**PP Lab 5**

**Name:** GP Anirudh

**Registration Number:** 180905452

**Section:** B

**Batch:** B2

**Roll Number:** 59

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 <cuda.h>

#include <stdlib.h>

#include <stdio.h>

// 1a, 1b, 1c

\_\_global\_\_ void vecAddKernel\_1a(float\* A, float\* B, float\* C)

{

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

C[idx] = A[idx] + B[idx];

}

\_\_global\_\_ void vecAddKernel\_1b(float\* A, float\* B, float\* C)

{

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

C[idx] = A[idx] + B[idx];

}

\_\_global\_\_ void vecAddKernel\_1c(float\* A, float\* B, float\* C, int n)

{

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

if (idx < n)

{

C[idx] = A[idx] + B[idx];

}

}

void vecAdd(float\* A, float\* B, float\* C, int n)

{

int size = n \* sizeof(float);

float\* d\_A;

float\* d\_B;

float\* d\_C;

cudaMalloc((void\*\*) &d\_A, size);

cudaMalloc((void\*\*) &d\_B, size);

cudaMalloc((void\*\*) &d\_C, size);

cudaMemcpy(d\_A, A, size, cudaMemcpyHostToDevice);

cudaMemcpy(d\_B, B, size, cudaMemcpyHostToDevice);

printf("A: ");

for (int i = 0; i < n; i++)

{

printf("%f, ", A[i]);

}

printf("\n");

printf("B: ");

for (int i = 0; i < n; i++)

{

printf("%f, ", B[i]);

}

printf("\n\n");

vecAddKernel\_1a<<<n, 1>>>(d\_A, d\_B, d\_C);

cudaMemcpy(C, d\_C, size, cudaMemcpyDeviceToHost);

printf("A+B (from 1a kernel): ");

for (int i = 0; i < n ; i++)

{

printf("%f, ", C[i]);

}

printf("\n");

vecAddKernel\_1b<<<1, n>>>(d\_A, d\_B, d\_C);

cudaMemcpy(C, d\_C, size, cudaMemcpyDeviceToHost);

printf("A+B (from 1b kernel): ");

for (int i = 0; i < n ; i++)

{

printf("%f, ", C[i]);

}

printf("\n");

vecAddKernel\_1c<<<ceil(n/256.0), n>>>(d\_A, d\_B, d\_C, n);

cudaMemcpy(C, d\_C, size, cudaMemcpyDeviceToHost);

printf("A+B (from 1c kernel): ");

for (int i = 0; i < n ; i++)

{

printf("%f, ", C[i]);

}

printf("\n");

cudaFree(d\_A);

cudaFree(d\_B);

cudaFree(d\_C);

}

int main()

{

float \*h\_A, \*h\_B, \*h\_C;

int n = 5;

int size = n \* sizeof(float);

h\_A = (float\*) malloc(size);

h\_B = (float\*) malloc(size);

h\_C = (float\*) malloc(size);

for (int i = 0; i < n; i++)

{

h\_A[i] = (i+1) \* 10;

h\_B[i] = i+1;

}

vecAdd(h\_A, h\_B, h\_C, n);

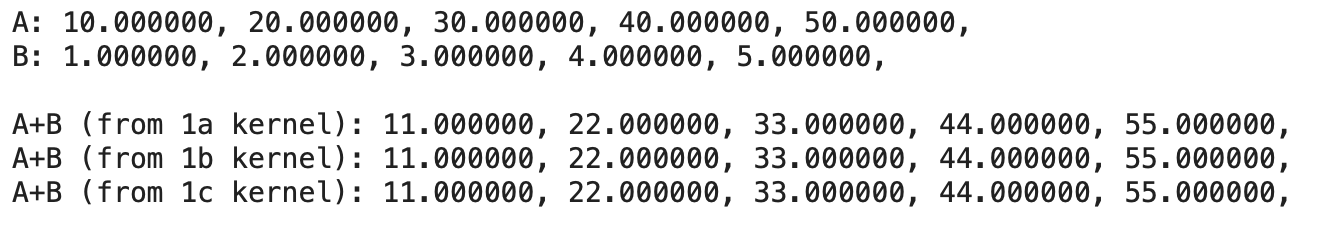
return 0;

}

**Output:**

**A picture containing text

Description automatically generated
**



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 <cuda.h>

#include <stdlib.h>

#include <stdio.h>

\_\_global\_\_ void selectionSortKernel(float\* unsorted\_arr, float\* sorted\_arr, int n)

{

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

float key = unsorted\_arr[idx];

int pos = 0;

for (int i = 0; i < n; i++)

{

if (unsorted\_arr[i] < key || (unsorted\_arr[i] == key && i < idx))

{

pos++;

}

}

sorted\_arr[pos] = key;

}

void selectionSort(float\* unsorted\_arr, float\* sorted\_arr, int n)

{

int size = n \* sizeof(float);

float\* d\_unsorted\_arr;

float\* d\_sorted\_arr;

cudaMalloc((void\*\*) &d\_unsorted\_arr, size);

cudaMalloc((void\*\*) &d\_sorted\_arr, size);

cudaMemcpy(d\_unsorted\_arr, unsorted\_arr, size, cudaMemcpyHostToDevice);

selectionSortKernel<<<1, n>>>(d\_unsorted\_arr, d\_sorted\_arr, n);

cudaMemcpy(sorted\_arr, d\_sorted\_arr, size, cudaMemcpyDeviceToHost);

cudaFree(d\_unsorted\_arr);

cudaFree(d\_sorted\_arr);

}

int main()

{

float \*h\_unsorted\_arr, \*h\_sorted\_arr;

int n = 5;

int size = n \* sizeof(float);

h\_unsorted\_arr = (float\*) malloc(size);

h\_sorted\_arr = (float\*) malloc(size);

for (int i = 0; i < 5; i++)

{

h\_unsorted\_arr[i] = rand() % 50;

}

selectionSort(h\_unsorted\_arr, h\_sorted\_arr, n);

printf("unsorted\_arr: ");

for (int i = 0; i < n; i++)

{

printf("%f, ", h\_unsorted\_arr[i]);

}

printf("\n\n");

printf("sorted\_arr: ");

for (int i = 0; i < n; i++)

{

printf("%f, ", h\_sorted\_arr[i]);

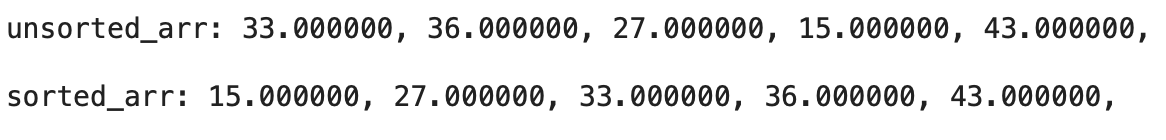
}

printf("\n");

return 0;

}

**Output:**



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 <cuda.h>

#include <stdlib.h>

#include <stdio.h>

\_\_global\_\_ void oddEven(float\* arr, int n)

{

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

if (idx % 2 == 1 && idx + 1 < n)

{

if (arr[idx] > arr[idx+1])

{

float temp = arr[idx];

arr[idx] = arr[idx+1];

arr[idx+1] = temp;

}

}

}

\_\_global\_\_ void evenOdd(float\* arr, int n)

{

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

if (idx%2 == 0 && idx+1 < n)

{

if (arr[idx] > arr[idx+1])

{

float temp = arr[idx];

arr[idx] = arr[idx+1];

arr[idx+1] = temp;

}

}

}

void oddEvenTranspositionSort(float\* arr, int n)

{

int size = n \* sizeof(float);

float\* d\_arr;

cudaMalloc((void\*\*) &d\_arr, size);

cudaMemcpy(d\_arr, arr, size, cudaMemcpyHostToDevice);

for (int i = 0; i <= n/2; i++)

{

oddEven<<<1, n>>>(d\_arr, n);

evenOdd<<<1, n>>>(d\_arr, n);

}

cudaMemcpy(arr, d\_arr, size, cudaMemcpyDeviceToHost);

cudaFree(d\_arr);

}

int main()

{

float \*h\_arr;

int n = 5;

int size = n \* sizeof(float);

h\_arr = (float\*) malloc(size);

for (int i = 0; i < 5; i++)

{

h\_arr[i] = rand() % 50;

}

printf("unsorted\_arr: ");

for (int i = 0; i < n; i++)

{

printf("%f, ", h\_arr[i]);

}

printf("\n\n");

oddEvenTranspositionSort(h\_arr, n);

printf("sorted\_arr: ");

for (int i = 0; i < n; i++)

{

printf("%f, ", h\_arr[i]);

}

printf("\n");

return 0;

}

**Output:**

