WEEK9 : Programs on Matrix using CUDA (Part1)
Name: Sagnik Chatterjee
Reg: 180905478
Roll No: 61
SEC: VIB

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.

```
% % cu
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

//each row computed by one thread : - part1
__global__ void row_one_thread(int *a, int *b, int *c, int m,
int n)
{
   int idx = threadIdx.x;
   printf("idx = %d\n", idx);
   for (int i = 0; i < m; ++i)
   {
       c[i + m * idx] = a[i + m * idx] + b[i + m * idx];
   }
}

//each col computed by one thread :- part2
__global__ void col_one_thread(int *a, int *b, int *c, int m,
int n)
{
   int idx = threadIdx.x;
   printf("idx = %d\n", idx);
```

```
    for (int i = 0; i < n; ++i, idx += m)
    {
        c[idx] = a[idx] + b[idx];
    }
}

//each element computed by one thread :- part3
__global__ void element_one_thread(int *a, int *b, int *c)
{
    int idx = blockDim.x * blockIdx.x + threadIdx.x;
    printf("idx = %d\n", idx);
    c[idx] = a[idx] + b[idx];
}

int main(void)
{
    int i, j;
    int *d_a, *d_b, *d_c, *d_d, *d_e;
    int m = 3;
    int n = 4;
    int a[m * n], b[m * n], c[m * n], d[m * n], e[m * n];

    for (i = 0; i < m * n; ++i)
    {
        a[i] = i + 1;
        b[i] = m * n - i - 1;
    }

    int size = sizeof(int) * m * n;

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

```
cudaMalloc((void **)&d_e, size);

printf("Matrix a:\n");
for (i = 0; i < n; ++i)
{
    for (j = 0; j < m; ++j)
    {
        printf("%d\t", a[i * m + j]);
    }
    printf("\n");
}
printf("\n");

printf("Matrix B:\n");
for (i = 0; i < n; ++i)
{
    for (j = 0; j < m; ++j)
    {
        printf("%d\t", b[i * m + j]);
    }
    printf("\n");
}
printf("\n");

// Copy inputs to device
cudaMemcpy(d_a, &a, size, cudaMemcpyHostToDevice);
cudaMemcpy(d_b, &b, size, cudaMemcpyHostToDevice);

row_one_thread<<<1, n>>>(d_a, d_b, d_c, m, n);
col_one_thread<<<1, n>>>(d_a, d_b, d_d, m, n);
element_one_thread<<<m, n>>>(d_a, d_b, d_e);

cudaMemcpy(&c, d_c, size, cudaMemcpyDeviceToHost);
cudaMemcpy(&d, d_d, size, cudaMemcpyDeviceToHost);
```

```
cudaMemcpy(&e, d_e, size, cudaMemcpyDeviceToHost);

printf("Part 1> A+B:\n");
for (i = 0; i < n; ++i)
{
    for (j = 0; j < m; ++j)
    {
        printf("%d\t", c[i * m + j]);
    }
    printf("\n");
}
printf("\n");

printf("Part 2> A+B:\n");
for (i = 0; i < n; ++i)
{
    for (j = 0; j < m; ++j)
    {
        printf("%d\t", d[i * m + j]);
    }
    printf("\n");
}
printf("\n");

printf("Part 3> A+B:\n");
for (i = 0; i < n; ++i)
{
    for (j = 0; j < m; ++j)
    {
        printf("%d\t", e[i * m + j]);
    }
    printf("\n");
}
printf("\n");
```

```
    cudaFree(d_a);
    cudaFree(d_b);
    cudaFree(d_c);
    cudaFree(d_d);
    cudaFree(d_e);

    return 0;
}
```

Output:

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.

```
%%cu
#include <stdio.h>
#include <stdlib.h>
```

```
#include <math.h>


//each row computed by one thread : - part1
__global__ void row_one_thread (int *a, int *b, int *c,int m
,int n) {
   int idx = threadIdx.x;
   printf("idx = %d\n", idx);
   for (int i = 0; i < m; ++i) {
       c[i + m * idx] = a[i + m * idx] * b[i + m * idx];
   }
}

//each col computed by one thread :- part2
__global__ void col_one_thread (int *a, int *b, int *c,int m
,int n ) {
   int idx = threadIdx.x;
   printf("idx = %d\n", idx);
   for (int i = 0; i < n; ++i, idx += m) {
       c[idx] = a[idx] * b[idx];
   }
}

//each element computed by one thread :- part3
__global__ void element_one_thread (int *a, int *b, int *c ) {
   int idx = blockDim.x * blockIdx.x + threadIdx.x;
   printf("idx = %d\n", idx);
   c[idx] = a[idx] * b[idx];
}

int main(void){
   int i,j;
   int *d_a,*d_b,*d_c,*d_d,*d_e;
   int m =3;
```

```
int n =4 ;
int a[m*n],b[m*n],c[m*n],d[m*n],e[m*n];

for (i = 0; i < m * n; ++i) {
    a[i] = i + 1;
    b[i] = m * n - i - 1;
}

int size = sizeof(int) * m * n;

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

printf("Matrix a:\n");
for (i = 0; i < n; ++i) {
    for (j = 0; j < m; ++j) {
        printf("%d\t", a[i * m + j]);
    }
    printf("\n");
}
printf("\n");

printf("Matrix B:\n");
for (i = 0; i < n; ++i) {
    for (j = 0; j < m; ++j) {
        printf("%d\t", b[i * m + j]);
    }
    printf("\n");
}
printf("\n");
```

```
 // Copy inputs to device
cudaMemcpy(d_a, &a, size, cudaMemcpyHostToDevice);
cudaMemcpy(d_b, &b, size, cudaMemcpyHostToDevice);

row_one_thread<<<1,n>>>(d_a,d_b,d_c,m,n);
col_one_thread<<<1, n>>>(d_a, d_b, d_d,m,n);
element_one_thread<<<m,n>>>(d_a,d_b,d_e);

cudaMemcpy(&c, d_c, size, cudaMemcpyDeviceToHost);
cudaMemcpy(&d, d_d, size, cudaMemcpyDeviceToHost);
cudaMemcpy(&e, d_e, size, cudaMemcpyDeviceToHost);

printf("Part 1> A*B:\n");
for (i = 0; i < n; ++i) {
    for (j = 0; j < m; ++j) {
        printf("%d\t", c[i * m + j]);
    }
    printf("\n");
}
printf("\n");


printf("Part 2> A*B:\n");
for (i = 0; i < n; ++i) {
    for (j = 0; j < m; ++j) {
        printf("%d\t", d[i * m + j]);
    }
    printf("\n");
}
printf("\n");

printf("Part 3> A*B:\n");
for (i = 0; i < n; ++i) {
    for (j = 0; j < m; ++j) {
```

```
            printf("%d\t", e[i * m + j]);
        }
        printf("\n");
    }
    printf("\n");


    cudaFree(d_a);
    cudaFree(d_b);
    cudaFree(d_c);
    cudaFree(d_d);
    cudaFree(d_e);

    return 0;
}
```

Output: