Homework No. 5: Matrix Multiplication on a GPU Using CUDA

By Shahzad Khan

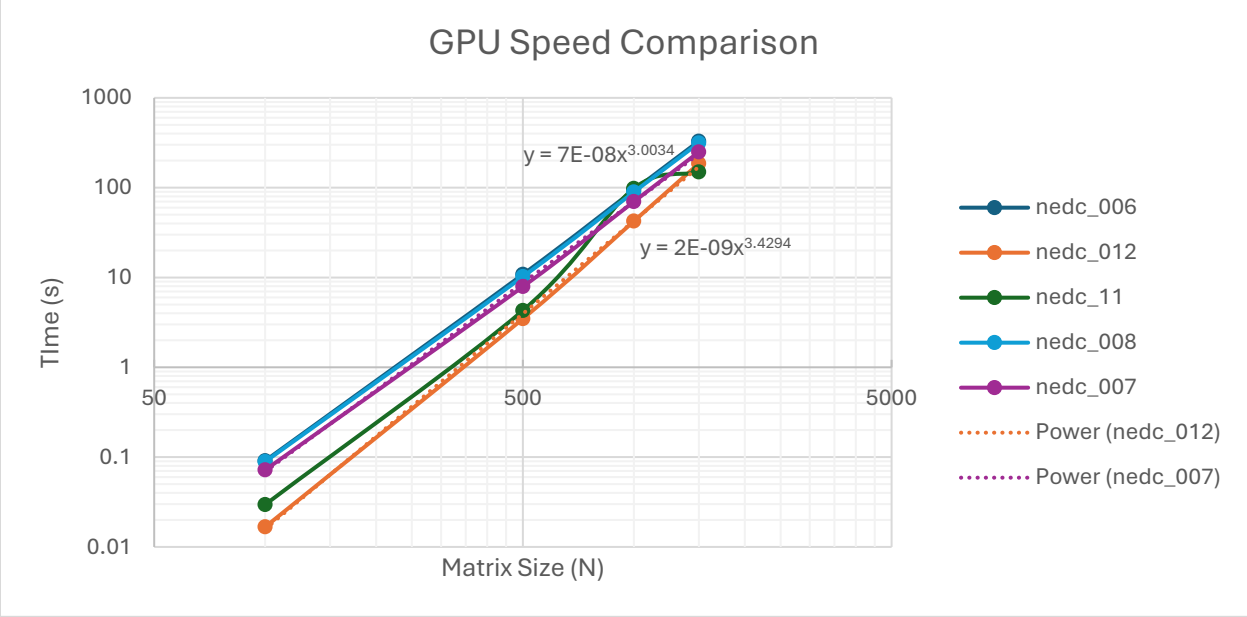
The goal for this assignment was to implement our indexing version of my matrix multiplication code to be run on a gpu. As a reminder our original indexing code was the following

```
bool multiMatrix(float* mat3, float* mat2, float* mat1, long nrows, long ncols){

 if(!mat1 || !mat2 || !mat3){
  return false;
 }

 for (long i = 0; i < nrows; i++) {
  for (long j = 0; j < nrows; j++) {
   float sum = 0;
   for (long k = 0; k < ncols; k++) {
    sum += mat1[i * ncols + k] * mat2[k * nrows + j];
   }
   mat3[i * nrows + j] = sum;
  }
 }
 return true;
}
```

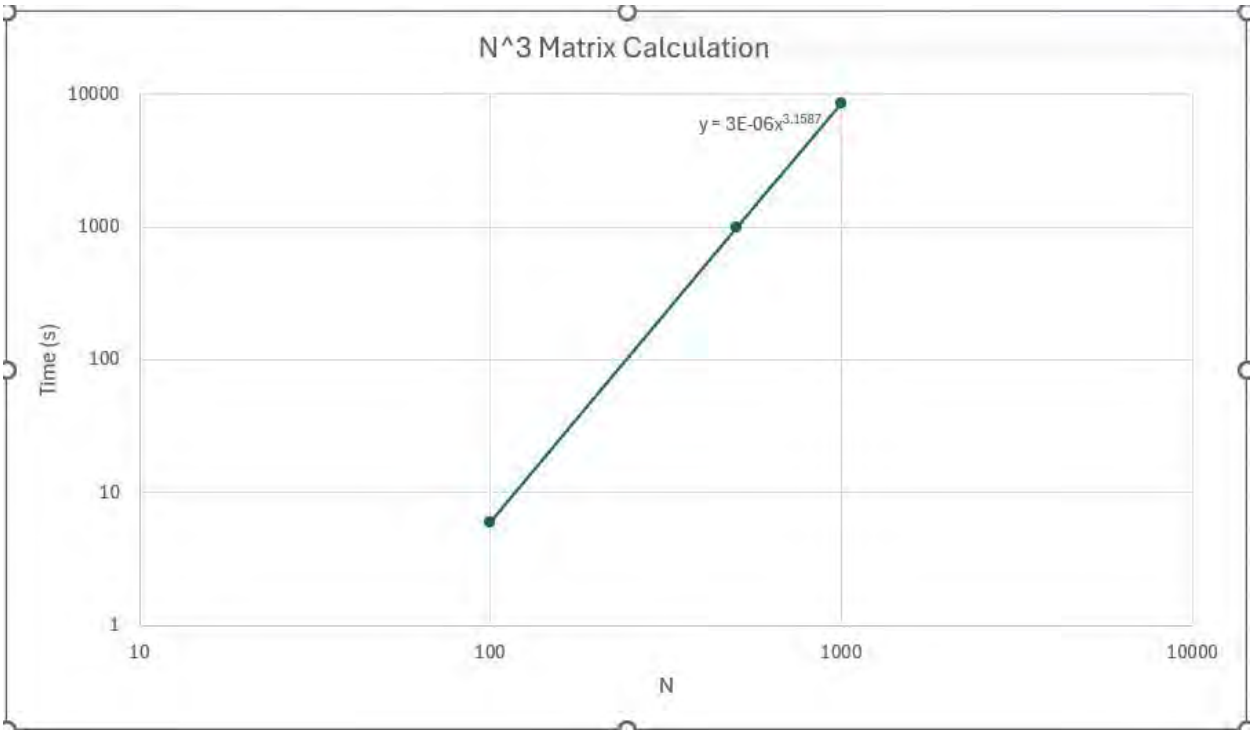
This gave us the time complexity of $O(n^3)$ which when running this program with larger matrices caused our runtime to grow exponentially.

Now converting this code to run on a gpu was simple, since our goal was to simply over implement this only using 1 thread and 1 block <<<1, 1>>> to get a baseline for our next homework.

Our results are shown here



As we can see, it seems the fastest gpu was nedc_012, but something interesting is that nedc_11 did the best with my highest N, 1500. Even outclassing nedc_012.



Looking back at my original hw_01 graph we can see that the time complexity around the same at $O(N^3)$ which makes sense because we aren't utilizing any parallelization

In conclusion, this assignment provided a good baseline for future GPU coding assignments with the different speeds of the GPUs. Nedc_012 being overall the fastest and nedc_011 having dip with a 1000x1000 matrix, and then being the fastest with a 1500x1500 matrix.