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

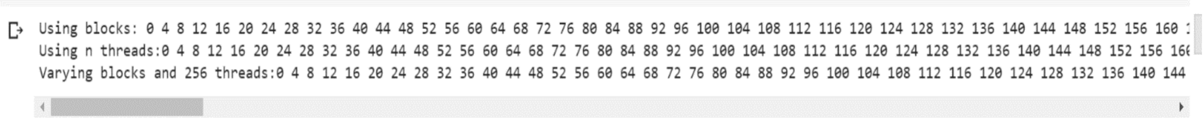
a) block size as N

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:



```
Using blocks: 0 4 8 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72 76 80 84 88 92 96 100 104 108 112 116 120 124 128 132 136 140 144 148 152 156 160 1
Using n threads:0 4 8 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72 76 80 84 88 92 96 100 104 108 112 116 120 124 128 132 136 140 144 148 152 156 16
Varying blocks and 256 threads:0 4 8 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72 76 80 84 88 92 96 100 104 108 112 116 120 124 128 132 136 140 144
```

QUESTION 2:

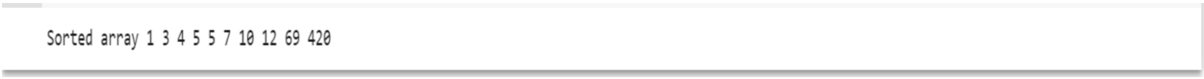
Write and execute a CUDA program to read an array of N integer values. Sort the array in parallel using parallel selection sort and store the result in another array.

```
%%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:



```
Sorted array 1 3 4 5 5 7 10 12 69 420
```

QUESTION 3:

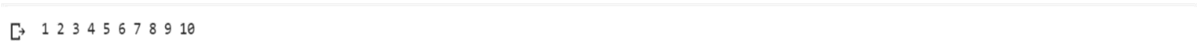
Write a execute a CUDA program to read an integer array of size N. Sort this array using odd-even transposition sorting. Use 2 kernels.

```
%%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:



```
1 2 3 4 5 6 7 8 9 10
```