ELEC 374

Machine Problem 3

Luka Gobovic

20215231

# Output

## Data transfer times using malloc:

Graphical user interface, text

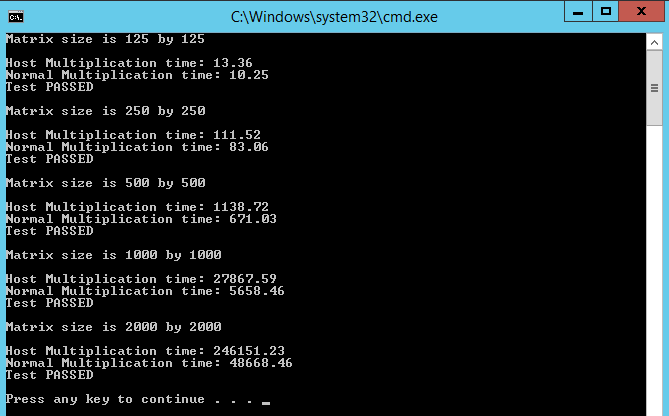
Description automatically generated

## Using cudaMallocHost:

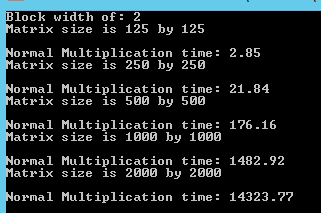
Graphical user interface, text

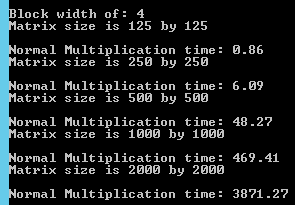
Description automatically generated

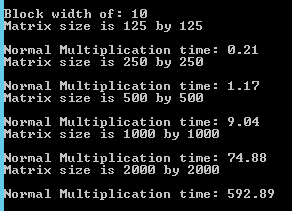
## For a single block and 1 thread per block

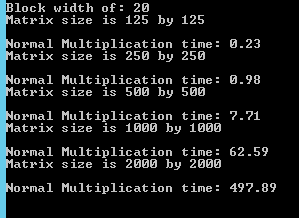


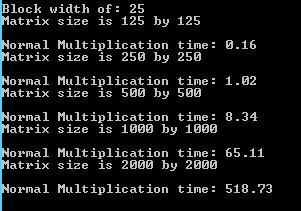
No checks against the CPU were added due to the very long run time of the 2000x2000 matrix. It would take too long.











# Code Part 1

Text

Description automatically generatedText

Description automatically generated

# Code Part 2

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generatedText

Description automatically generated

# Code Part 3

Text

Description automatically generatedText

Description automatically generated

Text

Description automatically generatedText

Description automatically generated

# Analysis

## Part 1:

When using both methods, the transfer times seem to be relatively similar for to the device and back from the device. However, when using cudaHostMalloc to allocate memory on the host, the transfer time form the host to the device is around 1/4th of the transfer time back from the device to the host. Generally, as the matrix sizes increase, it can be seen that using cudaHostMalloc results in overall lower transfer times. cudaMallocHost() is a CUDA function that allocates pinned memory. Pinned memory is faster for data transfer between the host and the device because the operating system can avoid copying the data to the pagefile on disk. The pinned memory is stored in page-locked physical memory, which can be directly accessed by the GPU.

## Part 2:

When considering data transport time, offloading matrix multiplication to the device (GPU) is not always advantageous. The size of the matrices and the amount of available bandwidth between the host and the device determine whether to offload the computation to the GPU.

The computation time may be dominated by the data transfer time for smaller matrix sizes, making offloading to the device slower. The processing time on the GPU, on the other hand, might be substantially faster than the CPU for higher matrix sizes, balancing the data transmission time and producing a net speedup.

## Part 3:

a) Since each element of the input matrix is loaded once into the GPU memory for each multiplication, the total number of loads per element is equal to the total number of multiplications performed. The total number of multiplications needed to compute the output is MATRIX\_WIDTH^3. So if we have a width of 250, the total number of loads would be 250\*250\*250 = 15,625,000

b) For each matrix multiplication, each element requires one multiplication and addition. Therefore the number of floating-point operations is equal to 2 multiplied by the number of elements in the output matrix. The number of memory access is equal to the number of elements in both input matrices.

CGMA can be calculated as follows:

CGMA = (number of floating-point operations)/(number of memory accesses)

Number of floating-point operations = 2 \* (matrix size)^3

Number of bytes accessed = 2 \* (matrix size)^2 \* sizeof(float)

Therefore, the CGMA in FLOPs/B is:

CGMA = (2 \* (matrix size)^3) / (2 \* (matrix size)^2 \* sizeof(float))

= (matrix size) / sizeof(float)

For a 250x250 matrix of floats (each float taking 4 bytes), the CGMA is:

CGMA = 250 / 4 = 62.5 FLOPs/B

Generally, for matrix multiplication, the CGMA is 0.25 FLOP/B.