Assignment – 5

1.

%%cu

#define N 6

#define n 2

#include<iostream>

using namespace std;

\_\_global\_\_ void dp(int \*a,int \*b,int \*c);

int main()

{

    int a[N],b[N],c;

    for(int i = 0;i<N;i++)

    {

        a[i] = i+1;

        b[i] = 2\*i+1;

    }

    int \*d\_a,\*d\_b,\*d\_c;

    int b1 = N\*sizeof(int);

    cudaMalloc(&d\_a,b1);

    cudaMalloc(&d\_b,b1);

    cudaMalloc(&d\_c,sizeof(int));

    cudaMemcpy(d\_a,&a,b1,cudaMemcpyHostToDevice);

    cudaMemcpy(d\_b,&b,b1,cudaMemcpyHostToDevice);

    dp<<<(int)ceil((float)N/n),n>>>(d\_a,d\_b,d\_c);

    cudaMemcpy(&c,d\_c,sizeof(int),cudaMemcpyDeviceToHost);

    cout<<c<<endl;

    cudaFree(d\_a);

    cudaFree(d\_b);

    cudaFree(d\_c);

    return 0;

}

\_\_global\_\_ void dp(int \*a,int \*b,int \*c)

{

    \*c = 0;

     \_\_shared\_\_ int temp[N];

    int t = blockIdx.x\*blockDim.x + threadIdx.x;

        temp[threadIdx.x] = a[t]\*b[t];

    \_\_syncthreads();

    if(threadIdx.x==0){

        int s = 0;

        for(int i = 0;i<n;i++)

        {

            s = s  + temp[i];

        }

    atomicAdd(c,s);

    }

}



2.

%%cu

#define b 4

#include<iostream>

#include<vector>

using namespace std;

\_\_global\_\_ void addk (int\* d\_a,int\* d\_b,int\* d\_sum,size\_t ss,int m,int n)

{

    int x = blockDim.x\*blockIdx.x + threadIdx.x;

    int y = blockDim.y\*blockIdx.y + threadIdx.y;

    int index = ss/sizeof(int);

    if(x<m&&y<n)

    {

        d\_sum[x\*index + y] = d\_a[x\*index + y] + d\_b[x\*index + y];

    }

}

void printMat(int\* matsum,int m,int n)

{

    for(int i = 0;i<m;i++)

    {

        for(int j = 0;j<n;j++)

        {

            cout<<matsum[i\*n+j]<<"  ";

        }cout<<endl;

    }

}

int main(){

    int m,n;

    //cin>>m>>n;

    m = b,n=b;

    int \*mat1 = (int\*)malloc(m\*n\*sizeof(int));

    int \*mat2 = (int\*)malloc(m\*n\*sizeof(int));

    int \*matsum = (int\*)malloc(m\*n\*sizeof(int));

    for(int i = 0;i<m\*n;i++)

    {

        mat1[i] = i;

        mat2[i] = i;

    }

    int \*d\_a,\*d\_b,\*d\_sum;

    size\_t ss;

    cudaMallocPitch((void\*\*)&d\_a,&ss,n\*sizeof(int),m);

    cudaMallocPitch((void\*\*)&d\_b,&ss,n\*sizeof(int),m);

    cudaMallocPitch((void\*\*)&d\_sum,&ss,n\*sizeof(int),m);

    cudaMemcpy2D(d\_a,ss,mat1,n\*sizeof(int),n\*sizeof(int),m,cudaMemcpyHostToDevice);

    cudaMemcpy2D(d\_b,ss,mat2,n\*sizeof(int),n\*sizeof(int),m,cudaMemcpyHostToDevice);

    dim3 block(1,1,1);

    dim3 tpb(b,b,1);

    block.x =((m/b)+((m%b)==0?0:1));

    block.y =((m/b)+((m%b)==0?0:1));

    addk<<<block,tpb>>>(d\_a,d\_b,d\_sum,ss,m,n);

    cudaThreadSynchronize();

    cudaMemcpy2D(matsum,n\*sizeof(int),d\_sum,ss,n\*sizeof(int),m,cudaMemcpyDeviceToHost);

    cudaFree(d\_a);

    cudaFree(d\_b);

    cudaFree(d\_sum);

    printMat(matsum,m,n);

}

