WEEK 8: PCAP LAB
Matrix Operations on CUDA

Name: Sagnik Chatterjee
Reg: 180905478
Roll No:61
Sec:VI B

Q1. Write a program in CUDA to add two Matrices for the following specifications:

a. Each row of resultant matrix to be computed by one thread.

b. Each column of resultant matrix to be computed by one thread.

c. Each element of resultant matrix to be computed by one thread.

```
/***
Author :Sagnik Chatterjee
*/
//Q1 complete
/**
Program in CUDA to add 2 matrices for the following specs:

a. Each row of resultant matrix to be computed by one thread.

b. Each column of resultant matrix to be computed by one thread.

c. Each element of resultant matrix to be computed by one
thread.
*/

%%cu
#include <stdio.h>
#include <stdlib.h>
#include "cuda_runtime.h"
#include "device_launch_parameters.h"

const int n=3;
```

```
__global__ void vec_add_row_thread(int a[][n], int b[][n], int
c[][n])
{
    int tidx=threadIdx.x;
  if(tidx<n){
    for(int i=0;i<n;i++){
      c[tidx][i]=a[tidx][i]+b[tidx][i];
    }

  }
}
__global__ void vec_add_col_thread(int a[][n], int b[][n], int
c[][n])
{
    int tidx=threadIdx.x;
  if(tidx<n){
    for(int i=0;i<n;i++){
      c[i][tidx]=a[i][tidx]+b[i][tidx];
    }
  }
}
__global__ void vec_add_element_thread(int a[][n], int b[][n],
int c[][n])
{
    int tidx=threadIdx.x;
  int tidy=threadIdx.y;
  c[tidx][tidy]=a[tidx][tidy]+b[tidx][tidy];
}
int main()
{
    int (*d_a)[n],(*d_b)[n],(*d_c)[n];
    int size=sizeof(int);

  int a[n][n] = {{11,22,33},{45,85,64},{78,89,19}};
```

```
int b[n][n] = {{1,2,3},{4,5,6},{7,8,9}};
int c[n][n] = {{0,0,0},{0,0,0},{0,0,0}};

cudaMalloc((void **)&d_a,size*n*n);
cudaMalloc((void **)&d_b,size*n*n);
cudaMalloc((void **)&d_c,size*n*n);

cudaMemcpy(d_a,&a,size*n*n,cudaMemcpyHostToDevice);
cudaMemcpy(d_b,&b,size*n*n,cudaMemcpyHostToDevice);

vec_add_row_thread<<<1,n>>>(d_a,d_b,d_c);
  cudaMemcpy(c,d_c,size*n*n,cudaMemcpyDeviceToHost);
//matrices a and b
printf("A = \n");
for(int i=0;i<n;i++){
    for(int j=0;j<n;j++){
        printf("%d, ",a[i][j]);
    }
    printf("\n");
}
printf("B = \n");
for(int i=0;i<n;i++){
    for(int j=0;j<n;j++){
        printf("%d, ",b[i][j]);
    }
    printf("\n");
}
printf("C \n");
printf(" Part a> Single thread for a row: \n");
for(int i=0;i<n;i++){
   for(int j=0;j<n;j++){
       printf("%d ",c[i][j]);
   }
   printf("\n");
```

```
  }
  printf("\n");

  vec_add_col_thread<<<1,n>>>(d_a,d_b,d_c);
    cudaMemcpy(c,d_c,size*n*n,cudaMemcpyDeviceToHost);
  printf("Part b> Single thread for a column: \n");
  for(int i=0;i<n;i++){
     for(int j=0;j<n;j++){
         printf("%d ",c[i][j]);
     }
     printf("\n");
  }
  printf("\n");

  dim3 threadsPerBlock(n,n);
  vec_add_element_thread<<<1,threadsPerBlock>>>(d_a,d_b,d_c);
    cudaMemcpy(c,d_c,size*n*n,cudaMemcpyDeviceToHost);
  printf("Part c> Single  thread for a element:\n");
  for(int i=0;i<n;i++){
     for(int j=0;j<n;j++){
         printf("%d ",c[i][j]);
     }
     printf("\n");
  }
  cudaFree(d_a);
  cudaFree(d_b);
  cudaFree(d_c);
}
```

Screenshot:

```
A =
11, 22, 33,
45, 85, 64,
78, 89, 19,
B =
1, 2, 3,
4, 5, 6,
7, 8, 9,
C
 Part a> Single thread for a row:
12 24 36
49 90 70
85 97 28

Part b> Single thread for a column:
12 24 36
49 90 70
85 97 28

Part c> Single  thread for a element:
12 24 36
49 90 70
85 97 28
```

Q2 . Write a program in CUDA to multiply two Matrices for the following specifications:

a. Each row of resultant matrix to be computed by one thread.

b. Each column of resultant matrix to be computed by one thread.

c. Each element of resultant matrix to be computed by one thread.

```
/***
Author :Sagnik Chatterjee
*/
//Q2 complete
/**
Write a program to multiply two matrices for following:

a. Each row of resultant matrix to be computed by one thread.
```

```
b. Each column of resultant matrix to be computed by one thread.

c. Each element of resultant matrix to be computed by one
thread.
*/
%%cu
#include<stdio.h>
#include "cuda_runtime.h"
#include "device_launch_parameters.h"

const int n=3;
__global__ void vec_mul_row_thread(int a[][n], int b[][n], int
c[][n])
{
    int tidx=threadIdx.x;
  if(tidx<n){
    for(int i=0;i<n;i++){
        c[tidx][i]=0;
      for(int j=0;j<n;j++){
        c[tidx][i]+=a[tidx][j]*b[j][i];
      }
    }

  }
}
__global__ void vec_mul_col_thread(int a[][n], int b[][n], int
c[][n])
{
    int tidx=threadIdx.x;
  if(tidx<n){
    for(int i=0;i<n;i++){
        c[i][tidx]=0;
        for(int j=0;j<n;j++){
          c[i][tidx]+=a[i][j]*b[j][tidx];
```

```
        }
    }
  }
}
__global__ void vec_mul_element_thread(int a[][n], int b[][n],
int c[][n])
{
    int tidx=threadIdx.x;
  int tidy=threadIdx.y;
 c[tidx][tidy]=0;
  for(int i=0;i<n;i++)
  c[tidx][tidy]+=a[tidx][i]*b[i][tidy];
}

int main()
{
    int (*d_a)[n],(*d_b)[n],(*d_c)[n];
    int size=sizeof(int);
  int a[n][n] = {{11,22,33},{44,55,66},{77,88,99}};
  int b[n][n] = {{21,22,23},{34,35,36},{37,38,39}};
  int c[n][n] = {{0,0,0},{0,0,0},{0,0,0}};

    cudaMalloc((void **)&d_a,size*n*n);
  cudaMalloc((void **)&d_b,size*n*n);
  cudaMalloc((void **)&d_c,size*n*n);
    cudaMemcpy(d_a,&a,size*n*n,cudaMemcpyHostToDevice);
  cudaMemcpy(d_b,&b,size*n*n,cudaMemcpyHostToDevice);

    vec_mul_row_thread<<<1,n>>>(d_a,d_b,d_c);
    cudaMemcpy(c,d_c,size*n*n,cudaMemcpyDeviceToHost);

  //matrices a and b
  printf("A = \n");
  for(int i=0;i<n;i++){
```

```
    for(int j=0;j<n;j++){
        printf("%d, ",a[i][j]);
    }
    printf("\n");
}
printf("B = \n");
for(int i=0;i<n;i++){
    for(int j=0;j<n;j++){
        printf("%d, ",b[i][j]);
    }
    printf("\n");
}
printf("C \n");
printf("Part a> Single Thread for a row: \n");
for(int i=0;i<n;i++){
   for(int j=0;j<n;j++){
       printf("%d ",c[i][j]);
   }
   printf("\n");
}
printf("\n");

vec_mul_col_thread<<<1,n>>>(d_a,d_b,d_c);
  cudaMemcpy(c,d_c,size*n*n,cudaMemcpyDeviceToHost);
printf("Part b> Single thread for a column: \n");
for(int i=0;i<n;i++){
   for(int j=0;j<n;j++){
       printf("%d ",c[i][j]);
   }
   printf("\n");
}
printf("\n");

dim3 threadsPerBlock(n,n);
```

```
  vec_mul_element_thread<<<1,threadsPerBlock>>>(d_a,d_b,d_c);
    cudaMemcpy(c,d_c,size*n*n,cudaMemcpyDeviceToHost);
  printf("Part C> Using single thread for a element:\n");
  for(int i=0;i<n;i++){
     for(int j=0;j<n;j++){
         printf("%d ",c[i][j]);
     }
     printf("\n");
  }
  cudaFree(d_a);
  cudaFree(d_b);
  cudaFree(d_c);
}
```

Screenshot:

```
A =
11, 22, 33,
44, 55, 66,
77, 88, 99,
B =
21, 22, 23,
34, 35, 36,
37, 38, 39,
C
Part a> Single Thread for a row:
2200 2266 2332
5236 5401 5566
8272 8536 8800

Part b> Single thread for a column:
2200 2266 2332
5236 5401 5566
8272 8536 8800

Part C> Using single thread for a element:
2200 2266 2332
5236 5401 5566
8272 8536 8800
```