Midterm Project Report
Student: Lina Mi
CSC8014 Topics: GPU Programming

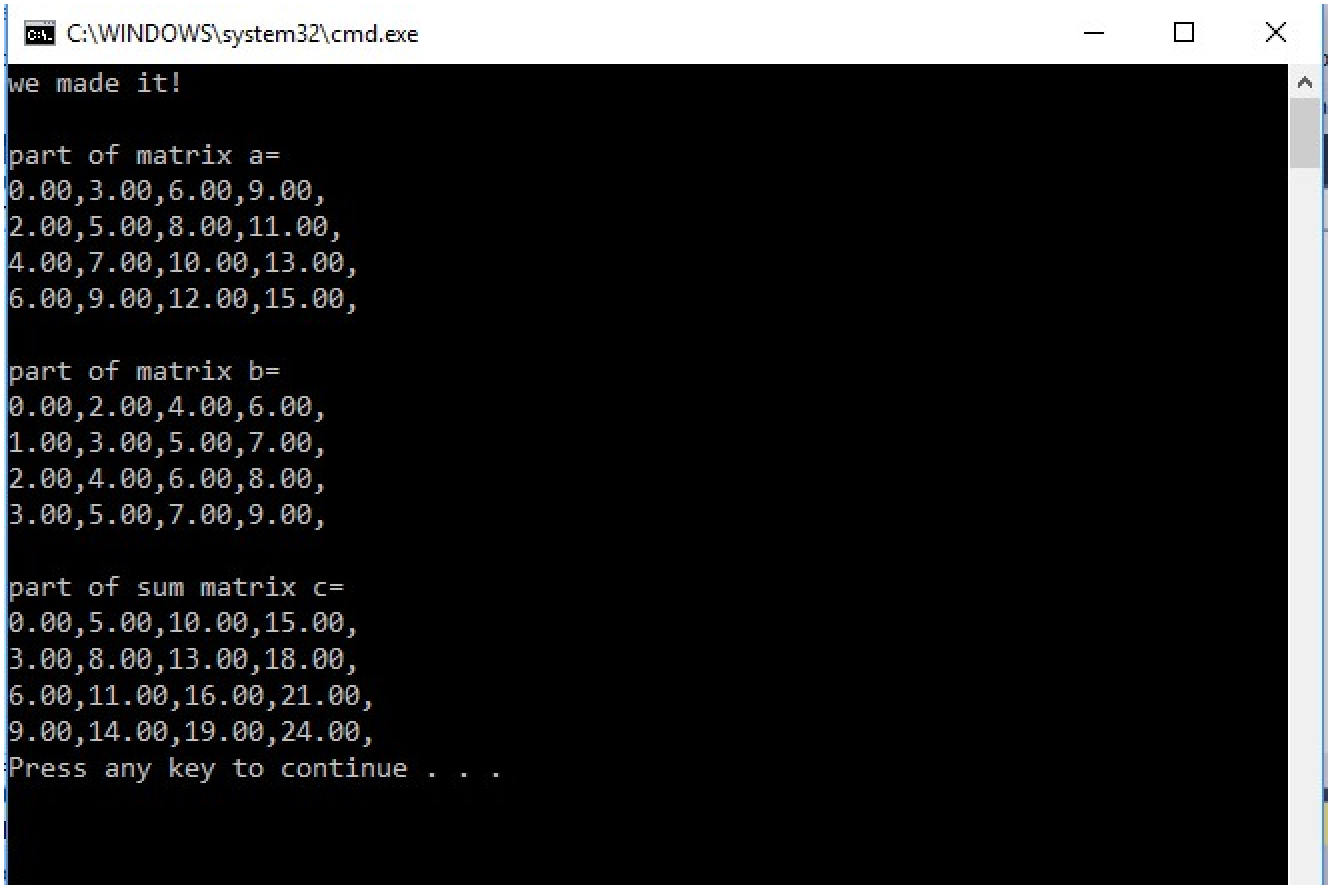
Midterm Project: Matrix Operations

1. Matrix addition (Assume M = N)

The code is listed as following:

```
#include <C:/Program Files (x86)/Microsoft Visual Studio 14.0/VC/include/common/book.h>
#include <cuda.h>
#include<cuda_runtime.h>
#include<stdio.h>



#define M  128 /*the row size of matrix */
#define N  128 /*the column size of matrix */

/*define the kernal function that implement the element addition of two square matrix */
__global__ void matrixAdd(float *a, float *b, float *c, int T) {
	/*use two dimension storage, transfer the thread index and block index of each elements of
matrix into offset index of each element*/
	int x, y;
	x = threadIdx.x + blockIdx.x*blockDim.x;
	y = threadIdx.y + blockIdx.y*blockDim.y;
	int Index = y*blockDim.x*gridDim.x + x; /*calculate the offset index of elements of flatened
matrixes*/
	while(Index < T)
	{
		c[Index] = a[Index] + b[Index];
		Index += blockDim.x*blockDim.y*gridDim.x*gridDim.y;
	}
}

int main(void) {
	float a[M][N], b[M][N], c[M][N];
	float *dev_a, *dev_b, *dev_c;
	dim3 blocks(M*N/32, M*N/32);
	dim3 threads(32, 32);

	/*allocate the memory on the GPU*/
	HANDLE_ERROR(cudaMalloc((void**) &dev_a, M*N*sizeof(float)));
	HANDLE_ERROR(cudaMalloc((void**) &dev_b, M*N*sizeof(float)));
	HANDLE_ERROR(cudaMalloc((void**) &dev_c, M*N*sizeof(float)));

	/*initiate the two square matrixes that will be added*/
	for (int i = 0; i < M; i++) {
		for (int j = 0; j < N; j++) {
			a[i][j] = 2.0*i + 3.0*j;
			b[i][j] = i + 2.0*j;
		}
//		printf("\n");
	}


	//copy initialized matrixes a and b to GPU
	HANDLE_ERROR(cudaMemcpy(dev_a, a, M*N*sizeof(float), cudaMemcpyHostToDevice));
	HANDLE_ERROR(cudaMemcpy(dev_b, b, M*N*sizeof(float), cudaMemcpyHostToDevice));

	//launch the addition function on GPU
matrixAdd << <blocks, threads >> > (dev_a, dev_b, dev_c, M*N);

/*make sure all threads finised calculation before copy the result back to CPU*/
cudaThreadSynchronize();

//copy the sum of two matrixes back to CPU
HANDLE_ERROR(cudaMemcpy(c, dev_c, M*N*sizeof(float), cudaMemcpyDeviceToHost));

/*check the result of addition of two matrixes*/
bool success = true;
for (int i = 0; i < N; i++) {
       for (int j = 0; j < N; j++) {
              if ((a[i][j] + b[i][j]) != c[i][j]) {
          printf("error:%d+%d!=%d\n", a[i][j], b[i][j], c[i][j]);
                      success = false;
              }

       }
}
if (success) printf("we made it!\n");

/*print part of the matrix a and b*/
printf("\npart of matrix a=\n");
for (int i = 0; i < 4; i++) {
       for (int j = 0; j < 4; j++) {
              printf("%0.2f,", a[i][j]);
       }
       printf("\n");
}

printf("\npart of matrix b=\n");
for (int i = 0; i < 4; i++) {
       for (int j = 0; j < 4; j++) {
              printf("%0.2f,", b[i][j]);
       }
       printf("\n");
}

/*print out part of sum matrix c */
printf("\npart of sum matrix c=\n");
for (int i = 0; i < 4; i++) {
       for (int j = 0; j < 4; j++) {
              printf("%0.2f,", c[i][j]);
       }
       printf("\n");
}

/*free allocated GPU memory*/
cudaFree(dev_a);
cudaFree(dev_b);
cudaFree(dev_c);

return (0);

}
```

The output of the codes:



```
we made it!

part of matrix a=
0.00,3.00,6.00,9.00,
2.00,5.00,8.00,11.00,
4.00,7.00,10.00,13.00,
6.00,9.00,12.00,15.00,

part of matrix b=
0.00,2.00,4.00,6.00,
1.00,3.00,5.00,7.00,
2.00,4.00,6.00,8.00,
3.00,5.00,7.00,9.00,

part of sum matrix c=
0.00,5.00,10.00,15.00,
3.00,8.00,13.00,18.00,
6.00,11.00,16.00,21.00,
9.00,14.00,19.00,24.00,
Press any key to continue . . .
```

2. Matrix multiplication

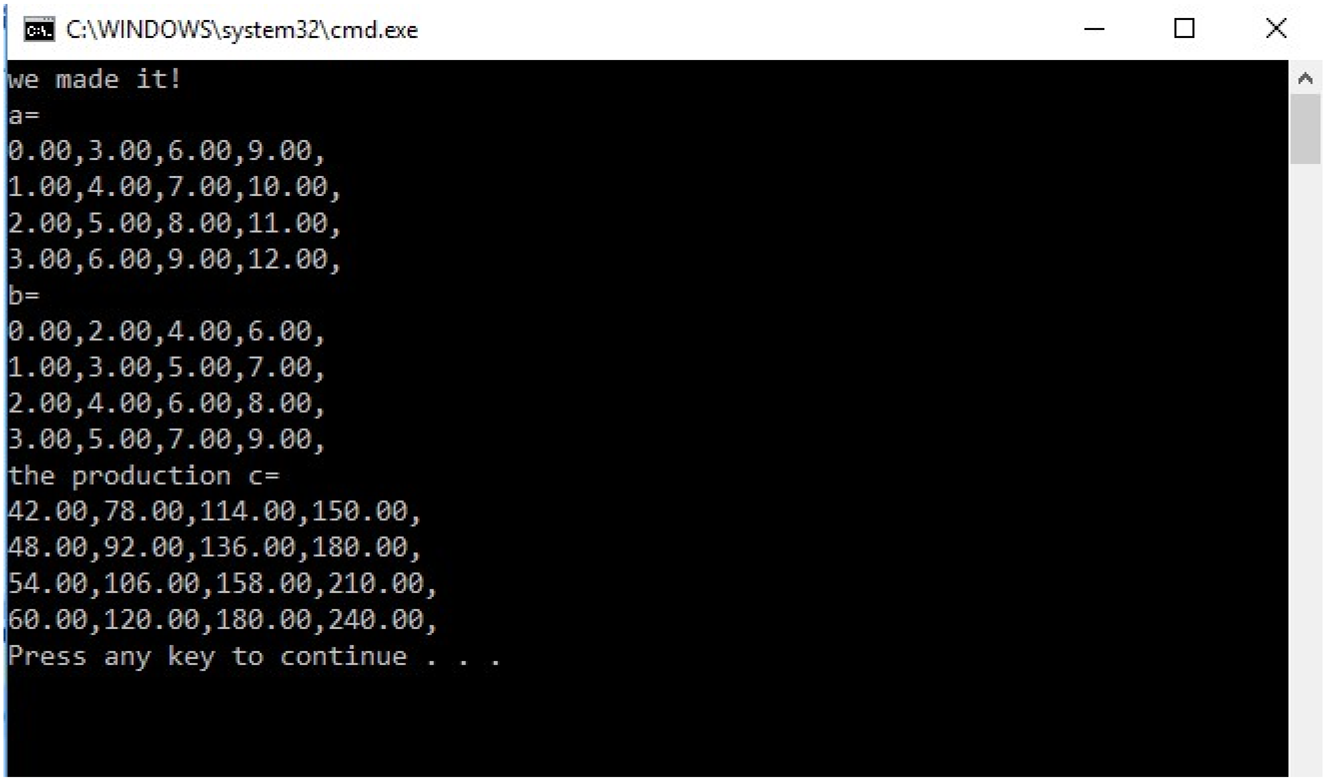
The codes is listed as following: (in order to demonstrate result, the size of square matrix is set to be 4×4, this code can implement square matrix multiplication with the maximum size of 128×128, at this case, blocks will be (N/16,N/16) and threads will be(16,16)

```
#include "C:/Program Files (x86)/Microsoft Visual Studio 14.0/VC/include/common/book.h"
#include<cuda.h>
#include<cuda_runtime.h>
#include<stdio.h>
#include<stdlib.h>

#define N 4 /*define the size of matrix, the maximum size is 128*/



__global__ void matrixMult(float *a, float *b, float *c, int M) {
       int col = threadIdx.x + blockIdx.x*blockDim.x;
       int row = threadIdx.y + blockIdx.y*blockDim.y;
       float sum=0;
       if(col<M && row<M){
              for(int k=0; k<M; k++)
                     sum+=a[row*M+k]*b[k*M+col];
              c[row*M + col] = sum;
       }

}


int main(){
       float a[N][N], b[N][N],c[N][N];
       float *dev_a, *dev_b, *dev_c;
```

```
    dim3 blocks(2,2); /*in the case of matrix size of 128, here will be blocks(N/16,N/16)*/
    dim3 threads(2,2); /*in the case of matrix size of 128, here will be threads(16,16)*/


    HANDLE_ERROR(cudaMalloc((void **)&dev_a, N*N*sizeof(float)));
    HANDLE_ERROR(cudaMalloc((void **)&dev_b, N*N*sizeof(float)));
    HANDLE_ERROR(cudaMalloc((void **)&dev_c, N*N*sizeof(float)));

    /*initiate the square matrixes that will be multiplied*/
    for(int i=0; i<N; i++){
           for(int j=0; j<N; j++){
                  a[i][j]=float(i+3.0*j);
                  b[i][j]=float(i+2.0*j);
           }
    }

    /*load two matrixes from CPU to GPU*/
    HANDLE_ERROR(cudaMemcpy(dev_a, a, N*N*sizeof(float), cudaMemcpyHostToDevice));
 HANDLE_ERROR(cudaMemcpy(dev_b, b, N*N*sizeof(float), cudaMemcpyHostToDevice));

    /*implement the matrix multiplication on GPU*/
    matrixMult<<<blocks, threads>>>(dev_a, dev_b, dev_c, N);

    /*make sure all threads complete calculation*/
    cudaThreadSynchronize();

    /*copy the result of matrix multiplication from GPU back to CPU*/
    HANDLE_ERROR(cudaMemcpy(c, dev_c, N*N*sizeof(float), cudaMemcpyDeviceToHost));

    /*check the result of matrix multiplication*/
    bool success = true;
    for(int i=0; i<N; i++){
           for(int j=0; j<N;j++){
                  float sum=0;
                  for(int k=0;k<N;k++)
                         sum+=a[i][k]*b[k][j];
                  if (sum != c[i][j]) {
                         success = false;
                         printf("there is error in c[%d][%d]=%.2f", i, j,c[i][j] );
                  }
           }
    }


    if(success) printf("we made it!");

    /*print out the two matrixes of a and b*/
    printf("a=\n");
    for (int i = 0; i < N; i++) {
           for (int j = 0; j < N; j++)
                  printf("%0.2f,", a[i][j]);
                  printf("\n");
    }

    printf("b=\n");
    for (int i = 0; i < N; i++) {
           for (int j = 0; j < N; j++)
                  printf("%0.2f,", b[i][j]);
           printf("\n");
    }
/*print the production of matrix a and b: matrix c*/
    printf("the production matrix: c=\n");
    for (int i = 0; i < N; i++) {
           for (int j = 0; j < N; j++)
```

```
              printf("%0.2f,", c[i][j]);
       printf("\n");
}

/*free allocated GPU memory*/
cudaFree(dev_a);
cudaFree(dev_b);
cudaFree(dev_c);

return 0;
}
```

The output of code:



```
we made it!
a=
0.00,3.00,6.00,9.00,
1.00,4.00,7.00,10.00,
2.00,5.00,8.00,11.00,
3.00,6.00,9.00,12.00,
b=
0.00,2.00,4.00,6.00,
1.00,3.00,5.00,7.00,
2.00,4.00,6.00,8.00,
3.00,5.00,7.00,9.00,
the production c=
42.00,78.00,114.00,150.00,
48.00,92.00,136.00,180.00,
54.00,106.00,158.00,210.00,
60.00,120.00,180.00,240.00,
Press any key to continue . . .
```