WEEK 5

Devesh Mahajan

180905242

B-2

Roll No: 34

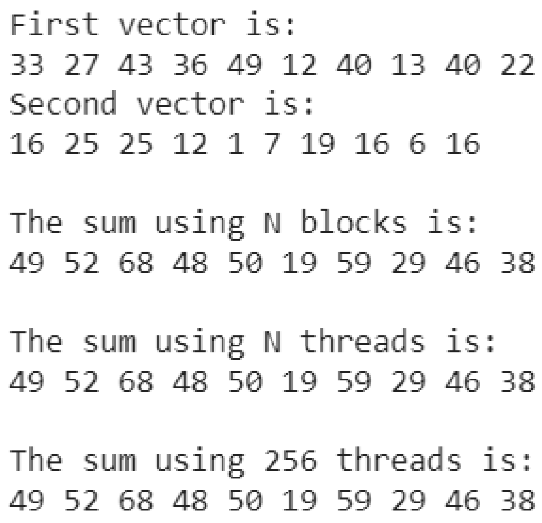
1. Write and execute a program in CUDA to add two vectors of length N to meet the following requirements using 3 different kernels

   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.

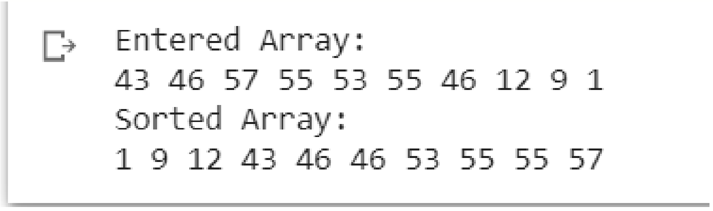
```
%%cu

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

__global__ void firstKernel(int *a, int *b, int *c, int n)
{
  int i = blockIdx.x;
  c[i] = a[i] + b[i];
}

__global__ void secondKernel(int *a, int *b, int *c, int n)
{
  int i = threadIdx.x;
  c[i] = a[i] + b[i];
}

__global__ void thirdKernel(int *a, int *b, int *c, int n)
{
  int i = threadIdx.x + blockIdx.x *blockDim.x;
  if (i < n)
    c[i] = a[i] + b[i];
}

int main()
{
  int *d_a, *d_b, *d_c, *d_d, *d_e;
  int n = 10;
  int a[n],b[n];
  for (int i = 0; i < n; i++)
  {
    a[i] = rand() % 50;
    b[i] = rand() % 30;
  }
  int c[n], d[n], e[n];
  printf("First vector is: \n");
  for (int i = 0; i < n; i++)
    printf("%d ", a[i]);
  printf("\n");
  printf("Second vector is: \n");
  for (int i = 0; i < n; i++)
    printf("%d ", b[i]);
  printf("\n");
  cudaMalloc((void **) &d_a, n* sizeof(int));
  cudaMalloc((void **) &d_b, n* sizeof(int));
  cudaMalloc((void **) &d_c, n* sizeof(int));
  cudaMalloc((void **) &d_d, n* sizeof(int));
  cudaMalloc((void **) &d_e, n* sizeof(int));
  cudaMemcpy(d_a, &a, n* sizeof(int), cudaMemcpyHostToDevice);
  cudaMemcpy(d_b, &b, n* sizeof(int), cudaMemcpyHostToDevice);
  firstKernel <<<n, 1>>> (d_a, d_b, d_c, n);
  secondKernel <<<1, n>>> (d_a, d_b, d_d, n);
  thirdKernel <<<ceil(n / 256.0), 256>>> (d_a, d_b, d_e, n);
  cudaMemcpy(&c, d_c, n* sizeof(int), cudaMemcpyDeviceToHost);
  cudaMemcpy(&d, d_d, n* sizeof(int), cudaMemcpyDeviceToHost);
  cudaMemcpy(&e, d_e, n* sizeof(int), cudaMemcpyDeviceToHost);
  printf("\n");
  printf("The sum using N blocks is: \n");
  for (int i = 0; i < n; i++)
    printf("%d ", c[i]);
  printf("\n\n");
  printf("The sum using N threads is: \n");
  for (int i = 0; i < n; i++)
    printf("%d ", d[i]);
  printf("\n\n");
  printf("The sum using 256 threads is: \n");
  for (int i = 0; i < n; i++)
    printf("%d ", e[i]);
  printf("\n\n");
  cudaFree(d_a);
  cudaFree(d_b);
  cudaFree(d_c);
  cudaFree(d_d);
  cudaFree(d_e);
  return 0;
}
```



```
First vector is:
33 27 43 36 49 12 40 13 40 22
Second vector is:
16 25 25 12 1 7 19 16 6 16

The sum using N blocks is:
49 52 68 48 50 19 59 29 46 38

The sum using N threads is:
49 52 68 48 50 19 59 29 46 38

The sum using 256 threads is:
49 52 68 48 50 19 59 29 46 38
```

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.

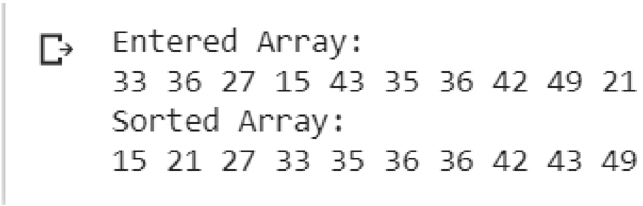
```
%%cu

#include <stdio.h>
#include <stdlib.h>

__global__ void parallelKernel(int *a, int n)
{
  int i = threadIdx.x;
  int data = a[i];
  int pos = 0;
  for (int j = 0; j < n; j++)
  {
    if (a[j] < data || (a[j] == data && j < i))
      pos++;
  }

  a[pos] = data;
}

void sortArr(int* h_arr, int n){
  int* d_arr = NULL;
  int size = n*sizeof(int);
  cudaError_t err = cudaSuccess;
  err = cudaMalloc((void **)&d_arr,size);
  if(err != cudaSuccess){
    printf("%s\n",cudaGetErrorString(err));
    exit(EXIT_FAILURE);
  }
  err = cudaMemcpy(d_arr,h_arr,size,cudaMemcpyHostToDevice);
  if(err != cudaSuccess){
    printf("%s\n",cudaGetErrorString(err));
    exit(EXIT_FAILURE);
  }
  parallelKernel<<<1,n>>>(d_arr,n);
  cudaMemcpy(h_arr,d_arr,size,cudaMemcpyDeviceToHost);
  cudaFree(d_arr);
}


int main(){
  int n = 10;
  int* arr = (int*)malloc(n*sizeof(int));
  for(int i=0;i<n;i++){
    arr[i] = rand()%50;
  }
  printf("Entered Array: \n");
  for(int i=0;i<10;i++){
    printf("%d ",arr[i]);
  }
  printf("\nSorted Array: \n");
  sortArr(arr,n);
  for(int i=0;i<10;i++){
    printf("%d ",arr[i] );
  }
  return 0;
}
```



```
Entered Array:
43 46 57 55 53 55 46 12 9 1
Sorted Array:
1 9 12 43 46 46 53 55 55 57
```

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>
#include <cuda.h>
#include <cuda_runtime.h>

__global__ void oddKernel(int* arr, int n){
  int i = blockIdx.x*blockDim.x+threadIdx.x;
  if(i<n-1 && i%2!=0){
    if(arr[i] > arr[i+1]){
      int temp = arr[i];
      arr[i] = arr[i+1];
      arr[i+1] = temp;
    }
  }
}
__global__ void evenKernel(int* arr, int n){
  int i = blockIdx.x*blockDim.x+threadIdx.x;
  if(i<n-1 && i%2==0){
    if(arr[i] > arr[i+1]){
      int temp = arr[i];
      arr[i] = arr[i+1];
      arr[i+1] = temp;
    }
  }
}
void oddEvenSort(int* h_arr, int n){
  int* d_arr = NULL;
  int size = n*sizeof(int);
  cudaError_t err = cudaSuccess;
  err = cudaMalloc((void **)&d_arr,size);
  if(err != cudaSuccess){
    printf("%s\n",cudaGetErrorString(err));
    exit(EXIT_FAILURE);
  }
  err = cudaMemcpy(d_arr,h_arr,size,cudaMemcpyHostToDevice);
  if(err != cudaSuccess){
    printf("%s\n",cudaGetErrorString(err));
    exit(EXIT_FAILURE);
  }
  int i = 0;
  while(i<=n/2){
    oddKernel<<<1,n>>>(d_arr,n);
    evenKernel<<<1,n>>>(d_arr,n);
    i++;
  }
  cudaMemcpy(h_arr,d_arr,size,cudaMemcpyDeviceToHost);
  cudaFree(d_arr);
}
int main(){
  int n = 10;
  int* arr = (int*)malloc(n*sizeof(int));
  for(int i=0;i<n;i++){
    arr[i] = rand()%50;
  }
  printf("Entered Array: \n");
  for(int i=0;i<10;i++){
    printf("%d ",arr[i]);
  }
  printf("\nSorted Array: \n");
  oddEvenSort(arr,n);
  for(int i=0;i<10;i++){
    printf("%d ",arr[i] );
  }
  return 0;
}
```



```
Entered Array: 
33 36 27 15 43 35 36 42 49 21 
Sorted Array: 
15 21 27 33 35 36 36 42 43 49 
```