ARUNIMA SINGH THAKUR

PP LAB 6

31, C, CSE

180905218

7TH MAY 2021

Question1

Write a program in CUDA to add two Matrices for the following specications:

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.h>
#include<stdlib.h>

__global__ void AddRow(int *a,int *b,int *c,int m,int n)
{
  int id=threadIdx.x;
  if(id<m)
  {
  for(int i=0;i<n;i++)
  {
    c[id*n+i]=a[id*n+i]+b[id*n+i];
  }
  }
}

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

```
}

__global__ void AddElement(int *a,int *b,int *c,int m,int n)
{
  int id=threadIdx.x;
  if(id<m*n)
  {
      c[id]=a[id]+b[id];
  }

}

int main()
{
  int m=3,n=3;
  //declare matrices
  int a[3][3]={{1,1,1},{2,2,2},{3,3,3}};
  int b[3][3]={{1,1,1},{2,2,2},{3,3,3}};
  int c[m][n];
  int *d_a,*d_b,*d_c;
  //allocate memory
  int size=m*n*sizeof(int);
  cudaMalloc((void**)&d_a,size);
  cudaMalloc((void**)&d_b,size);
  cudaMalloc((void**)&d_c,size);
  //copy from host to device
  cudaMemcpy(d_a,a,size,cudaMemcpyHostToDevice);
  cudaMemcpy(d_b,b,size,cudaMemcpyHostToDevice);

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

```
    // Launch add() kernels on GPU
    //Number of blocks=1 threads=m
    printf("\n 1 a) Add row per Thread\n");
    AddRow<<<1,m>>>(d_a, d_b, d_c,m,n);
    cudaMemcpy(c, d_c,size,cudaMemcpyDeviceToHost);
    for(int i=0;i<m;i++)
    {
     for(int j=0;j<n;j++)
     {
       printf("%d ",c[i][j]);
     }
     printf("\n");
    }
    //Number of blocks=1 threads=n
    printf("\n 1 b)Add column per Thread\n");
    AddCol<<<1,n>>>(d_a, d_b, d_c,m,n);
    cudaMemcpy(c, d_c,size,cudaMemcpyDeviceToHost);
    for(int i=0;i<m;i++)
    {
     for(int j=0;j<n;j++)
     {
       printf("%d ",c[i][j]);
     }
     printf("\n");
    }
    //Number of blocks=1 threads=m*n
    printf("\n 1 c)Add element per Thread\n");
    AddElement<<<1,m*n>>>(d_a, d_b, d_c,m,n);
    cudaMemcpy(c, d_c,size,cudaMemcpyDeviceToHost);
    for(int i=0;i<m;i++)
    {
     for(int j=0;j<n;j++)
     {
       printf("%d ",c[i][j]);
     }
     printf("\n");
    }
}
```

```
Matrix A:
1 1 1
2 2 2
3 3 3
Matrix B:
1 1 1
2 2 2
3 3 3

 1 a) Add row per Thread
2 2 2
4 4 4
6 6 6

 1 b)Add column per Thread
2 2 2
4 4 4
6 6 6

 1 c)Add element per Thread
2 2 2
4 4 4
6 6 6
```

Question2

Write a program in CUDA to multiply two Matrices for the following specications:

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 "cuda_runtime.h"
#include "device_launch_parameters.h"
#include <stdlib.h>
#include<stdio.h>
#define M1 6
#define N1 5
#define M2 5
#define N2 4
__global__ void mul_row_thread (int *A, int *B, int *C)
{
 int idx =threadIdx.x;
```

```
 for(int j=0;j<N2;j++)
 {
 int sum=0;
 for(int k=0;k<N1;k++)
 {
 sum+=A[idx*N1+k]*B[k*N2+j];
 }
 C[idx*N2+j] = sum;
 }
}
__global__ void mul_col_thread (int *A, int *B, int *C)
{
 int idx =threadIdx.x;
 for(int j=0;j<M1;j++)
 {
 int sum=0;
 for(int k=0;k<N1;k++)
 {
 sum+=A[idx*N1+k]*B[k*N2+j];
 }
 C[idx*N2+j] = sum;
 }

}
__global__ void mul_ele_thread (int *A, int *B, int *C)
{
 int col=threadIdx.x;
 int row=blockIdx.x;
 int sum=0;
 for(int k=0;k<N1;k++)
 {
 sum+=A[row*N1+k]*B[k*N2+col];
 }
 C[row*N2+col]=sum;

}
int main ()
{
 // Host copies of the variables
 int A[M1][N1], B[M2][N2], C1[M1][N2],C2[M1][N2],C3[M1][N2];
 if(N1!=M2)
 {
 printf("Dimensions mismatch\n");
 exit(0);
 }
 for(int i=0;i<M1;i++)
 {
 for(int j=0;j<N1;j++)
```

```
{
A[i][j]=rand()%10;
}
}
for(int i=0;i<M2;i++)
{
for(int j=0;j<N2;j++)
{
B[i][j]=rand()%10;
}
}
int *d_a, *d_b, *d_c1,*d_c2 ,*d_c3;
int sizeA = sizeof(int) * M1 * N1;
int sizeB = sizeof(int) * M2 * N2;
int sizeC = sizeof(int) * M1 * N2;
cudaMalloc((void**)&d_a, sizeA);
cudaMalloc((void**)&d_b, sizeB);
cudaMalloc((void**)&d_c1, sizeC);
cudaMalloc((void**)&d_c2, sizeC);
cudaMalloc((void**)&d_c3, sizeC);
cudaMemcpy(d_a, &A, sizeA, cudaMemcpyHostToDevice);
cudaMemcpy(d_b, &B, sizeB, cudaMemcpyHostToDevice);
mul_row_thread<<<1, M1>>>(d_a, d_b, d_c1);
cudaMemcpy(&C1, d_c1, sizeC, cudaMemcpyDeviceToHost);
mul_col_thread<<<1,N2>>>(d_a,d_b,d_c2);
cudaMemcpy(&C2, d_c2, sizeC, cudaMemcpyDeviceToHost);
mul_ele_thread<<<M1,N2>>>(d_a, d_b,d_c3);
cudaMemcpy(&C3, d_c3, sizeC, cudaMemcpyDeviceToHost);
int i,j;
printf("A:\n");
for (i = 0; i < M1; ++i) {
for (j = 0; j < N1; ++j) {
printf("%d\t", A[i][j]);
}
printf("\n");
}
printf("\n");
printf("B:\n");
for (i = 0; i < M2; ++i) {
for (j = 0; j < N2; ++j) {
printf("%d\t", B[i][j]);
}
printf("\n");
}
printf("\n");
printf("A * B:\n");
printf("\na)Each Row is computed by one thread:\n");
for (i = 0; i < M1; ++i) {
```

```
 for (j = 0; j < N2; ++j) {
 printf("%d\t", C1[i][j]);
 }
 printf("\n");
 }
 printf("\n");
 printf("\nb)Each Column is computed by one thread:\n");
 for (i = 0; i < M1; ++i) {
 for (j = 0; j < N2; ++j)
 {
 printf("%d\t", C2[i][j]);
 }
 printf("\n");
 }
 printf("\n");
 printf("\nc)Each element is computed by one thread:\n");
 for (i = 0; i < M1; ++i) {
 for (j = 0; j < N2; ++j) {
 printf("%d\t", C3[i][j]);
 }
 printf("\n");
 }
 printf("\n");
 cudaFree(d_a);
 cudaFree(d_b);
 cudaFree(d_c1);
 cudaFree(d_c2);
 cudaFree(d_c3);
 getchar();
 return 0;
}
```

```
A:
3       6       7       5       3
5       6       2       9       1
2       7       0       9       3
6       0       6       2       6
1       8       7       9       2
0       2       3       7       5

B:
9       2       2       8
9       7       3       6
1       2       9       3
1       9       4       7
8       4       5       0

A * B:

a)Each Row is computed by one thread:
117     119     122     116
118     141     87      145
114     146     76      121
110     66      104     80
113     161     135     140
68      103     86      70
```

```
b)Each Column is computed by one thread:
117     119     122     116
80      116     87      145
125     101     76      121
97      64      104     80
76      104     0       0
0       0       0       0


c)Each element is computed by one thread:
117     119     122     116
118     141     87      145
114     146     76      121
110     66      104     80
113     161     135     140
68      103     86      70
```