Practical No. 4

```
#include <cstdlib>
#include <iostream>

#define checkCudaErrors(call)                                              \
    do {                                                                   \
        cudaError_t err = call;                                            \
        if (err != cudaSuccess) {                                          \
            printf("CUDA error at %s %d: %s\n", __FILE__, __LINE__, cudaGetErrorString(err)); \
            exit(EXIT_FAILURE);                                            \
        }                                                                  \
    } while (0)

using namespace std;

// VectorAdd parallel function
__global__ void vectorAdd(int *a, int *b, int *result, int n) {
    int tid = threadIdx.x + blockIdx.x * blockDim.x;
    if (tid < n) {
        result[tid] = a[tid] + b[tid];
    }
}

int main() {
```

```
int *a, *b, *c;

int *a_dev, *b_dev, *c_dev;

int n = 1 << 4;


a = new int[n];

b = new int[n];

c = new int[n];

int *d = new int[n];

int size = n * sizeof(int);

checkCudaErrors(cudaMalloc(&a_dev, size));

checkCudaErrors(cudaMalloc(&b_dev, size));

checkCudaErrors(cudaMalloc(&c_dev, size));


// Array initialization..You can use Randon function to assign values

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

   a[i] = rand() % 1000;

   b[i] = rand() % 1000;

   d[i] = a[i] + b[i];  // calculating serial addition

}

cout << "Given array A is =>\n";

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

   cout << a[i] << ", ";

}

cout << "\n\n";
cout << "Given array B is =>\n";
for (int i = 0; i < n; i++) {
    cout << b[i] << ", ";
}
cout << "\n\n";

cudaEvent_t start, end;

checkCudaErrors(cudaEventCreate(&start));
checkCudaErrors(cudaEventCreate(&end));

checkCudaErrors(cudaMemcpy(a_dev, a, size, cudaMemcpyHostToDevice));
checkCudaErrors(cudaMemcpy(b_dev, b, size, cudaMemcpyHostToDevice));
int threads = 1024;
int blocks = (n + threads - 1) / threads;
checkCudaErrors(cudaEventRecord(start));

// Parallel addition program
vectorAdd<<<blocks, threads>>>(a_dev, b_dev, c_dev, n);

checkCudaErrors(cudaEventRecord(end));
checkCudaErrors(cudaEventSynchronize(end));

float time = 0.0;
checkCudaErrors(cudaEventElapsedTime(&time, start, end));
checkCudaErrors(cudaMemcpy(c, c_dev, size, cudaMemcpyDeviceToHost));


// Calculate the error term.


cout << "CPU sum is =>\n";
for (int i = 0; i < n; i++) {
    cout << d[i] << ", ";
}
cout << "\n\n";


cout << "GPU sum is =>\n";
for (int i = 0; i < n; i++) {
    cout << c[i] << ", ";
}
cout << "\n\n";


int error = 0;
for (int i = 0; i < n; i++) {
    error += d[i] - c[i];
    if (0 != (d[i] - c[i])) {
        cout << "Error at (" << i << ") => GPU: " << c[i] << ", CPU: " << d[i] << "\n";
    }
}
```

```
    cout << "\nError : " << error;
    cout << "\nTime Elapsed: " << time;

    return 0;
}

/*
OUTPUT:

Given array A is =>
383, 777, 793, 386, 649, 362, 690, 763, 540, 172, 211, 567, 782, 862, 67, 929,

Given array B is =>
886, 915, 335, 492, 421, 27, 59, 926, 426, 736, 368, 429, 530, 123, 135, 802,

CPU sum is =>
1269, 1692, 1128, 878, 1070, 389, 749, 1689, 966, 908, 579, 996, 1312, 985, 202, 1731,

GPU sum is =>
1269, 1692, 1128, 878, 1070, 389, 749, 1689, 966, 908, 579, 996, 1312, 985, 202, 1731,

Error : 0
Time Elapsed:  0.017408

*/
```