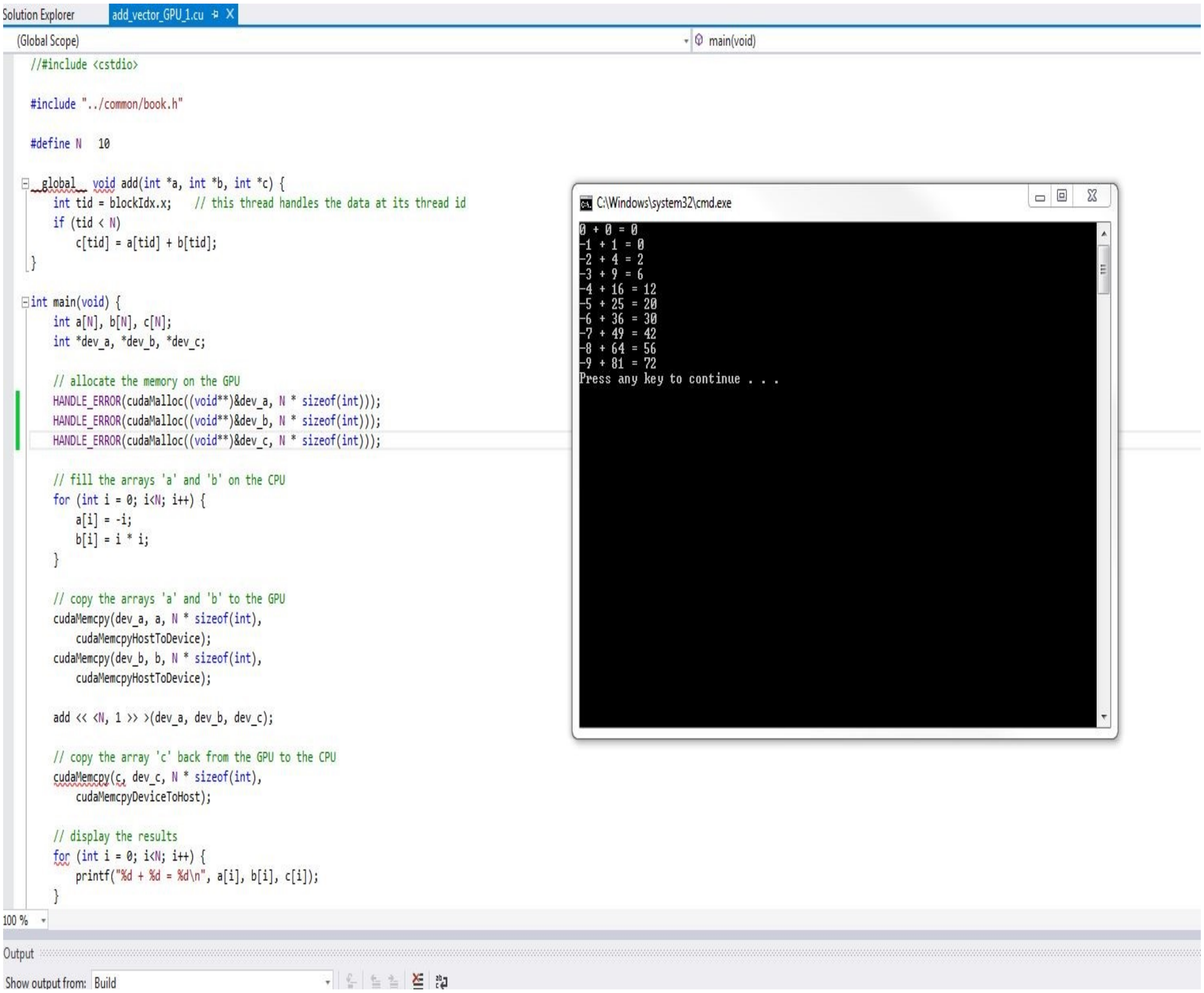
CSC8014 Topics: GPU Programming

Student: Lina Mi @01377283

Homework #1

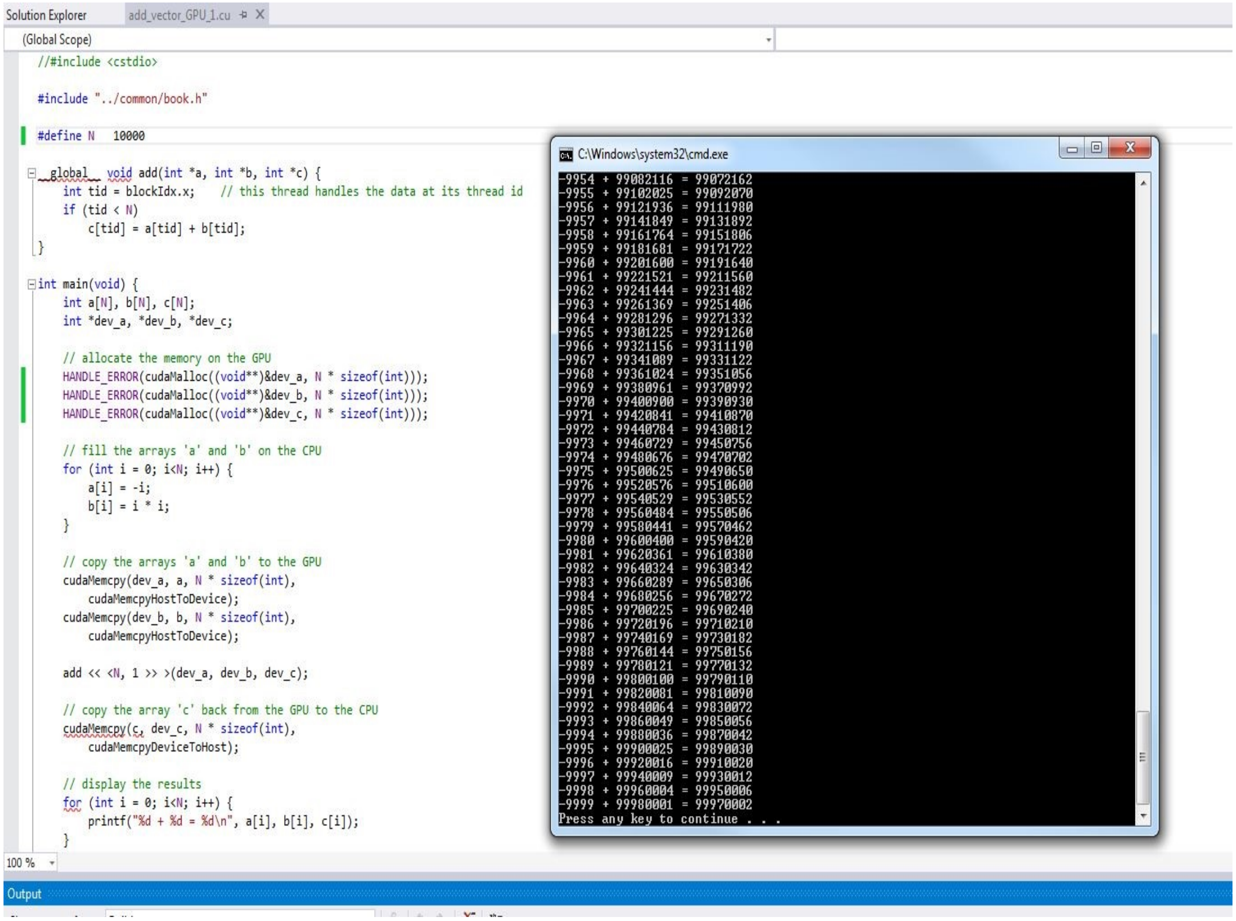
1. Compile and run the code of adding two vectors **a** and **b** in Chapter 4.
    a. What is the result? You can take a screenshot of the result.



```
//#include <cstdio>

#include "../common/book.h"

#define N   10

__global__ void add(int *a, int *b, int *c) {
    int tid = blockIdx.x;    // this thread handles the data at its thread id
    if (tid < N)
        c[tid] = a[tid] + b[tid];
}

int main(void) {
    int a[N], b[N], c[N];
    int *dev_a, *dev_b, *dev_c;

    // allocate the memory on the GPU
    HANDLE_ERROR(cudaMalloc((void**)&dev_a, N * sizeof(int)));
    HANDLE_ERROR(cudaMalloc((void**)&dev_b, N * sizeof(int)));
    HANDLE_ERROR(cudaMalloc((void**)&dev_c, N * sizeof(int)));

    // fill the arrays 'a' and 'b' on the CPU
    for (int i = 0; i<N; i++) {
        a[i] = -i;
        b[i] = i * i;
    }

    // copy the arrays 'a' and 'b' to the GPU
    cudaMemcpy(dev_a, a, N * sizeof(int),
        cudaMemcpyHostToDevice);
    cudaMemcpy(dev_b, b, N * sizeof(int),
        cudaMemcpyHostToDevice);

    add << <N, 1 >> >(dev_a, dev_b, dev_c);

    // copy the array 'c' back from the GPU to the CPU
    cudaMemcpy(c, dev_c, N * sizeof(int),
        cudaMemcpyDeviceToHost);

    // display the results
    for (int i = 0; i<N; i++) {
        printf("%d + %d = %d\n", a[i], b[i], c[i]);
    }
```

```
0 + 0 = 0
-1 + 1 = 0
-2 + 4 = 2
-3 + 9 = 6
-4 + 16 = 12
-5 + 25 = 20
-6 + 36 = 30
-7 + 49 = 42
-8 + 64 = 56
-9 + 81 = 72
Press any key to continue . . .
```

b. Change the number N from 10 to 10000. Compile and run the code again. What is the result?

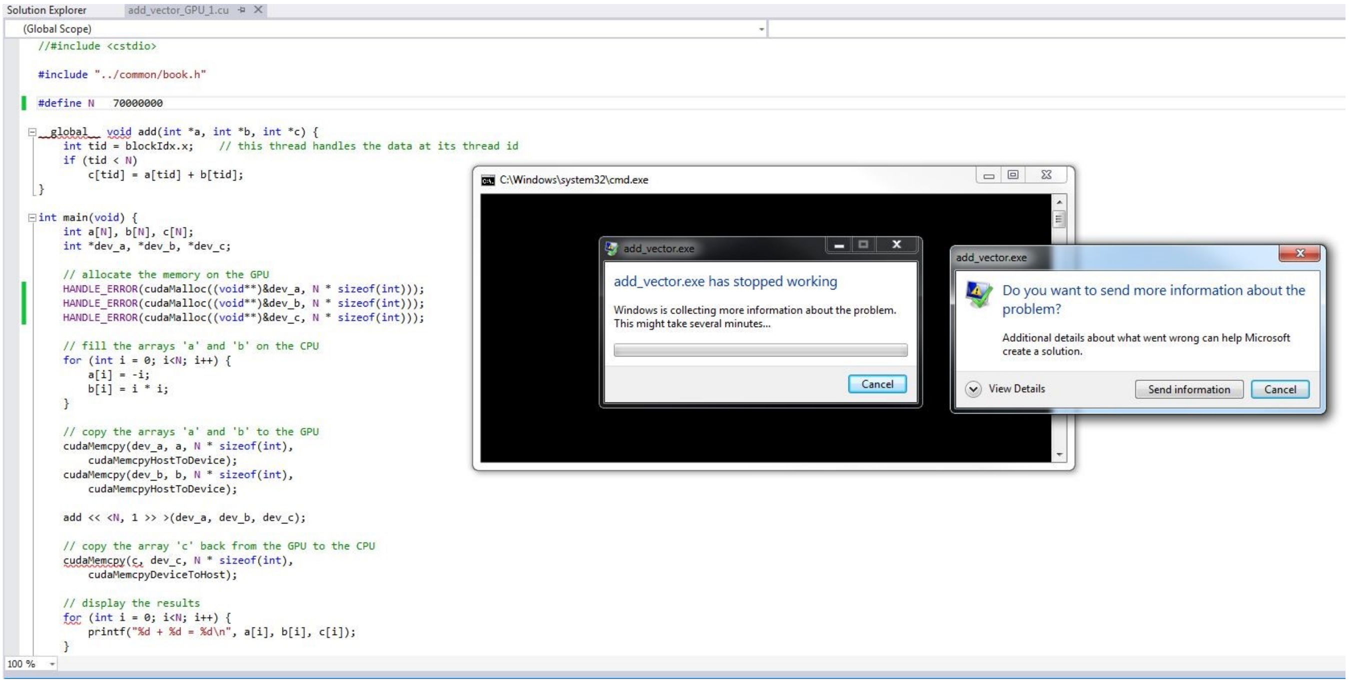


```
//#include <cstdio>

#include "../common/book.h"

#define N   10000

__global__ void add(int *a, int *b, int *c) {
    int tid = blockIdx.x;    // this thread handles the data at its thread id
    if (tid < N)
        c[tid] = a[tid] + b[tid];
}

int main(void) {
    int a[N], b[N], c[N];
    int *dev_a, *dev_b, *dev_c;

    // allocate the memory on the GPU
    HANDLE_ERROR(cudaMalloc((void**)&dev_a, N * sizeof(int)));
    HANDLE_ERROR(cudaMalloc((void**)&dev_b, N * sizeof(int)));
    HANDLE_ERROR(cudaMalloc((void**)&dev_c, N * sizeof(int)));

    // fill the arrays 'a' and 'b' on the CPU
    for (int i = 0; i<N; i++) {
        a[i] = -i;
        b[i] = i * i;
    }

    // copy the arrays 'a' and 'b' to the GPU
    cudaMemcpy(dev_a, a, N * sizeof(int),
        cudaMemcpyHostToDevice);
    cudaMemcpy(dev_b, b, N * sizeof(int),
        cudaMemcpyHostToDevice);

    add << <N, 1 >> >(dev_a, dev_b, dev_c);

    // copy the array 'c' back from the GPU to the CPU
    cudaMemcpy(c, dev_c, N * sizeof(int),
        cudaMemcpyDeviceToHost);

    // display the results
    for (int i = 0; i<N; i++) {
        printf("%d + %d = %d\n", a[i], b[i], c[i]);
    }
```

C:\Windows\system32\cmd.exe

```
-9954 + 99082116 = 99072162
-9955 + 99102025 = 99092070
-9956 + 99121936 = 99111980
-9957 + 99141849 = 99131892
-9958 + 99161764 = 99151806
-9959 + 99181681 = 99171722
-9960 + 99201600 = 99191640
-9961 + 99221521 = 99211560
-9962 + 99241444 = 99231482
-9963 + 99261369 = 99251406
-9964 + 99281296 = 99271332
-9965 + 99301225 = 99291260
-9966 + 99321156 = 99311190
-9967 + 99341089 = 99331122
-9968 + 99361024 = 99351056
-9969 + 99380961 = 99370992
-9970 + 99400900 = 99390930
-9971 + 99420841 = 99410870
-9972 + 99440784 = 99430812
-9973 + 99460729 = 99450756
-9974 + 99480676 = 99470702
-9975 + 99500625 = 99490650
-9976 + 99520576 = 99510600
-9977 + 99540529 = 99530552
-9978 + 99560484 = 99550506
-9979 + 99580441 = 99570462
-9980 + 99600400 = 99590420
-9981 + 99620361 = 99610380
-9982 + 99640324 = 99630342
-9983 + 99660289 = 99650306
-9984 + 99680256 = 99670272
-9985 + 99700225 = 99690240
-9986 + 99720196 = 99710210
-9987 + 99740169 = 99730182
-9988 + 99760144 = 99750156
-9989 + 99780121 = 99770132
-9990 + 99800100 = 99790110
-9991 + 99820081 = 99810090
-9992 + 99840064 = 99830072
-9993 + 99860049 = 99850056
-9994 + 99880036 = 99870042
-9995 + 99900025 = 99890030
-9996 + 99920016 = 99910020
-9997 + 99940009 = 99930012
-9998 + 99960004 = 99950006
-9999 + 99980001 = 99970002
Press any key to continue . . .
```

c. Change the number N to 70000000. Then compile and run the code one more time. Does the code still execute? Explain what happens.



```
//#include <cstdio>

#include "../common/book.h"

#define N   70000000

__global__ void add(int *a, int *b, int *c) {
    int tid = blockIdx.x;    // this thread handles the data at its thread id
    if (tid < N)
        c[tid] = a[tid] + b[tid];
}

int main(void) {
    int a[N], b[N], c[N];
    int *dev_a, *dev_b, *dev_c;

    // allocate the memory on the GPU
    HANDLE_ERROR(cudaMalloc((void**)&dev_a, N * sizeof(int)));
    HANDLE_ERROR(cudaMalloc((void**)&dev_b, N * sizeof(int)));
    HANDLE_ERROR(cudaMalloc((void**)&dev_c, N * sizeof(int)));

    // fill the arrays 'a' and 'b' on the CPU
    for (int i = 0; i<N; i++) {
        a[i] = -i;
        b[i] = i * i;
    }

    // copy the arrays 'a' and 'b' to the GPU
    cudaMemcpy(dev_a, a, N * sizeof(int),
        cudaMemcpyHostToDevice);
    cudaMemcpy(dev_b, b, N * sizeof(int),
        cudaMemcpyHostToDevice);

    add << <N, 1 >> >(dev_a, dev_b, dev_c);

    // copy the array 'c' back from the GPU to the CPU
    cudaMemcpy(c, dev_c, N * sizeof(int),
        cudaMemcpyDeviceToHost);

    // display the results
    for (int i = 0; i<N; i++) {
        printf("%d + %d = %d\n", a[i], b[i], c[i]);
    }
```

add_vector.exe has stopped working

Windows is collecting more information about the problem. This might take several minutes...

Do you want to send more information about the problem?

Additional details about what went wrong can help Microsoft create a solution.

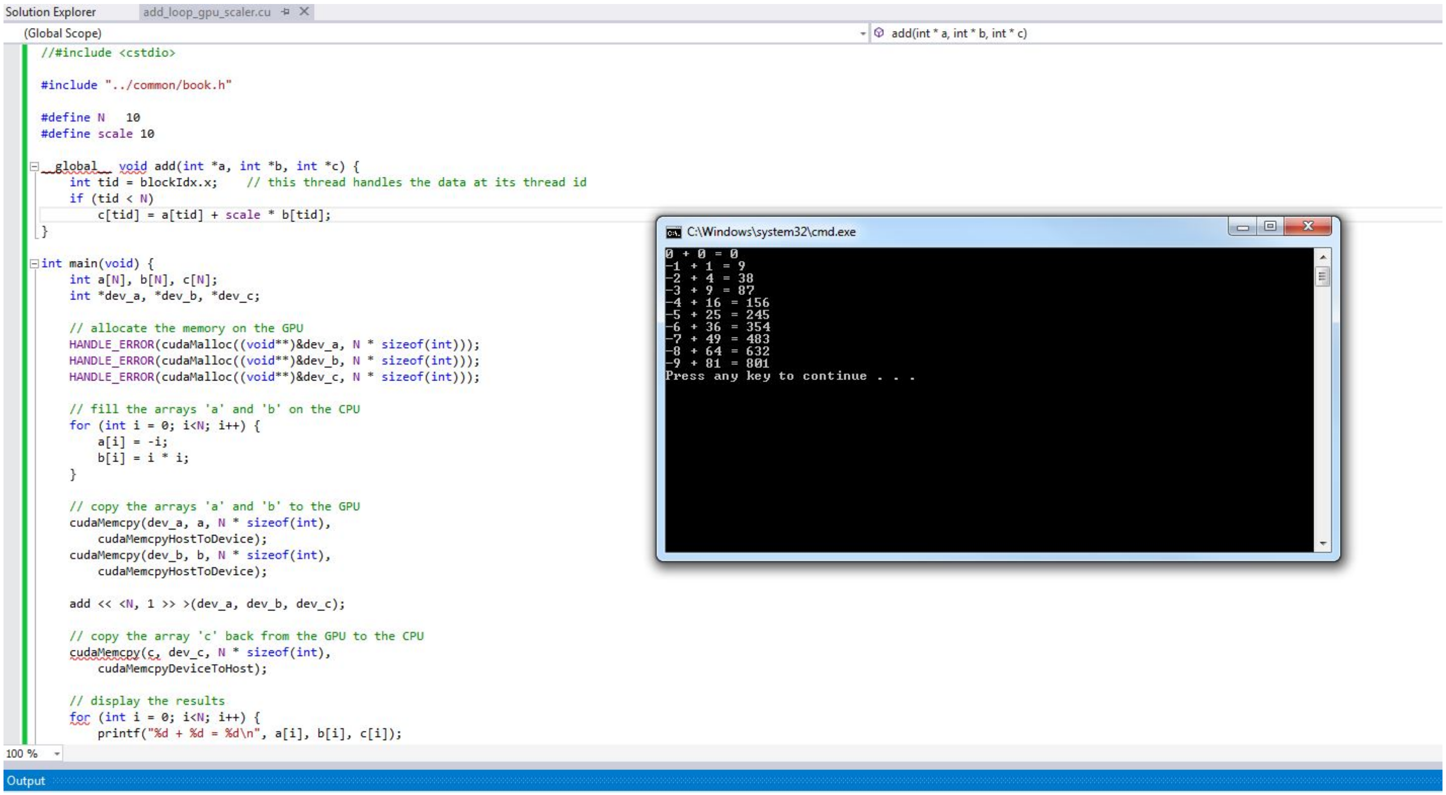
The code stopped executing, because the number of 70000000 is very large and square of it is too large to be hold with a *int* type variable (32bit int type variable can hold maximum value is 2^32=4,294,967,296<70000000^2)

2. Change the code (Also attached as add_loop_gpu.zip) in Chapter 4 to perform the following operation for vectors **a** and **b**:

   c[i] = a[i] + scale * b[i]

   where scale is an integer.

   a. Compile and run the code with N = 10 and scale = 10.



```
//#include <cstdio>

#include "../common/book.h"

#define N   10
#define scale 10

__global__ void add(int *a, int *b, int *c) {
    int tid = blockIdx.x;    // this thread handles the data at its thread id
    if (tid < N)
        c[tid] = a[tid] + scale * b[tid];
}

int main(void) {
    int a[N], b[N], c[N];
    int *dev_a, *dev_b, *dev_c;

    // allocate the memory on the GPU
    HANDLE_ERROR(cudaMalloc((void**)&dev_a, N * sizeof(int)));
    HANDLE_ERROR(cudaMalloc((void**)&dev_b, N * sizeof(int)));
    HANDLE_ERROR(cudaMalloc((void**)&dev_c, N * sizeof(int)));

    // fill the arrays 'a' and 'b' on the CPU
    for (int i = 0; i<N; i++) {
        a[i] = -i;
        b[i] = i * i;
    }

    // copy the arrays 'a' and 'b' to the GPU
    cudaMemcpy(dev_a, a, N * sizeof(int),
        cudaMemcpyHostToDevice);
    cudaMemcpy(dev_b, b, N * sizeof(int),
        cudaMemcpyHostToDevice);

    add << <N, 1 >> >(dev_a, dev_b, dev_c);

    // copy the array 'c' back from the GPU to the CPU
    cudaMemcpy(c, dev_c, N * sizeof(int),
        cudaMemcpyDeviceToHost);

    // display the results
    for (int i = 0; i<N; i++) {
        printf("%d + %d = %d\n", a[i], b[i], c[i]);
```

```
0 + 0 = 0
-1 + 1 = 9
-2 + 4 = 38
-3 + 9 = 87
-4 + 16 = 156
-5 + 25 = 245
-6 + 36 = 354
-7 + 49 = 483
-8 + 64 = 632
-9 + 81 = 801
Press any key to continue . . .
```

Please submit the source code in blackboard.