Practice

Exercise 1

- Function that solve a system of linear equations in the host.
- Function that solves a system of linear equations in the device through the launch of a kernel (1 block and 1 thread).

The kernel must receive all coefficients as a vector (of size 6).

A linear system with the form:

> ax + by = c dx + ey = f,

Can be solved by the formulas:

> x = (ce - bf) / (ae -bd) y = (af - cd) / (ae - bd)

```
#include "cuda_runtime.h"
#include "device_launch_parameters.h"

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

__host__ void linearSolveCPU(float* n, float* x, float* y) {
    *x = (n[2] * n[4] - n[1] * n[5]) / (n[0] * n[4] - n[1] * n
        [3]);
    *y = (n[0] * n[5] - n[2] * n[3]) / (n[0] * n[4] - n[1] * n
        [3]);
}

__global__ void linearSolveGPU(float* n, float* x, float* y)
{
    *x = (n[2] * n[4] - n[1] * n[5]) / (n[0] * n[4] - n[1] * n
        [3]);
    *y = (n[0] * n[5] - n[2] * n[3]) / (n[0] * n[4] - n[1] * n
        [3]);
}

int main()
{
    float* n_host = (float*)malloc(sizeof(float) * 6); // if
        malloc, you need to initialize all spaces one by one
    float* x_host = (float*)malloc(sizeof(float));
    float* y_host = (float*)malloc(sizeof(float));

    float* x_gpu = (float*)malloc(sizeof(float));
    float* y_gpu = (float*)malloc(sizeof(float));

```

```
27      float* n_device;
28      float* x_device;
29      float* y_device;
30
31      cudaMalloc((void**)&n_device, sizeof(float) * 6);
32      cudaMalloc((void**)&x_device, sizeof(float));
33      cudaMalloc((void**)&y_device, sizeof(float));
34
35      n_host[0] = 5;
36      n_host[1] = 1;
37      n_host[2] = 4;
38      n_host[3] = 2;
39      n_host[4] = -3;
40      n_host[5] = 5;
41
42      *x_host = 0;
43      *y_host = 0;
44      *x_gpu = 0;
45      *y_gpu = 0;
46
47      cudaMemcpy(n_device, n_host, sizeof(float) * 6,
           cudaMemcpyHostToDevice);
48      cudaMemcpy(x_device, x_host, sizeof(float),
           cudaMemcpyHostToDevice);
49      cudaMemcpy(y_device, y_host, sizeof(float),
           cudaMemcpyHostToDevice);
50
51      linearSolveCPU(n_host, x_host, y_host);
52      printf("CPU result \n");
53      printf("x = %f y = %f \n", *x_host, *y_host);
54
55      linearSolveGPU <<< 1, 1 >>> (n_device, x_device, y_device)
           ;
56      cudaMemcpy(x_gpu, x_device, sizeof(float),
           cudaMemcpyDeviceToHost);
57      cudaMemcpy(y_gpu, y_device, sizeof(float),
           cudaMemcpyDeviceToHost);
58      printf("GPU result \n");
59      printf("x = %f y = %f \n", *x_gpu, *y_gpu);
60
61      free(n_host);
62      free(x_host);
63      free(y_host);
64      free(x_gpu);
65      free(y_gpu);
66
67      cudaFree(n_device);
68      cudaFree(x_device);
69      cudaFree(y_device);
70
71      return 0;
```

```
72 }
```

Lab 03

Make a program in c/c++ in which you launch a kernel with one block and one thread. The kernel must solve a quadratic equation in the form:

> ax^2 + bx + c = 0,

where its solutions are given by:

> x1 = (-b + sqrt(b^2 - 4ac)) / 2a x2 = (-b - sqrt(b^2 - 4ac)) / 2a

For the implementation, you must consider:

1. Ask the user for coefficients a, b and c.
2. The program must show the solutions for the equation or a message stating that the solution does NOT exist if the result is an imaginary number.

Tests

- a = 1, b = -5, c = 6 -> x1 = 2, x2 = 3
- a = 1, b = 1, c = 1 -> The solution does not exist

Solution

```
1  #include "cuda_runtime.h"
2  #include "device_launch_parameters.h"
3
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <math.h>
7
8  __global__ void solveGPU(double* dev_abc, double* dev_x1x2,
       bool* dev_error)
9  {
10     double root = (dev_abc[1] * dev_abc[1]) - (4 * dev_abc[0]
           * dev_abc[2]);
11     // printf("root: %lf\n", root);
12     if (root < 0) {
13         *dev_error = true;
14     }
15     else {
16         *dev_error = false;
```

```
17          dev_x1x2[0] = ((-1 * dev_abc[1] - sqrt(root)) / (2 *
               dev_abc[0]));
18          dev_x1x2[1] = ((-1 * dev_abc[1] + sqrt(root)) / (2 *
               dev_abc[0]));
19      }
20
21  }
22
23  int main() {
24      double* n_host = (double*)malloc(sizeof(double) * 3); //
           not cast, error
25      double* x1x2_host = (double*)malloc(sizeof(double) * 2);
26      bool* error_host = (bool*)malloc(sizeof(bool));
27
28      double* n_dev;
29      double* x1x2_dev;
30      bool* error_dev;
31      cudaMalloc((void**)&n_dev, sizeof(double) * 3);
32      cudaMalloc((void**)&x1x2_dev, sizeof(double) * 2);
33      cudaMalloc((void**)&error_dev, sizeof(bool)); // &bool
           error
34
35      for (int i = 0; i < 3; i++) {
36          printf("%c: ", char(i + 97)); //printf("%s", (i + 65))
               ; exception
37          scanf("%lf", &n_host[i]); // "A:%lf" not error, but
               input incomplete // \n weird results
38      }
39
40      x1x2_host[0] = 0;
41      x1x2_host[1] = 0;
42      *error_host = false;
43
44      cudaMemcpy(n_dev, n_host, sizeof(double) * 3,
           cudaMemcpyHostToDevice);
45      cudaMemcpy(x1x2_dev, x1x2_host, sizeof(double) * 2,
           cudaMemcpyHostToDevice); // not necessary
46      cudaMemcpy(error_dev, error_host, sizeof(bool),
           cudaMemcpyHostToDevice); // not necessary
47
48      solveGPU << < 1, 1 >> > (n_dev, x1x2_dev, error_dev);
49
50      // cout << "cuda ptr " << *error_dev << endl; // no error,
           but execption at runtime
51      cudaMemcpy(error_host, error_dev, sizeof(bool),
           cudaMemcpyDeviceToHost);
52      cudaMemcpy(x1x2_host, x1x2_dev, sizeof(double) * 2,
           cudaMemcpyDeviceToHost);
53      if (*error_host) {
54          printf("GPU Result:\n");
55          printf("The solution does not exist\n");
```

```
56      }
57      else {
58          printf("GPU Result:\n");
59          printf("x1 = %lf x2 = %lf\n", x1x2_host[0], x1x2_host
                [1]);
60      }
61
62  }
```

Other Findings

```
1   #include "cuda_runtime.h"
2   #include "device_launch_parameters.h"
3
4   #include <stdio.h>
5   #include <stdlib.h>
6   #include <math.h>
7   #include<iostream>
8
9   using namespace std;
10
11  __global__ void solveGPU(double* dev_abc, double* dev_x1x2,
        int* dev_error)
12  {
13      double root = (dev_abc[1] * dev_abc[1]) - (4 * dev_abc[0]
            * dev_abc[2]);
14      if (root < 0) {
15          *dev_error = true;
16      }
17      else {
18          *dev_error = false;
19          dev_x1x2[0] = ((-1 * dev_abc[1] - sqrt(root)) / (2 *
                dev_abc[0]));
20          dev_x1x2[1] = ((-1 * dev_abc[1] + sqrt(root)) / (2 *
                dev_abc[0]));
21      }
22
23  }
24
25  int main() {
26      double n_host[3] = { 0 };
27      double x1x2_host[2] = { 0 };
28      bool error_host = false;
29
30      double* n_dev;
31      double* x1x2_dev;
32      int* error_dev; // gives no error
33      cudaMalloc((void**)&n_dev, sizeof(double) * 3);
34      cudaMalloc((void**)&x1x2_dev, sizeof(double) * 2);
```

```
35      cudaMalloc((void**)&error_dev, sizeof(bool));
36
37      for (int i = 0; i < 3; i++) {
38          printf("%c: ", char(i + 97));
39          scanf("%lf", &n_host[i]);
40      }
41
42      cudaMemcpy(n_dev, n_host, sizeof(double) * 3,
           cudaMemcpyHostToDevice);
43      cudaMemcpy(x1x2_dev, x1x2_host, sizeof(double) * 2,
           cudaMemcpyHostToDevice); // not necessary
44      cudaMemcpy(error_dev, &error_host, sizeof(bool),
           cudaMemcpyHostToDevice); // not necessary
45
46      solveGPU << < 1, 1 >> > (n_dev, x1x2_dev, error_dev);
47
48
49      cudaMemcpy(&error_host, error_dev, sizeof(bool),
           cudaMemcpyDeviceToHost);
50      cudaMemcpy(x1x2_host, x1x2_dev, sizeof(double) * 2,
           cudaMemcpyDeviceToHost);
51      if (error_host) {
52          printf("GPU Result:\n");
53          printf("The solution does not exist\n");
54      }
55      else {
56          printf("GPU Result:\n");
57          printf("x1 = %lf x2 = %lf\n", x1x2_host[0], x1x2_host
               [1]);
58      }
59
60      //free(n_host); // exc
61      //free(x1x2_host); // exc
62      //free(&error_host); // exc
63
64      cudaFree(n_dev);
65      cudaFree(x1x2_dev);
66      cudaFree(error_dev);
67  }
```

```
1  int* test;
2  cudaMalloc((void**)&test, sizeof(bool)); // no error
```

```
1  #include "cuda_runtime.h"
2  #include "device_launch_parameters.h"
3
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <math.h>
7  #include<iostream>
8
```

```
9  using namespace std;
10
11 __global__ void solveGPU(double* dev_abc, double* dev_x1x2,
       int* dev_error)
12 {
13     double root = (dev_abc[1] * dev_abc[1]) - (4 * dev_abc[0]
           * dev_abc[2]);
14     if (root < 0) {
15         *dev_error = true;
16     }
17     else {
18         *dev_error = false;
19         dev_x1x2[0] = ((-1 * dev_abc[1] - sqrt(root)) / (2 *
               dev_abc[0]));
20         dev_x1x2[1] = ((-1 * dev_abc[1] + sqrt(root)) / (2 *
               dev_abc[0]));
21     }
22
23 }
24
25 int main() {
26     double n_host[3] = { 0 };
27     double x1x2_host[2] = { 0 };
28     bool error_host = false;
29
30     double* n_dev;
31     double* x1x2_dev;
32     int* error_dev; // gives no error
33     cudaMalloc((void**)&n_dev, sizeof(double) * 3);
34     cudaMalloc((void**)&x1x2_dev, sizeof(double) * 2);
35     cudaMalloc((void**)&error_dev, sizeof(bool));
36
37     for (int i = 0; i < 3; i++) {
38         printf("%c: ", char(i + 97));
39         scanf("%lf", &n_host[i]);
40     }
41
42     cudaMemcpy(n_dev, n_host, sizeof(double) * 3,
           cudaMemcpyHostToDevice);
43     cudaMemcpy(x1x2_dev, x1x2_host, sizeof(double) * 2,
           cudaMemcpyHostToDevice); // not necessary
44     cudaMemcpy(error_dev, &error_host, sizeof(bool),
           cudaMemcpyHostToDevice); // not necessary
45
46     solveGPU << < 1, 1 >> > (n_dev, x1x2_dev, error_dev);
47
48
49     cudaMemcpy(&error_host, error_dev, sizeof(bool),
           cudaMemcpyDeviceToHost);
50     cudaMemcpy(x1x2_host, x1x2_dev, sizeof(double) * 2,
           cudaMemcpyDeviceToHost);
```

```
51      if (error_host) {
52          printf("GPU Result:\n");
53          printf("The solution does not exist\n");
54      }
55      else {
56          printf("GPU Result:\n");
57          printf("x1 = %lf x2 = %lf\n", x1x2_host[0], x1x2_host
                [1]);
58      }
59  }
```

- `cudaMalloc(void** devPtr, size_t size)`: Allocates `size` bytes of linear memory on the device and returns in *devPtr a pointer to the allocated memory. Memory not cleared.
- `cudaMemcpy (void* dst, const void* src, size_t count, cudaMemcpyKind kind)`: Copies `count` bytes from the memory area pointed to by src to the memory area pointed to by dst. Calling cudaMemcpy() with dst and src pointers that do not match the direction of the copy results in an undefined behavior.