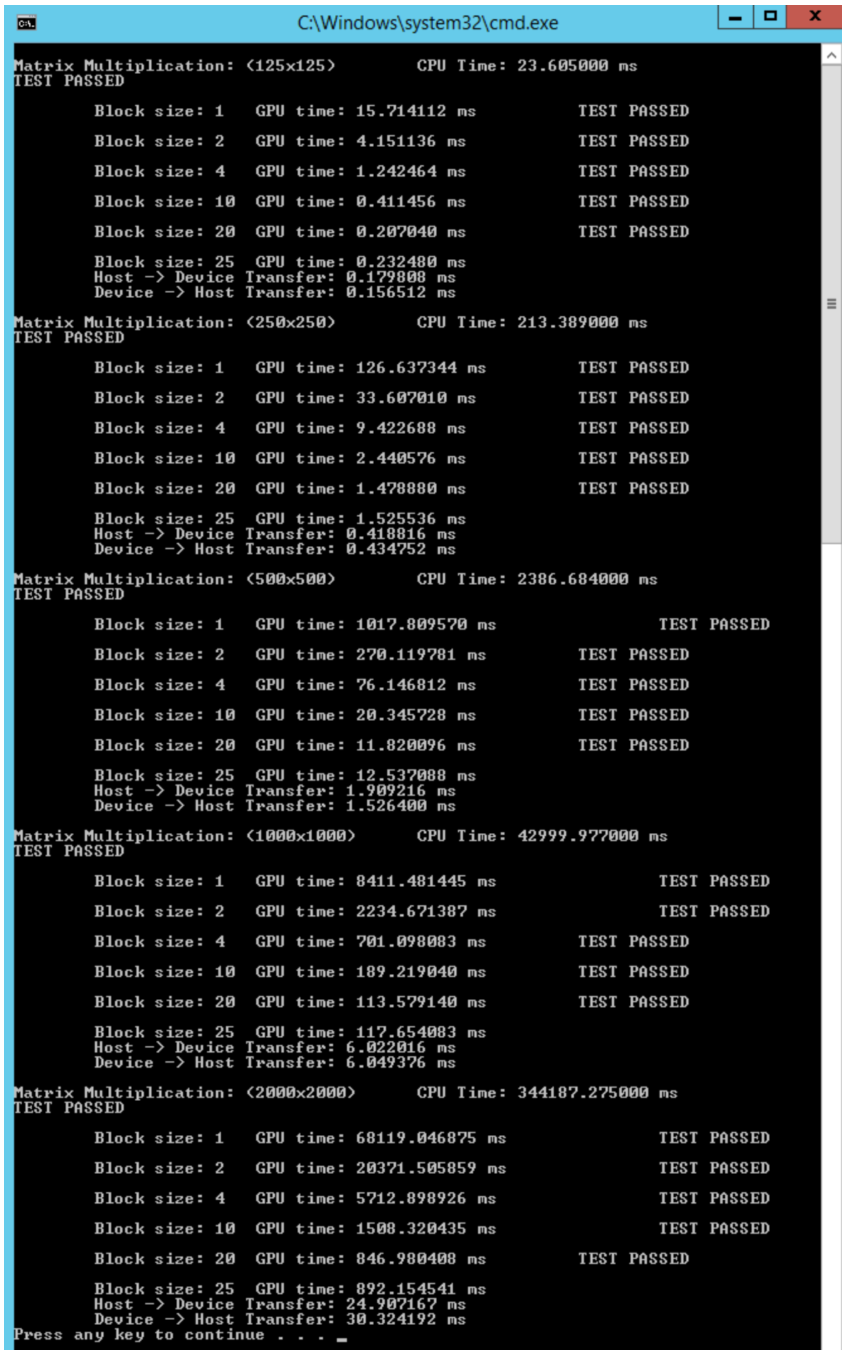
Machine Problem 3

Abdellah Ghassel (20230384)

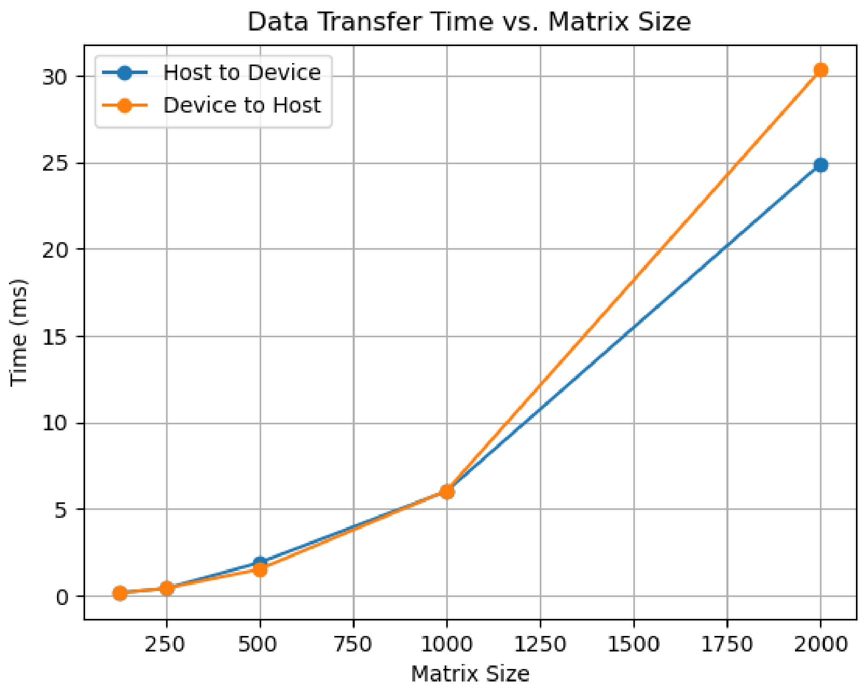
“I do hereby verify that this machine problem submission is my own work and contains my own original ideas, concepts, and designs. No portion of this report or code has been copied in whole or in part from another source, with the possible exception of properly referenced material".

Terminal Output



```
Matrix Multiplication: (125x125)        CPU Time: 23.605000 ms
TEST PASSED

        Block size: 1   GPU time: 15.714112 ms          TEST PASSED

        Block size: 2   GPU time: 4.151136 ms           TEST PASSED

        Block size: 4   GPU time: 1.242464 ms           TEST PASSED

        Block size: 10  GPU time: 0.411456 ms           TEST PASSED

        Block size: 20  GPU time: 0.207040 ms           TEST PASSED

        Block size: 25  GPU time: 0.232480 ms
        Host -> Device Transfer: 0.179808 ms
        Device -> Host Transfer: 0.156512 ms

Matrix Multiplication: (250x250)        CPU Time: 213.389000 ms
TEST PASSED

        Block size: 1   GPU time: 126.637344 ms         TEST PASSED

        Block size: 2   GPU time: 33.607010 ms          TEST PASSED

        Block size: 4   GPU time: 9.422688 ms           TEST PASSED

        Block size: 10  GPU time: 2.440576 ms           TEST PASSED

        Block size: 20  GPU time: 1.478880 ms           TEST PASSED

        Block size: 25  GPU time: 1.525536 ms
        Host -> Device Transfer: 0.418816 ms
        Device -> Host Transfer: 0.434752 ms

Matrix Multiplication: (500x500)        CPU Time: 2386.684000 ms
TEST PASSED

        Block size: 1   GPU time: 1017.809570 ms                TEST PASSED

        Block size: 2   GPU time: 270.119781 ms         TEST PASSED

        Block size: 4   GPU time: 76.146812 ms          TEST PASSED

        Block size: 10  GPU time: 20.345728 ms          TEST PASSED

        Block size: 20  GPU time: 11.820096 ms          TEST PASSED

        Block size: 25  GPU time: 12.537088 ms
        Host -> Device Transfer: 1.909216 ms
        Device -> Host Transfer: 1.526400 ms

Matrix Multiplication: (1000x1000)      CPU Time: 42999.977000 ms
TEST PASSED

        Block size: 1   GPU time: 8411.481445 ms                TEST PASSED

        Block size: 2   GPU time: 2234.671387 ms                TEST PASSED

        Block size: 4   GPU time: 701.098083 ms         TEST PASSED

        Block size: 10  GPU time: 189.219040 ms         TEST PASSED

        Block size: 20  GPU time: 113.579140 ms         TEST PASSED

        Block size: 25  GPU time: 117.654083 ms
        Host -> Device Transfer: 6.022016 ms
        Device -> Host Transfer: 6.049376 ms

Matrix Multiplication: (2000x2000)      CPU Time: 344187.275000 ms
TEST PASSED

        Block size: 1   GPU time: 68119.046875 ms               TEST PASSED

        Block size: 2   GPU time: 20371.505859 ms               TEST PASSED

        Block size: 4   GPU time: 5712.898926 ms                TEST PASSED

        Block size: 10  GPU time: 1508.320435 ms                TEST PASSED

        Block size: 20  GPU time: 846.980408 ms         TEST PASSED

        Block size: 25  GPU time: 892.154541 ms
        Host -> Device Transfer: 24.907167 ms
        Device -> Host Transfer: 30.324192 ms
Press any key to continue . . . _
```

Part 1



Data transfers between devices and hosts aren't significantly different in terms of performance. Transfer times increase with matrix size for both types of transfers. The transfer of data from the device to the host may take longer in some cases due to extra overhead, such as cache synchronization and memory management. Compared to the overall transfer time, this difference would likely be negligible.

Python Code for Part 1:

```
import pandas as pd
import matplotlib.pyplot as plt

transfer_data = [
    (125, 0, 0.179808),
    (125, 1, 0.156512),
    (250, 0, 0.418816),
    (250, 1, 0.434752),
    (500, 0, 1.909216),
    (500, 1, 1.526400),
    (1000, 0, 6.022016),
    (1000, 1, 6.049376),
    (2000, 0, 24.907167),
    (2000, 1, 30.324192),
]

# Create a DataFrame
transfer_df = pd.DataFrame(transfer_data, columns=["matrixSize", "case", "time"])

# Separate host-to-device and device-to-host data
host_to_device_data = transfer_df[transfer_df['case'] == 0]
device_to_host_data = transfer_df[transfer_df['case'] == 1]

# Plot host-to-device data
plt.plot(host_to_device_data['matrixSize'], host_to_device_data['time'], marker='o', label='Host to Device')
```

```
# Plot device-to-host data
plt.plot(device_to_host_data['matrixSize'], device_to_host_data['time'], marker='o', label='Device to Host')

plt.xlabel('Matrix Size')
plt.ylabel('Time (ms)')
plt.title('Data Transfer Time vs. Matrix Size')
plt.legend()
plt.grid()
plt.show()
```

Part 2

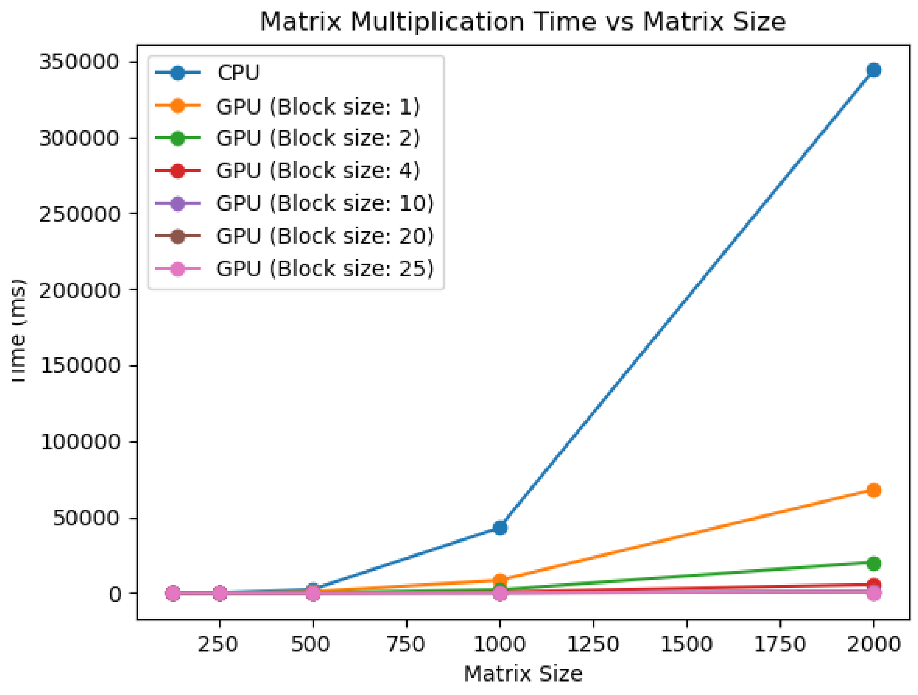
Is it always beneficial to offload your matrix multiplication to the device?

Offloading matrix multiplication to the device is not always beneficial, as it depends on various factors such as matrix size, device computational power, and data transfer times. For small matrices, the overhead of transferring data between the host and the device might outweigh the performance benefits of using the GPU for matrix multiplication. As seen in the terminal output, the GPU outperformed the CPU in all cases. This is further shown in the plot in Part 3.

However, for larger matrices, the computational advantage of the GPU becomes more significant, and it is more likely to be beneficial to offload matrix multiplication to the device. This is because the GPU can perform many more calculations in parallel, resulting in faster matrix multiplication, even when accounting for the data transfer time.

Part 3

It can be seen from the plot below, that a block size of 4 is optimal, and the computational time increases nearly exponentially for the CPU time as the matrix size increases, due to O(N^2) time complexity.



Python Code for Plot 2:

```
import pandas as pd
import matplotlib.pyplot as plt

data_points = [
    (125, "CPU", 0, 23.605000),
    (125, "GPU", 1, 15.714112),
    (125, "GPU", 2, 4.151136),
    (125, "GPU", 4, 1.242464),
    (125, "GPU", 10, 0.411456),
    (125, "GPU", 20, 0.207040),
    (125, "GPU", 25, 0.232480),
    (250, "CPU", 0, 213.389000),
    (250, "GPU", 1, 126.637344),
    (250, "GPU", 2, 33.607010),
    (250, "GPU", 4, 9.422688),
    (250, "GPU", 10, 2.440576),
    (250, "GPU", 20, 1.478880),
    (250, "GPU", 25, 1.525536),
    (500, "CPU", 0, 2386.684000),
    (500, "GPU", 1, 1017.809570),
    (500, "GPU", 2, 270.119781),
    (500, "GPU", 4, 76.146812),
    (500, "GPU", 10, 20.345728),
    (500, "GPU", 20, 11.820096),
    (500, "GPU", 25, 12.537088),
    (1000, "CPU", 0, 42999.977000),
    (1000, "GPU", 1, 8411.481445),
    (1000, "GPU", 2, 2234.671387),
    (1000, "GPU", 4, 701.098083),
    (1000, "GPU", 10, 189.219040),
    (1000, "GPU", 20, 113.579140),
    (1000, "GPU", 25, 117.654083),
    (2000, "CPU", 0, 344187.275000),
    (2000, "GPU", 1, 68119.046875),
    (2000, "GPU", 2, 20371.505859),
    (2000, "GPU", 4, 5712.898926),
    (2000, "GPU", 10, 1508.320435),
    (2000, "GPU", 20, 846.980408),
    (2000, "GPU", 25, 892.154541)
]

# Create a DataFrame
df = pd.DataFrame(data_points, columns=["matrixSize", "processor", "blockSize", "time"])

# Separate CPU and GPU data
cpu_data = df[df['processor'] == 'CPU']
gpu_data = df[df['processor'] == 'GPU']

# Plot CPU data
plt.plot(cpu_data['matrixSize'], cpu_data['time'], marker='o', label='CPU')

# Get unique block sizes
block_sizes = gpu_data['blockSize'].unique()

# Get unique block sizes
block_sizes = gpu_data['blockSize'].unique()

# Plot GPU data for each block size
for block_size in block_sizes:
    gpu_block_data = gpu_data[gpu_data['blockSize'] == block_size]
    plt.plot(gpu_block_data['matrixSize'], gpu_block_data['time'], marker='o', label=f'GPU (Block size: {block_size})')

# Customize plot appearance
plt.xlabel('Matrix Size')
plt.ylabel('Time (ms)')
plt.title('Matrix Multiplication Time vs Matrix Size')
plt.legend()
plt.show()
```

(a) How many times is each element of each input matrix loaded during the execution of the kernel?

Note: from the slides.

There are two global memory accesses performed: one for floating-point multiplication and the other the floating point addition. Thus, one global memory access fetches M[] element and the other fetches N[] element for 8 bytes total.

```
_global__ void matrixMultKernel(float* A, float* B, float* C, int N) {

  int row = blockIdx.y * blockDim.y + threadIdx.y;
    int col = blockIdx.x * blockDim.x + threadIdx.x;
  int idx = row * N + col;

  if (row < N && col < N) {
    C[idx] = 0.0;
    for (int i = 0; i < N; i++)
      C[idx] += A[row * N + i] * B[i * N + col];
  }
```

b) What is the floating-point computation to memory-access (CGMA) ratio in each thread? Consider multiply and addition as separate operations and ignore the global memory store at the end. Only count global memory loads towards your off-chip bandwidth.

Elaborating from above, the first floating-point operation multiplies M[] and N[] elements, and the other accumulates the product into Pvalue.

The ratio of floating-point operations (FLOP) to bytes accessed from global memory:

$$\frac{2FLOP}{8B} = 0.25\frac{FLOP}{B}$$