**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:**

1. **Each row of resultant matrix to be computed by one thread.**
2. **Each column of resultant matrix to be computed by one thread.**
3. **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++)

    {

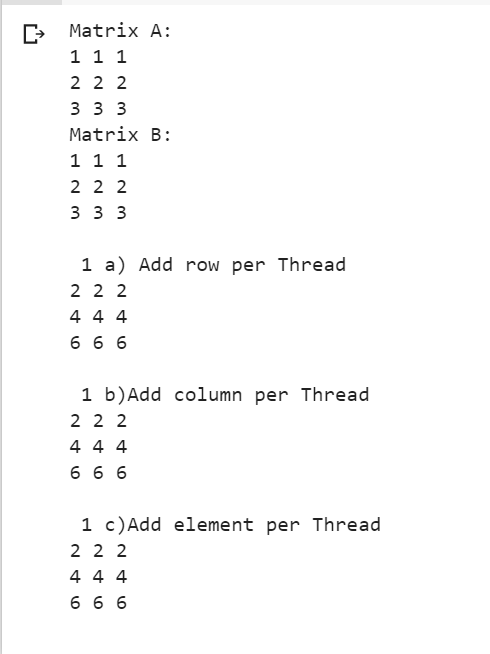
      printf("%d ",c[i][j]);

    }

    printf("\n");

   }

}

****

**Question2**

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

1. **Each row of resultant matrix to be computed by one thread.**
2. **Each column of resultant matrix to be computed by one thread**
3. **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);

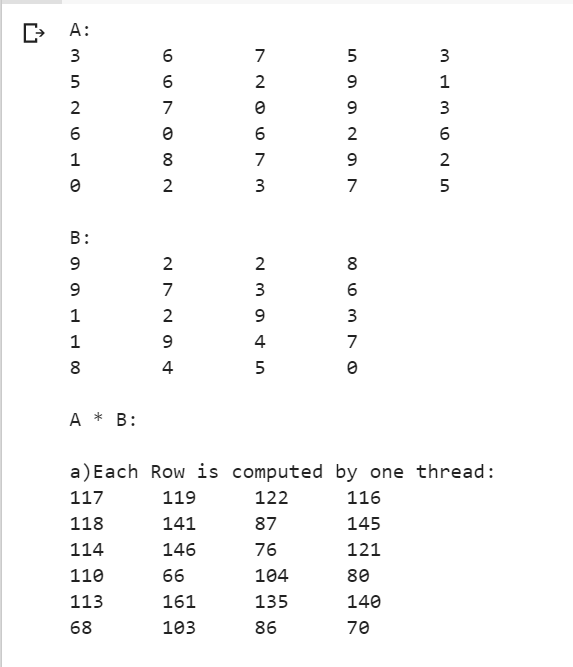
 cudaFree(d\_c2);

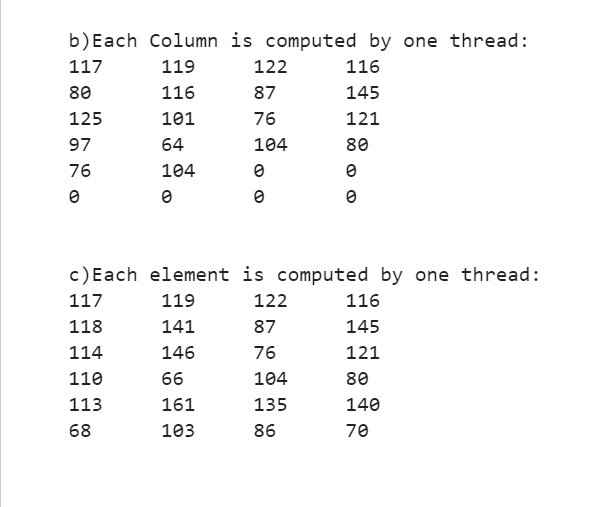
 cudaFree(d\_c3);

 getchar();

 return 0;

}

****

****