Name : Tushar Pathak
RegNumber : 230905396
Roll Number: B48

Lab 6: Programs for Array in CUDA

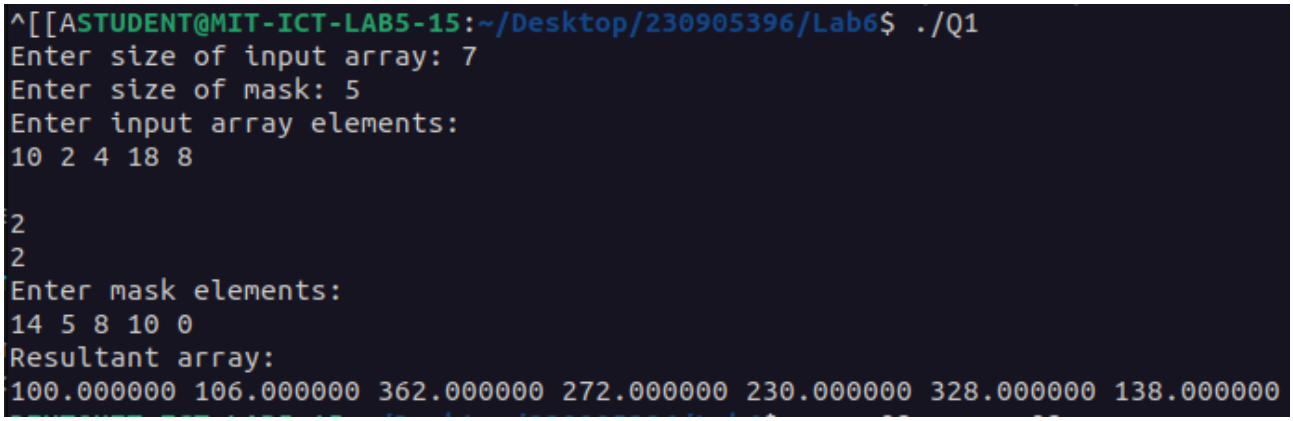
Q1). Write a program in CUDA which performs convolution operation on ID input array of size N amd width using a mask arrary M of size mask_width to produce the resultant one dimensional airny P of size width.

Code:

```
#include <stdio.h>
#include <cuda.h>

__global__ void convolution1D(float *N, float *M, float *P,
                              int width, int mask_width) {

    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int half = mask_width / 2;

    if (i < width) {
        float sum = 0.0f;

        for (int j = 0; j < mask_width; j++) {
            int idx = i - half + j;
            if (idx >= 0 && idx < width)
                sum += N[idx] * M[j];
        }
        P[i] = sum;
    }
}

int main() {

    int width, mask_width;

    printf("Enter size of input array: ");
    scanf("%d", &width);

    printf("Enter size of mask: ");
    scanf("%d", &mask_width);

    float *h_N = (float*)malloc(width * sizeof(float));
    float *h_M = (float*)malloc(mask_width * sizeof(float));
    float *h_P = (float*)malloc(width * sizeof(float));

    printf("Enter input array elements:\n");
    for (int i = 0; i < width; i++)
        scanf("%f", &h_N[i]);

    printf("Enter mask elements:\n");
    for (int i = 0; i < mask_width; i++)
        scanf("%f", &h_M[i]);
```

```
    float *d_N, *d_M, *d_P;

    cudaMalloc(&d_N, width * sizeof(float));
    cudaMalloc(&d_M, mask_width * sizeof(float));
    cudaMalloc(&d_P, width * sizeof(float));

    cudaMemcpy(d_N, h_N, width * sizeof(float), cudaMemcpyHostToDevice);
    cudaMemcpy(d_M, h_M, mask_width * sizeof(float), cudaMemcpyHostToDevice);

    dim3 blockDim(256);
    dim3 gridDim((width + blockDim.x - 1) / blockDim.x);

    convolution1D<<<gridDim, blockDim>>>(d_N, d_M, d_P, width, mask_width);

    cudaMemcpy(h_P, d_P, width * sizeof(float), cudaMemcpyDeviceToHost);

    printf("Resultant array:\n");
    for (int i = 0; i < width; i++)
        printf("%f ", h_P[i]);

    cudaFree(d_N); cudaFree(d_M); cudaFree(d_P);
    free(h_N); free(h_M); free(h_P);

    return 0;
}
```

Output :

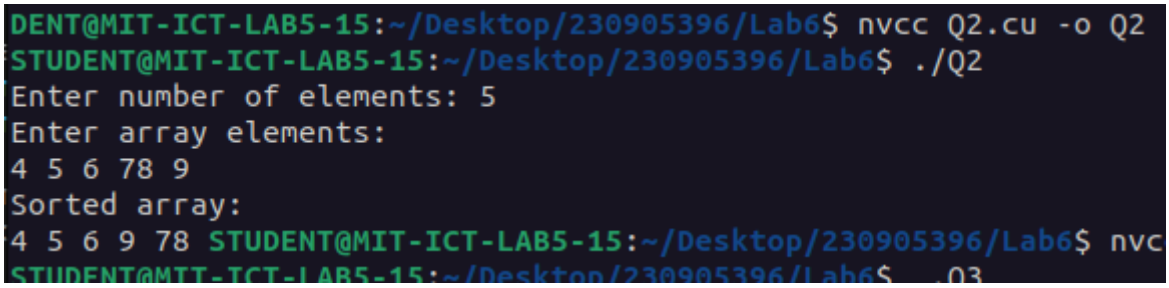


Q2)Write a program in CUDA to perform selection sort in parallel.

Code:

```
#include "cuda_runtime.h"
#include "device_launch_parameters.h"
#include <stdio.h>
#include <stdlib.h>

__global__ void selsort(int *a, int *b, int n){
    int tid = blockIdx.x*blockDim.x + threadIdx.x;
    int pos = 0;
    int data = a[tid];
    for(int i=0; i<n; i++){
      if(i == tid)continue;
      if(a[i] < data || (a[i]==data && i<tid))pos++;
    }
    b[pos]=data;
}

int main(void) {
    int n;
    int *h_a, *h_b, *d_a, *d_b;

    printf("Enter size of array");
    scanf("%d", &n);

    int size = n * sizeof(int);

    h_a = (int*)malloc(size);
    h_b = (int*)malloc(size);

    printf("Enter array elements:\n");
    for(int i = 0; i < n; i++)
      scanf("%d", &h_a[i]);;

    cudaMalloc((void **)&d_a, size);
    cudaMalloc((void **)&d_b, size);

    cudaMemcpy(d_a, h_a, size, cudaMemcpyHostToDevice);

    dim3 dimGrid(ceil(n/256.0), 1, 1);
    dim3 dimBlock(256, 1, 1);
    selsort <<< dimGrid, dimBlock >>> (d_a, d_b, n);
    cudaMemcpy(h_b, d_b, size, cudaMemcpyDeviceToHost);
    printf("Resultant sorted array is:\n");
    for(int i = 0; i < n; i++)
      printf("%d ", h_b[i]);
    printf("\n");

    cudaFree(d_a); cudaFree(d_b);
    free(h_a); free(h_b);
    return 0;
}
```

Output:



Q3)Write a program in CUDA to perform odd even transposition sort in parallel.

Code:

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

__global__ void evenPhaseKernel(int *arr, int n) {

    int tid = blockIdx.x * blockDim.x + threadIdx.x;
    int i = 2 * tid;   // even index

    if (i + 1 < n) {
        if (arr[i] > arr[i + 1]) {
            int temp = arr[i];
            arr[i] = arr[i + 1];
            arr[i + 1] = temp;
        }
    }
}

__global__ void oddPhaseKernel(int *arr, int n) {

    int tid = blockIdx.x * blockDim.x + threadIdx.x;
    int i = 2 * tid + 1;   // odd index

    if (i + 1 < n) {
        if (arr[i] > arr[i + 1]) {
            int temp = arr[i];
            arr[i] = arr[i + 1];
            arr[i + 1] = temp;
        }
    }
}

int main() {

    int n;

    printf("Enter number of elements: ");
    scanf("%d", &n);
```

```
    int *h_arr = (int*)malloc(n * sizeof(int));

    printf("Enter array elements:\n");
    for (int i = 0; i < n; i++)
        scanf("%d", &h_arr[i]);

    int *d_arr;
    cudaMalloc(&d_arr, n * sizeof(int));
    cudaMemcpy(d_arr, h_arr, n * sizeof(int), cudaMemcpyHostToDevice);

    // Number of comparison pairs ≈ n/2
    dim3 blockDim(256);
    dim3 gridDim((n/2 + blockDim.x - 1) / blockDim.x);

    // Perform n phases
    for (int i = 0; i < n; i++) {

        if (i % 2 == 0)
            evenPhaseKernel<<<gridDim, blockDim>>>(d_arr, n);
        else
            oddPhaseKernel<<<gridDim, blockDim>>>(d_arr, n);

        cudaDeviceSynchronize();
    }

    cudaMemcpy(h_arr, d_arr, n * sizeof(int), cudaMemcpyDeviceToHost);

    printf("Sorted array:\n");
    for (int i = 0; i < n; i++)
        printf("%d ", h_arr[i]);

    cudaFree(d_arr);
    free(h_arr);

    return 0;
}
```

Output: