**ARUNIMA SINGH THAKUR**

**180905218**

**31, C, CSE**

**PP LAB5**

**5TH MAY 2021**

**QUESTION 1:**

**Write and execute a program in CUDA to add two vectors of length N to meet the following requirements using 3 different kernels**

**a) block size as N**

**b) N threads within a block**

**c) Keep the number of threads per block as 256 (constant) and vary the number of blocks to handle N elements.**

%%cu

#include<stdio.h>

#include "cuda\_runtime.h"

#include "device\_launch\_parameters.h"

const int n=272;

\_\_global\_\_ void vec\_add\_blocks(int\* a, int\* b, int\* c)

{

  int tid=blockIdx.x;

  if(tid<n){

      c[tid]=a[tid]+b[tid];

  }

}

\_\_global\_\_ void vec\_add\_threads(int\* a, int\* b, int\* c)

{

  int tid=threadIdx.x;

  if(tid<n){

      c[tid]=a[tid]+b[tid];

  }

}

\_\_global\_\_ void vec\_add\_blocks\_threads(int\* a, int\* b, int\* c)

{

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

  while ( tid < n ){

      c[tid]=a[tid]+b[tid];

      tid+= blockDim.x \* gridDim.x ;

  }

}

int main()

{

  int a[n],b[n],c[n];

  int \*d\_a,\*d\_b,\*d\_c;

  int size=sizeof(int);

  cudaMalloc((void \*\*)&d\_a,size\*n);

  cudaMalloc((void \*\*)&d\_b,size\*n);

  cudaMalloc((void \*\*)&d\_c,size\*n);

  for(int i=0;i<n;i++){

      a[i]=i;

      b[i]=3\*i;

  }

  cudaMemcpy(d\_a,&a,size\*n,cudaMemcpyHostToDevice);

  cudaMemcpy(d\_b,&b,size\*n,cudaMemcpyHostToDevice);

  vec\_add\_blocks<<<n,1>>>(d\_a,d\_b,d\_c);

  cudaMemcpy(&c,d\_c,size\*n,cudaMemcpyDeviceToHost);

  printf("Using blocks: ");

  for(int i=0;i<n;i++){

      printf("%d ",c[i]);

  }

  printf("\n");

  vec\_add\_threads<<<1,n>>>(d\_a,d\_b,d\_c);

  cudaMemcpy(&c,d\_c,size\*n,cudaMemcpyDeviceToHost);

  printf("Using n threads:");

  for(int i=0;i<n;i++){

      printf("%d ",c[i]);

  }

  printf("\n");

  vec\_add\_blocks\_threads<<<(n+255)/256,256>>>(d\_a,d\_b,d\_c);

  cudaMemcpy(&c,d\_c,size\*n,cudaMemcpyDeviceToHost);

  printf("Varying blocks and 256 threads:");

  for(int i=0;i<n;i++){

      printf("%d ",c[i]);

  }

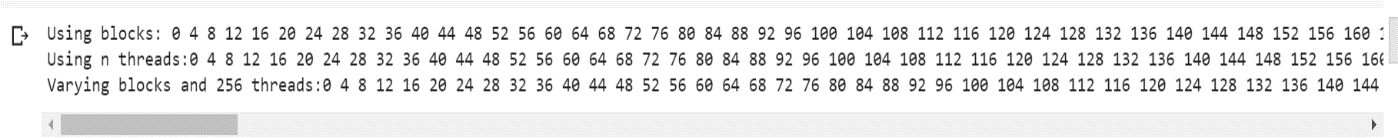
  cudaFree(d\_a);

  cudaFree(d\_b);

  cudaFree(d\_c);

}

**SAMPLE OUTPUT:**



**QUESTION 2:**

**Write and execute a CUDA program to read an array of N integer values. Sort the array in parallel using parallel selection sort and store the result in another array.**

%%cu

#include<stdio.h>

#include "cuda\_runtime.h"

#include "device\_launch\_parameters.h"

const int n=10;

\_\_global\_\_ void selection\_sort\_parallel(int\* a, int\* c)

{

  int tid=threadIdx.x;

  int pos=0;

  if(tid<n){

      for(int j=0;j<n;j++){

          if((a[j]<a[tid])||(a[j]==a[tid])&&(j<tid)){

              pos=pos+1;

          }

      }

      c[pos]=a[tid];

  }

}

int main()

{

  int c[n];

  int \*d\_a,\*d\_c;

  int size=sizeof(int);

  int a[10]={5,3,69,1,420,4,5,7,10,12};

  cudaMalloc((void \*\*)&d\_a,size\*n);

  cudaMalloc((void \*\*)&d\_c,size\*n);

  cudaMemcpy(d\_a,&a,size\*n,cudaMemcpyHostToDevice);

  cudaMemcpy(d\_c,&c,size\*n,cudaMemcpyHostToDevice);

  selection\_sort\_parallel<<<1,n>>>(d\_a,d\_c);

  cudaMemcpy(&c,d\_c,size\*n,cudaMemcpyDeviceToHost);

  printf("Sorted array ");

  for(int i=0;i<n;i++){

      printf("%d ",c[i]);

  }

  cudaFree(d\_a);

  cudaFree(d\_c);

}

**SAMPLE INPUT:**

**5,3,69,1,420,4,5,7,10,12**

**SAMPLE OUTPUT:**

****

**QUESTION 3:**

**Write a execute a CUDA program to read an integer array of size N. Sort this array using odd-even transposition sorting. Use 2 kernels.**

%%cu

#include <stdio.h>

#include <stdlib.h>

\_\_global\_\_

void oddsort(int \*a,int n)

{

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

    int el1=2\*i+1;int el2=2\*i+2;

    if(el1<n && el2<n)

    {

        if(a[el1]>a[el2])

        {

            int temp=a[el1];

            a[el1]=a[el2];

            a[el2]=temp;

        }

    }

}

\_\_global\_\_

void evensort(int \*a,int n)

{

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

    int el1=2\*i;int el2=2\*i+1;

    if(el1<n && el2<n)

    {

        if(a[el1]>a[el2])

        {

            int temp=a[el1];

            a[el1]=a[el2];

            a[el2]=temp;

        }

    }

}

void hostfunc(int \*a,int n)

{

    int size = n\*sizeof(int);

    int \*d\_a;

    cudaMalloc((void\*\*)&d\_a,size);

    cudaMemcpy(d\_a,a,size,cudaMemcpyHostToDevice);

    for(int i=0;i<n/2;i++)

    {

        evensort<<<1,256>>>(d\_a,n);

        oddsort<<<1,256>>>(d\_a,n);

    }

    cudaMemcpy(a,d\_a,size,cudaMemcpyDeviceToHost);

    cudaFree(d\_a);

}

int main()

{

    int n=10;

    int a[10]={5,3,6,1,7,9,2,10,4,8};

    hostfunc(a,n);

    for(int i=0;i<n;i++)

    {

        printf("%d ",a[i]);

    }

    printf("\n");

}

**SAMPLE INPUT:**

**5,3,6,1,7,9,2,10,4,8**

**SAMPLE OUTPUT:**

