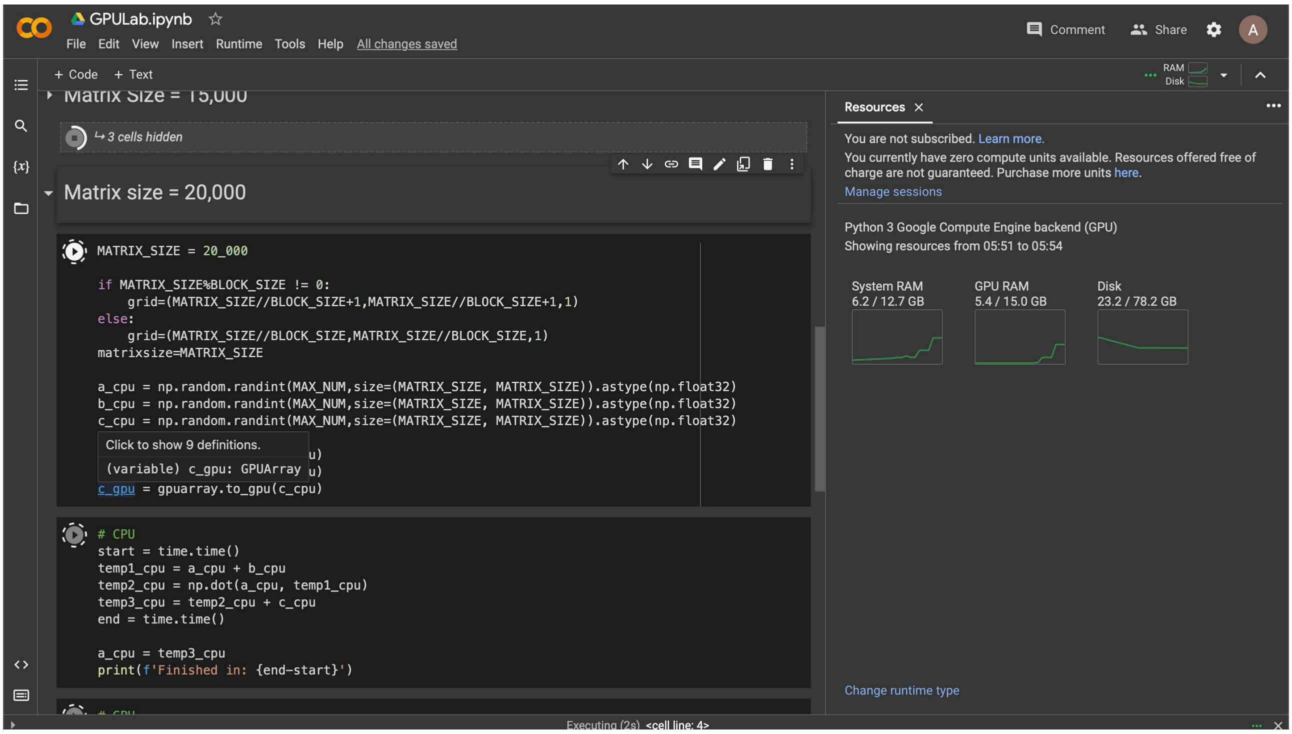
```
def multi_cpu(matrix_a, matrix_b):
  result = np.zeros((matrix_a.shape[0], matrix_a.shape[1]))
  for i in range(matrix_a.shape[0]):
    for j in range(matrix_b.shape[1]):
      for k in range(matrix_b.shape[0]):
        result[i][j] += matrix_a[i][k] * matrix_b[k][j];
  return result
mod = compiler.SourceModule(kernel_code)

multi_gpu = mod.get_function("multi_gpu")
add_gpu = mod.get_function("add_gpu")

BLOCK_SIZE = 32
MAX_NUM = 1024

MATRIX_SIZE = 3
if MATRIX_SIZE%BLOCK_SIZE != 0:
    grid=(MATRIX_SIZE//BLOCK_SIZE+1,MATRIX_SIZE//BLOCK_SIZE+1,1)
else:
    grid=(MATRIX_SIZE//BLOCK_SIZE,MATRIX_SIZE//BLOCK_SIZE,1)
matrixsize=MATRIX_SIZE

a_cpu = np.random.randint(MAX_NUM,size=(MATRIX_SIZE,
MATRIX_SIZE)).astype(np.float32)
b_cpu = np.random.randint(MAX_NUM,size=(MATRIX_SIZE,
MATRIX_SIZE)).astype(np.float32)
c_cpu = np.random.randint(MAX_NUM,size=(MATRIX_SIZE,
MATRIX_SIZE)).astype(np.float32)

a_gpu = gpuarray.to_gpu(a_cpu)
b_gpu = gpuarray.to_gpu(b_cpu)
c_gpu = gpuarray.to_gpu(c_cpu)

MATRIX_SIZE = 3
if MATRIX_SIZE%BLOCK_SIZE != 0:
    grid=(MATRIX_SIZE//BLOCK_SIZE+1,MATRIX_SIZE//BLOCK_SIZE+1,1)
else:
    grid=(MATRIX_SIZE//BLOCK_SIZE,MATRIX_SIZE//BLOCK_SIZE,1)
matrixsize=MATRIX_SIZE

a_cpu = np.random.randint(MAX_NUM,size=(MATRIX_SIZE,
MATRIX_SIZE)).astype(np.float32)
b_cpu = np.random.randint(MAX_NUM,size=(MATRIX_SIZE,
MATRIX_SIZE)).astype(np.float32)
c_cpu = np.random.randint(MAX_NUM,size=(MATRIX_SIZE,
MATRIX_SIZE)).astype(np.float32)

a_gpu = gpuarray.to_gpu(a_cpu)
b_gpu = gpuarray.to_gpu(b_cpu)
c_gpu = gpuarray.to_gpu(c_cpu)

# CPU
```

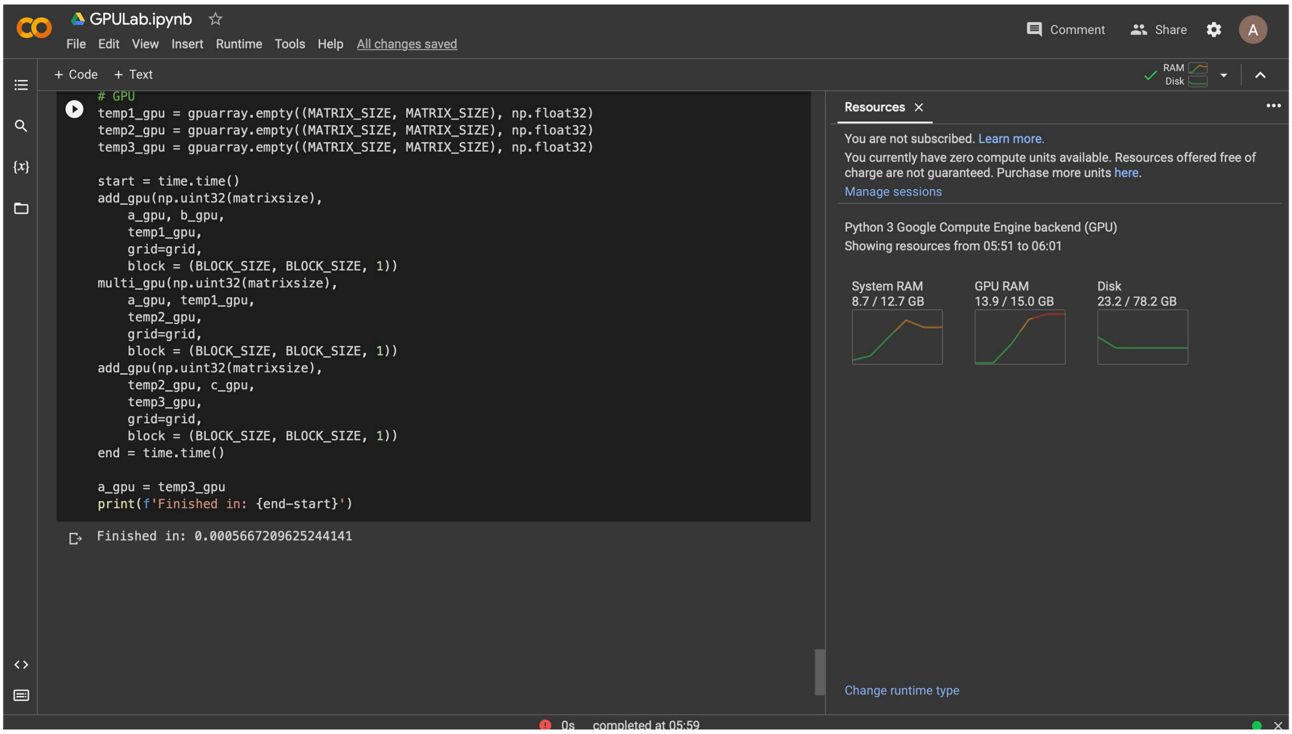
```python
start = time.time()
temp1_cpu = add_cpu(a_cpu, b_cpu)
temp2_cpu = multi_cpu(a_cpu, temp1_cpu)
temp3_cpu = add_cpu(temp2_cpu, c_cpu)
end = time.time()

a_cpu = temp3_cpu
print(f'Finished in: {end-start}')

# GPU
temp1_gpu = gpuarray.empty((MATRIX_SIZE, MATRIX_SIZE), np.float32)
temp2_gpu = gpuarray.empty((MATRIX_SIZE, MATRIX_SIZE), np.float32)
temp3_gpu = gpuarray.empty((MATRIX_SIZE, MATRIX_SIZE), np.float32)

start = time.time()
add_gpu(np.uint32(matrixsize),
    a_gpu, b_gpu,
    temp1_gpu,
    grid=grid,
    block = (BLOCK_SIZE, BLOCK_SIZE, 1))
multi_gpu(np.uint32(matrixsize),
    a_gpu, temp1_gpu,
    temp2_gpu,
    grid=grid,
    block = (BLOCK_SIZE, BLOCK_SIZE, 1))
add_gpu(np.uint32(matrixsize),
    temp2_gpu, c_gpu,
    temp3_gpu,
    grid=grid,
    block = (BLOCK_SIZE, BLOCK_SIZE, 1))
end = time.time()

a_gpu = temp3_gpu
print(f'Finished in: {end-start}')
```

# Sample Runs


GPULab.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
RAM
Disk
temp2_cpu = np.dot(a_cpu, temp1_cpu)
temp3_cpu = temp2_cpu + c_cpu
end = time.time()
a_cpu = temp3_cpu
print(f'Finished in: {end-start}')
/usr/local/lib/python3.9/dist-packages/google/colab/_variable_inspector.py:27: UserWarning: dev
globals().clear()
# GPU
temp1_gpu = gpuarray.empty((MATRIX_SIZE, MATRIX_SIZE), np.float32)
temp2_gpu = gpuarray.empty((MATRIX_SIZE, MATRIX_SIZE), np.float32)
temp3_gpu = gpuarray.empty((MATRIX_SIZE, MATRIX_SIZE), np.float32)
start = time.time()
add_gpu(np.uint32(matrixsize),
a_gpu, b_gpu,
temp1_gpu,
grid=grid,
block = (BLOCK_SIZE, BLOCK_SIZE, 1))
multi_gpu(np.uint32(matrixsize),
a_gpu, temp1_gpu,
temp2_gpu,
grid=grid,
block = (BLOCK_SIZE, BLOCK_SIZE, 1))
add_gpu(np.uint32(matrixsize),
temp2_gpu, c_gpu,
temp3_gpu,
grid=grid,
block = (BLOCK_SIZE, BLOCK_SIZE, 1))
end = time.time()
a_gpu = temp3_gpu
print(f'Finished in: {end-start}')
Resources
You are not subscribed. Learn more.
You currently have zero compute units available. Resources offered free of charge are not guaranteed. Purchase more units here.
Manage sessions
Python 3 Google Compute Engine backend (GPU)
Showing resources from 05:51 to 05:56
System RAM
10.3 / 12.7 GB
GPU RAM
12.4 / 15.0 GB
Disk
23.2 / 78.2 GB
Change runtime type
Executing (3s) <cell line: 4>


GPULab.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
RAM
Disk
[28]
10s
a_gpu = gpuarray.to_gpu(a_cpu)
b_gpu = gpuarray.to_gpu(b_cpu)
c_gpu = gpuarray.to_gpu(c_cpu)
/usr/local/lib/python3.9/dist-packages/google/colab/_variable_inspector.py:27: UserWarning: dev
globals().clear()
# CPU
start = time.time()
temp1_cpu = a_cpu + b_cpu
temp2_cpu = np.dot(a_cpu, temp1_cpu)
temp3_cpu = temp2_cpu + c_cpu
end = time.time()
a_cpu = temp3_cpu
print(f'Finished in: {end-start}')
/usr/local/lib/python3.9/dist-packages/google/colab/_variable_inspector.py:27: UserWarning: dev
globals().clear()
# GPU
temp1_gpu = gpuarray.empty((MATRIX_SIZE, MATRIX_SIZE), np.float32)
temp2_gpu = gpuarray.empty((MATRIX_SIZE, MATRIX_SIZE), np.float32)
temp3_gpu = gpuarray.empty((MATRIX_SIZE, MATRIX_SIZE), np.float32)
start = time.time()
add_gpu(np.uint32(matrixsize),
a_gpu, b_gpu,
temp1_gpu,
grid=grid,
block = (BLOCK_SIZE, BLOCK_SIZE, 1))
multi_gpu(np.uint32(matrixsize),
a_gpu, temp1_gpu,
Resources
You are not subscribed. Learn more.
You currently have zero compute units available. Resources offered free of charge are not guaranteed. Purchase more units here.
Manage sessions
Python 3 Google Compute Engine backend (GPU)
Showing resources from 02:15 to 05:49
System RAM
10.5 / 12.7 GB
GPU RAM
12.4 / 15.0 GB
Disk
23.2 / 78.2 GB
Change runtime type
Executing (3s) <cell line: 4>


GPULab.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
RAM
Disk
Matrix Size = 15,000
↳ 3 cells hidden
Matrix size = 20,000
MATRIX_SIZE = 20_000
if MATRIX_SIZE%BLOCK_SIZE != 0:
grid=(MATRIX_SIZE//BLOCK_SIZE+1,MATRIX_SIZE//BLOCK_SIZE+1,1)
else:
grid=(MATRIX_SIZE//BLOCK_SIZE,MATRIX_SIZE//BLOCK_SIZE,1)
matrixsize=MATRIX_SIZE
a_cpu = np.random.randint(MAX_NUM,size=(MATRIX_SIZE, MATRIX_SIZE)).astype(np.float32)
b_cpu = np.random.randint(MAX_NUM,size=(MATRIX_SIZE, MATRIX_SIZE)).astype(np.float32)
c_cpu = np.random.randint(MAX_NUM,size=(MATRIX_SIZE, MATRIX_SIZE)).astype(np.float32)
Click to show 9 definitions.
(variable) c_gpu: GPUArray
c_gpu = gpuarray.to_gpu(c_cpu)
# CPU
start = time.time()
temp1_cpu = a_cpu + b_cpu
temp2_cpu = np.dot(a_cpu, temp1_cpu)
temp3_cpu = temp2_cpu + c_cpu
end = time.time()
a_cpu = temp3_cpu
print(f'Finished in: {end-start}')
Resources
You are not subscribed. Learn more.
You currently have zero compute units available. Resources offered free of charge are not guaranteed. Purchase more units here.
Manage sessions
Python 3 Google Compute Engine backend (GPU)
Showing resources from 05:51 to 05:54
System RAM
6.2 / 12.7 GB
GPU RAM
5.4 / 15.0 GB
Disk
23.2 / 78.2 GB
Change runtime type
Executing (2s) <cell line: 4>


GPULab.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
RAM
Disk
# GPU
temp1_gpu = gpuarray.empty((MATRIX_SIZE, MATRIX_SIZE), np.float32)
temp2_gpu = gpuarray.empty((MATRIX_SIZE, MATRIX_SIZE), np.float32)
temp3_gpu = gpuarray.empty((MATRIX_SIZE, MATRIX_SIZE), np.float32)
start = time.time()
add_gpu(np.uint32(matrixsize),
a_gpu, b_gpu,
temp1_gpu,
grid=grid,
block = (BLOCK_SIZE, BLOCK_SIZE, 1))
multi_gpu(np.uint32(matrixsize),
a_gpu, temp1_gpu,
temp2_gpu,
grid=grid,
block = (BLOCK_SIZE, BLOCK_SIZE, 1))
add_gpu(np.uint32(matrixsize),
temp2_gpu, c_gpu,
temp3_gpu,
grid=grid,
block = (BLOCK_SIZE, BLOCK_SIZE, 1))
end = time.time()
a_gpu = temp3_gpu
print(f'Finished in: {end-start}')
Finished in: 0.0005667209625244141
Resources
You are not subscribed. Learn more.
You currently have zero compute units available. Resources offered free of charge are not guaranteed. Purchase more units here.
Manage sessions
Python 3 Google Compute Engine backend (GPU)
Showing resources from 05:51 to 06:01
System RAM
8.7 / 12.7 GB
GPU RAM
13.9 / 15.0 GB
Disk
23.2 / 78.2 GB
Change runtime type
0s completed at 05:59

# Speed Comparisons
in seconds

Table 1

| Matrix Size | CPU | GPU |
|---|---|---|
| **3** | 0.00377678871154785 | 0.0012199878692627 |
| **10** | 0.000234603881835938 | 0.000370502471923828 |
| **100** | 0.00583577156066895 | 0.000510215759277344 |
| **1000** | 0.0613725185394287 | 0.000532388687133789 |
| **5000** | 3.59896087646484 | 0.000474929809570313 |
| **10000** | 30.3431296348572 | 0.000492095947265625 |
| **15000** | 91.3707048892975 | 0.000566720962524414 |

# Code explanation
- Kernel code: creates two functions that calculates the multiply and addition of matrix parallelized depending on block size and grid size
- Two functions to use for cpu calculation
- Compile the kernel of CUDA
- Referencing the two kernel functions
- Setting the grid size.
- Initialize the matrices
- Calculate CPU and GPU and see the time of each.

# References

- http://homepages.math.uic.edu/~jan/mcs507/gpuacceleration.pdf
- https://documen.tician.de/pycuda/array.html#module-pycuda.gpuarray
- https://ecatue.gitlab.io/gpu2018/pages/Cookbook/matrix_multiplication_cuda.html
- https://shephexd.github.io/development/2017/02/19/pycuda.html
- https://vitalitylearning.medium.com/a-short-notice-on-performing-matrix-multiplications-in-pycuda-cbfb00cf1450