

Week 7 PCAP Lab :Array Programs in CUDA

Name: Sagnik Chatterjee

Reg: 180905478

Roll No: 61

Sec :6 'B'

Q1. 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.

Soln:

Part1

```
%%cu
#include <stdio.h>
#include <stdlib.h>

/**
Author: Sagnik Chatterjee
*/

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

void printVector(int *a, int n){
    for(int i=0;i<n;i++){
        printf("%d," ,a[i]);
    }
    printf("\n");
}
```

```

int main()
{
    int *d_a, *d_b, *d_c;
    int n = 10;
    int a[n] = {1,2,3,4,5,6,7,8,9,10};
    int b[n] = {10,9,8,7,6,5,4,3,2,1};
    int c[n];
    printVector(a, n);
    printVector(b, n);
    cudaMalloc((void **)&d_a, n * sizeof(int));
    cudaMalloc((void **)&d_b, n * sizeof(int));
    cudaMalloc((void **)&d_c, n * sizeof(int));
    cudaMemcpy(d_a, a, n * sizeof(int), cudaMemcpyHostToDevice);
    cudaMemcpy(d_b, b, n * sizeof(int), cudaMemcpyHostToDevice);
    add_vectors<<<n, 1>>>(d_a, d_b, d_c);
    cudaMemcpy(c, d_c, n * sizeof(int), cudaMemcpyDeviceToHost);
    printVector(c, n);
    cudaFree(d_a);
    cudaFree(d_b);
    cudaFree(d_c);
}

```

Output:

```

1,2,3,4,5,6,7,8,9,10,
10,9,8,7,6,5,4,3,2,1,
11,11,11,11,11,11,11,11,11,11,

```

Part2:

```

%%cu
#include <stdio.h>
#include <stdlib.h>
/**
Author : Sagnik Chatterjee
*/

```

```

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

void printVector(int *a, int n)
{
    for (int i = 0; i < n; i++)
    {
        printf("%d, ", a[i]);
    }
    printf("\n");
}

int main()
{
    int *d_a, *d_b, *d_c;
    int n = 5;
    int a[n] = {2,3,4,4,1};
    int b[n] = {5,4,3,1,2};
    int c[n];
    printVector(a, n);
    printVector(b, n);
    cudaMalloc((void **)&d_a, n * sizeof(int));
    cudaMalloc((void **)&d_b, n * sizeof(int));
    cudaMalloc((void **)&d_c, n * sizeof(int));
    cudaMemcpy(d_a, a, n * sizeof(int), cudaMemcpyHostToDevice);
    cudaMemcpy(d_b, b, n * sizeof(int), cudaMemcpyHostToDevice);
    addN<<<1, n>>>(d_a, d_b, d_c);
    cudaMemcpy(c, d_c, n * sizeof(int), cudaMemcpyDeviceToHost);
    printVector(c, n);
    cudaFree(d_a);
    cudaFree(d_b);
    cudaFree(d_c);
}

```

Output:

```
2, 3, 4, 4, 1,
5, 4, 3, 1, 2,
7, 7, 7, 5, 3,
```

Part3:

```
%%cu
#include <stdio.h>
#include <stdlib.h>

/**
Author: Sagnik Chatterjee
*/

__global__ void sum256Block(int *a, int *b, int *c, int n)
{
    int i = (blockDim.x * blockIdx.x) + threadIdx.x;
    if (i < n)
    {
        c[i] = a[i] + b[i];
    }
}

void printVector(int *a, int n)
{
    for (int i = 0; i < n; i++)
    {
        printf("%d, ", a[i]);
    }
    printf("\n");
}

int main()
{
    int *d_a, *d_b, *d_c;
    int n = 10;
```

```

int a[n] = {1,2,3,4,5,6,7,8,9,10};
int b[n] = {11,12,13,14,15,16,17,18,19,20};
int c[n];
printVector(a, n);
printVector(b, n);
cudaMalloc((void **)&d_a, n * sizeof(int));
cudaMalloc((void **)&d_b, n * sizeof(int));
cudaMalloc((void **)&d_c, n * sizeof(int));
cudaMemcpy(d_a, a, n * sizeof(int), cudaMemcpyHostToDevice);
cudaMemcpy(d_b, b, n * sizeof(int), cudaMemcpyHostToDevice);
sum256Block<<<ceil(n / 256.0), 256>>>(d_a, d_b, d_c, n);
cudaMemcpy(c, d_c, n * sizeof(int), cudaMemcpyDeviceToHost);
for(int i=0;i<n;i++){
    printf("%d, ",c[i]);
}
cudaFree(d_a);
cudaFree(d_b);
cudaFree(d_c);
}

```

Output:

```

1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
12, 14, 16, 18, 20, 22, 24, 26, 28, 30,

```

Q2 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.

Soln:

```

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

```

```

/**
Author: Sagnik Chatterjee
*/

__global__ void parallel_kernel_pos(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 sort_array(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);
    }
    parallel_kernel_pos<<<1,n>>>(d_arr,n);
    cudaMemcpy(h_arr,d_arr,size,cudaMemcpyDeviceToHost);
    cudaFree(d_arr);
}

```

```

int main(){
    int n = 10;
    int arr[n] = {33,36,27,15,43,35,36,42,49,21};
    for(int i=0;i<10;i++){
        printf("%d ",arr[i]);
    }
    //after sorting
    sort_array(arr,n);
    printf("\n");
    for(int i=0;i<10;i++){
        printf("%d ",arr[i] );
    }
    return 0;
}

```

Output:

```

33 36 27 15 43 35 36 42 49 21
15 21 27 33 35 36 36 42 43 49

```

Q3 Write and execute a CUDA program to read an integer array of size N. Sort this array using odd-even transposition sorting. Use 2 kernels.

Soln:

```

%%cu
#include <stdio.h>
#include <stdlib.h>
/**
Author: Sagnik Chatterjee
*/
__device__ void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;}
__global__ void oddEven(int *a, int n)
{

```

```

int i = (blockDim.x * blockIdx.x) + threadIdx.x;
if ((i % 2 == 1) && (i < n - 1))
{
    if (a[i] > a[i + 1])
    {
        swap(&a[i], &a[i + 1]);
    }
}
}

__global__ void evenOdd(int *a, int n)
{
    int i = (blockDim.x * blockIdx.x) + threadIdx.x;
    if ((i % 2 == 0) && (i < n - 1))
    {
        if (a[i] > a[i + 1])
        {
            swap(&a[i], &a[i + 1]);
        }
    }
}

void printVector(int *a, int n)
{
    for (int i = 0; i < n; i++)
    {
        printf("%d, ", a[i]);
    }
    printf("\n");
}

int main()
{
    int *d_a;
    int n = 10;
    int a[n] = {13, 10, 4, 7, 1, 4, 8, 0, 5, 6};
    printVector(a, n);
}

```

```
cudaMalloc((void **)&d_a, n * sizeof(int));
cudaMemcpy(d_a, a, n * sizeof(int), cudaMemcpyHostToDevice);
for (int i = 0; i <= n / 2; i++)
{
    oddEven<<<ceil(n / 256.0), 256>>>(d_a, n);
    evenOdd<<<ceil(n / 256.0), 256>>>(d_a, n);
}
cudaMemcpy(a, d_a, n * sizeof(int), cudaMemcpyDeviceToHost);
printVector(a, n);
cudaFree(d_a);
}
```

Output:

```
13, 10, 4, 7, 1, 4, 8, 0, 5, 6,
0, 1, 4, 4, 5, 6, 7, 8, 10, 13,
```