```
/*
 * Purpose: Time square matrix multiplication usingh CPU, GPU, cblas ddot,
            cblas daxpy, cublas ddot, cublas daxpy.

 * Author: Gurpal Singh
 * Date: April 6, 2017
 * ME 571 Project 2
 */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <sys/resource.h>
#include <time.h>
#include "timer.h"
#include "gsl_cblas.h"
#include <cublas.h>

#include "cpumatrixmultiply.h"
#include "cpuddot.h"
#include "cpudaxpy.h"
#include "gpumatrixmultiply.h"
#include "gpuddot.h"
#include "gpudaxpy.h"


int main(void){

        //Initializing matrix Dimensions and scanning for values from user
        int m, n, k;

        printf("This program performs square  matrix multiplication where A is ’m x n’ and B is ’n x k’.\n");

        //Setting Matrix Dimensions
        printf("Enter the integer value for n: ");
        scanf("%d", &n);
        m = n;
        k = n;


        //Dynamic Memory Allocation for Matrices (I used flat arrays)
        //Matrix A
        double *a;
        cudaMallocManaged( &a, m * n * sizeof(double));

        //Matrix B
        double *b;
        cudaMallocManaged( &b, n * k * sizeof(double));

        //Matrix C
        double *c;
        cudaMallocManaged( &c, m * k * sizeof(double));

        //Initializing nmax and i and j used for loops
        int nmax = 19;
        int i, j;

        time_t t;
        srand( (unsigned) t );

        //Setting random values in Matrix A
        for (i = 0; i < m; i++){
                for (j = 0; j < n; j++){
                        a[i*n+j] = rand() % (nmax+1);
                }
        }

        //Setting random values in Matrix B
```

```
        for (i = 0; i < n; i++){
                for (j = 0; j < k; j++){
                        b[i*k+j] = rand() % (nmax+1);
                }
        }

        //Transposing and Storing as Column Major arrays
        double *acol;
        cudaMallocManaged( &acol,m *  n * sizeof(double));
        for(i = 0; i < n; i++){
                for(j = 0; j < m; j++){
                        acol[i*m+j] = a[j*n+i];
                }
        }

        double *bcol;
        cudaMallocManaged( &bcol,k * n * sizeof(double));
        for(int i = 0; i < k; i++){
                for( j = 0; j < n; j++){
                        bcol[i*n+j] = b[j*k+i];
                }
        }


        //Timing the CPU Matrix Multiplication Method
        StartTimer();
        CPU_Matrix_Multiply(m, n, k, a, b, c);
        double CPU_time = GetTimer();
        CPU_time = CPU_time*1000; //Converting to ms


        //Timing the CPU Matrix Multiplication using cblas ddot method
        StartTimer();
        CPU_ddot(m, n, k, a, bcol, c);
        double CPU_ddot_time = GetTimer();
        CPU_ddot_time = CPU_ddot_time*1000; //Converting to ms


        //Timing the CPU Matrix Multiplication using cblas daxpy
        StartTimer();
        CPU_daxpy(m, n, k, acol, bcol, c);
        double CPU_daxpy_time = GetTimer();
        CPU_daxpy_time = CPU_daxpy_time*1000;



        //Parallel GPU Code Block and Grid Dimensions
        dim3 block(16,16);
        dim3 grid( (n+15)/16, (n+15)/16 );

        //Timing the GPU Matrix Multiplication Kernel
        cudaEvent_t timeStart, timeStop;

        cudaEventCreate(&timeStart);
        cudaEventCreate(&timeStop);
        float elapsedTime; //Has to be type float units ms

        cudaEventRecord(timeStart, 0);
        GPU_Matrix_Multiply_Kernel<<<grid, block>>> (a, b, c, n);
        cudaDeviceSynchronize();

        cudaEventRecord(timeStop,0);
        cudaEventSynchronize(timeStop);
        cudaEventElapsedTime(&elapsedTime, timeStart, timeStop);

        cudaEventDestroy(timeStart);
        cudaEventDestroy(timeStop);

        //Timing the GPU cublas DDOT Matrix Multiplication Method
        StartTimer();
        GPU_ddot(m, n, k, a, bcol, c);
        double GPU_ddot_time = GetTimer();
        GPU_ddot_time = GPU_ddot_time*1000;


        //Timing the GPU cublas DAXPY Matrix Multiplication Method
        StartTimer();
        GPU_daxpy(m, n, k, acol, bcol, c);
        double GPU_daxpy_time = GetTimer();
        GPU_daxpy_time = GPU_daxpy_time*1000;


        //Writing The results to a file
        FILE *fptr = fopen("Matrix_Multiply_Results.txt", "a+");

        if (fptr == NULL) {
                printf("Error!");
                exit(1);
        }

        fprintf(fptr, "\n");
        fprintf(fptr, "Matrix Size: %d\n", n);
        fprintf(fptr, "elapsed wall time CPU Matrix Multiplication = %.3f ms\n", CPU_time);
        fprintf(fptr, "elapsed wall time CPU cblas DDOT %.3f ms\n", CPU_ddot_time);

        fprintf(fptr, "elapsed wall time CPU cblas DAXPY %.3f ms\n",CPU_daxpy_time);

        fprintf(fptr, "elapsed wall time GPU Matrix Multiplication %.3f ms\n", elapsedTime);


        fprintf(fptr, "elapsed wall time GPU cublas DDOT %.3f ms\n", GPU_ddot_time);

        fprintf(fptr, "elapsed wall time GPU cublas DAXPY %.3f ms\n", GPU_daxpy_time);
        fclose(fptr);

        //Clean Up
        cudaFree(a);
        cudaFree(b);
        cudaFree(c);
        cudaFree(acol);
        cudaFree(bcol);
}
```