1. How many floating operations are being performed in your matrix multiply kernel? Explain.
   There are (2*m*n*k-m*k) floating operations in the matrix multiplication kernel. There are a total of m*n*k multiplication operations as the output matrix is of size m*k and there are n number of multiplications per output element. There are a total of m*k(n-1) number of addition operations as the output matrix is of size m*k and there is n-1 number of addition operations per output.

2. How many global memory reads are being performed by your kernel? Explain.
   There are (2*n*m*k) number of global memory reads being performed by the kernel. The only time the kernel is reading from global memory is during the multiplication function. There are m*k number of elements in the output vector, and each element has n number of addition operations with each operation containing a multiplication operation. This equates to a total of 2*n memory reads per element in the output matrix.

3. How many global memory writes are being performed by your kernel? Explain.
   There are (m*n) + (n*k) + (m*k) global memory writes being performed by the kernel. This equates to the number of elements in all three matrixes as all the elements are modified once. In the multiplication kernel it only writes to global memory when the output matrix element has been computed.

4. Describe what possible optimizations can be implemented to your kernel to achieve a performance speed-up.
   Storing matrix A and B in shared memory can increase performance as shared memory is magnitudes faster than global memory. Specifically only storing the row of matrix A and the column of matrix B being used to compute the value of the output matrix element since each thread equates to an element in the output matrix.

5. Name three applications of matrix multiplication.
   Blur filter on images, 3-D animation, Physics engine

CPU & GPU Implementation Test

Testing with input of 5000, 4000, 3000

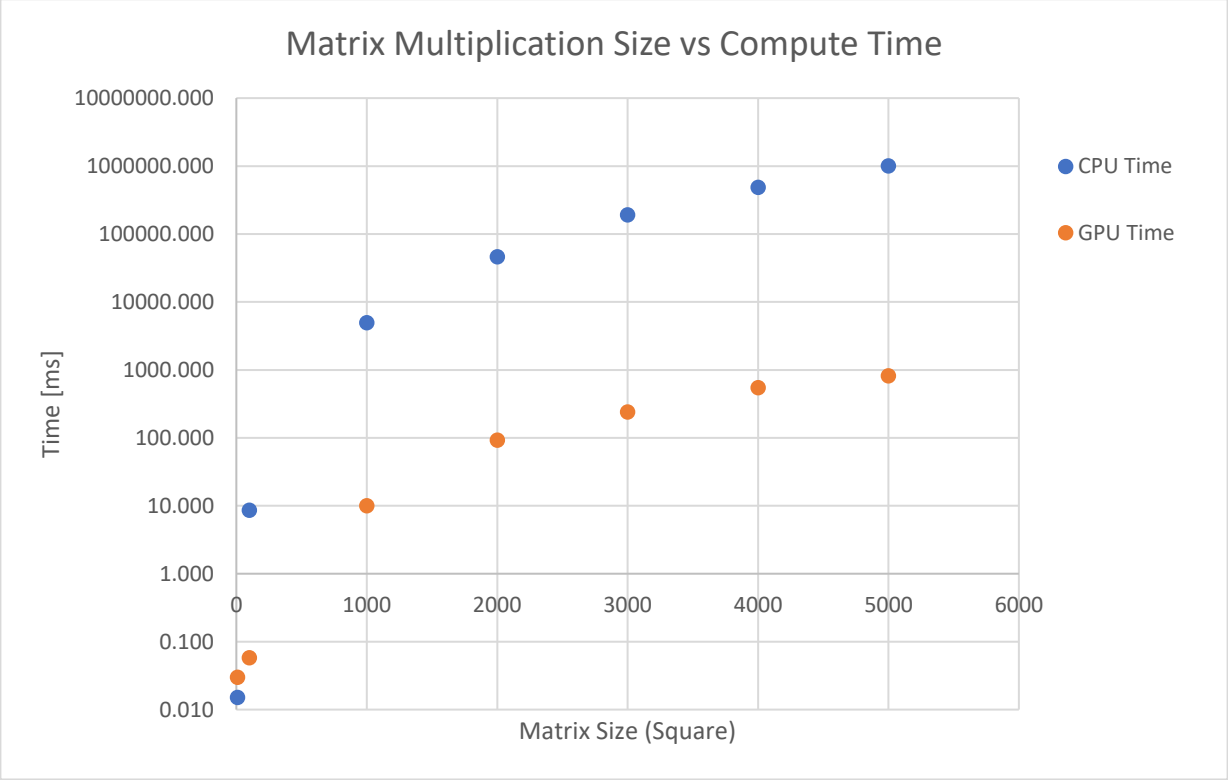
```
singkhamj@dh-node2:~/cuda/build/lab3$ ./MatrixMultiplication 5000 4000 3000

******************CPU Implementation*****************
Calculated Output = 4000.000000
True Output = 4000
Output Difference = 0.000000

CPU ellapsed time: 453007.843750 milliseconds

****************Clearing Output Memory***************
Sum should be 0. Sum in memory = 0.000000

**************Initializing GPU Variables**************

******************GPU Implementation*****************
Matrix C

Calculated Output = 4000.000000
True Output = 4000
Output Difference = 0.000000

CPU ellapsed time: 536.584229 milliseconds
```

Julian Singkham

Regular CPU and GPU Implementation:

Matrices are created prior to matrix multiplication in the CPU and GPU

Regular GPU and CPU Implementation			
Matrix Size	CPU Time [ms]	GPU Time [ms]	GPU Speed Gain
10	1.513E-02	2.986E-02	0.506899786
100	8.612E+00	5.798E-02	148.518919
1000	4.955E+03	1.000E+01	495.5417969
2000	4.613E+04	9.242E+01	499.0685949
3000	1.909E+05	2.404E+02	794.1989206
4000	4.867E+05	5.500E+02	884.8684809
5000	1.010E+06	8.164E+02	1236.69889



Julian Singkham

Regular CPU and GPU with multiple kernels:

Times include memory allocations to the matricies, inialization of the matrices and the multiplication function. The CPU and GPU create matrices independently.

Multiple Kernels Implementation			
Matrix Size	CPU Time [ms]	GPU Time [ms]	GPU Speed Gain
5000	1.097E+06	2.552E+03	429.9898087
6000	1.773E+06	4.125E+03	429.7690072
7000	3.563E+06	6.683E+03	533.0500546