```c
/*
 * Purpose: Demonstrate matrix multiplication in
 * CPU and GPU with global memory and shared memory usage
 * Date and time: 04/09/2014
 *
 * Last modified: Dustin (Ting-Hsuan) Ma
 * Date : November 20, 2018
 * Author: Inanc Senocak
 *
 * to compile blas: nvcc -lcublas -O2 gpu_matrixMultiply.cu -o GPU.exe
 * to execute: ./matrixMult.exe <m> <n> <k>
 */

#include "cublas_v2.h"
#include "timer.h"
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/resource.h>
#include <time.h>

#define BLOCKSIZE 16

typedef double REAL;
typedef int    INT;

void printMatrix(REAL *matrix, const int nrow, const int ncol)
{
        int i, j, idx;

        for (j = 0; j < nrow; j++) {
                for (i = 0; i < ncol; i++) {
                        idx = i + j * ncol;
                        printf("%8.2f ; ", matrix[idx]);
                }
                printf("\n");
        }
        printf("\n");
}

void InitializeMatrices(REAL *a, REAL *b, const int M, const int N, const int K)
{
        int i, j, idx;

        // initialize matrices a & b
        for (j = 0; j < M; j++) {
                for (i = 0; i < N; i++) {
                        idx    = i + j * N;
                        a[idx] = (REAL) idx;
                }
        }

        for (j = 0; j < N; j++) {
                for (i = 0; i < K; i++) {
                        idx    = i + j * K;
                        b[idx] = (REAL) idx;
                }
        }
}

__global__ void matrixMultiplyGPU_gl(REAL *a, REAL *b, REAL *c, const int M, const int N,
                                     const int K)
{
        // Block index

        int bx = blockIdx.x;
        int by = blockIdx.y;

        // Thread index

        int tx = threadIdx.x;
        int ty = threadIdx.y;

        // Row index of matrices a and c

        int row = by * BLOCKSIZE + ty;

        // Column index of matrices a and b
        int col = bx * BLOCKSIZE + tx;

        REAL C_temp = 0.;

        for (int k = 0; k < N; k++)
                C_temp += a[k + row * N] * b[col + k * K];

        c[col + row * K] = C_temp;
```



```c
}

int main(INT argc, char *argv[])
{
        if (argc < 3) {
                perror("Command-line usage: executableName <M> <N> <K>");
                exit(1);
        }

        int M = atof(argv[1]);
        int N = atof(argv[2]);
        int K = atof(argv[3]);

        REAL *a_d, *b_d, *c_d, *d_d, *e_d;

        cudaMallocManaged(&a_d, M * N * sizeof(*a_d));
        cudaMallocManaged(&b_d, N * K * sizeof(*b_d));
        cudaMallocManaged(&c_d, M * K * sizeof(*c_d)); // Used for GPU
        cudaMallocManaged(&d_d, M * K * sizeof(*d_d)); // Used for cublasDDOT
        cudaMallocManaged(&e_d, M * K * sizeof(*e_d)); // Used for cublasDAXPY

        InitializeMatrices(a_d, b_d, M, N, K);

        // Setting up GPU enviorment
        dim3 dimBlock(BLOCKSIZE, BLOCKSIZE);
        dim3 dimGrid((K + BLOCKSIZE - 1) / BLOCKSIZE, (M + BLOCKSIZE - 1) / BLOCKSIZE);

        float elapsedTime_gpu, elapsedTime_DDOT, elapsedTime_DAXPY;

        printf("=====MultKernel=====\n");
        cudaEvent_t timeStart, timeStop; // WARNING!!! use events only to time the device
        cudaEventCreate(&timeStart);
        cudaEventCreate(&timeStop);
        cudaEventRecord(timeStart, 0);

        matrixMultiplyGPU_gl<<<dimGrid, dimBlock>>>(a_d, b_d, c_d, M, N, K);

        cudaDeviceSynchronize();
        cudaEventRecord(timeStop, 0);
        cudaEventSynchronize(timeStop);
        cudaEventElapsedTime(&elapsedTime_gpu, timeStart, timeStop);

        //    printMatrix( c_d, M, K );
        printf("C[2] = %3.1f\n", c_d[2]);
        printf("elapsed wall time (GPU) = %5.2f ms\n", elapsedTime_gpu);

        printf("=====cublasDDOT=====\n");
        cublasHandle_t handle;
        cublasCreate(&handle);

        cudaEventRecord(timeStart, 0);

        for (int i = 0; i < M; i++) {
                for (int j = 0; j < K; j++) {
                        cublasDdot(handle, N, a_d + j * N, 1, b_d + i, K, d_d + i + j * K);
                }
        }

        cudaEventRecord(timeStop, 0);
        cudaEventSynchronize(timeStop);
        cudaEventElapsedTime(&elapsedTime_DDOT, timeStart, timeStop);
        //    printMatrix( d_d, M, K );
        printf("D[2] = %3.1f\n", d_d[2]);
        printf("elapsed wall time (cublasDDOT) = %5.2f ms\n", elapsedTime_DDOT);

        printf("=====cublasDAXPY=====\n");
        cudaEventRecord(timeStart, 0);

        for (int j = 0; j < M; j++) {
                for (int i = 0; i < K; i++) {
                        cublasDaxpy(handle, M, b_d + j + i * K, a_d + i, N, e_d + j, K);
                }
        }

        cudaEventRecord(timeStop, 0);
        cudaEventSynchronize(timeStop);
        cudaEventElapsedTime(&elapsedTime_DAXPY, timeStart, timeStop);
        //    printMatrix( e_d, M, K );
        printf("E[2] = %3.1f\n", e_d[2]);
        printf("elapsed wall time (cublasDAXPY) = %5.2f ms\n", elapsedTime_DAXPY);
        printf("\n");
        cublasDestroy(handle);

        // Deallocating Memory
        cudaFree(a_d);
        cudaFree(b_d);
```

```
        cudaFree(c_d);
        cudaFree(d_d);
        cudaFree(e_d);
        cudaEventDestroy(timeStart);
        cudaEventDestroy(timeStop);

        return (EXIT_SUCCESS);
}
```