UNIVERSIDAD PANAMERICANA

Universidad Panamericana

Campus Guadalajara

Escuela de ingenierías

Fundamentos de programación en paralelo

REPORTE DE PRÁCTICA

IDENTIFICACIÓN DE LA PRÁCTICA

Práctica	**3**	**Nombre de la práctica**	**Lanzamiento de un kernel**
Fecha	**25/08/2021**	**Nombre del profesor**	**Alma Nayeli Rodríguez Vázquez**
Nombre del estudiante		**Mariana Ávalos Arce**	

OBJETIVO

El objetivo de esta práctica consiste en implementar un kernel y mandarlo llamar utilizando un bloque y un hilo.

PROCEDIMIENTO

Realiza la implementación siguiendo estas instrucciones.

Realiza un programa en C/C++ utilizando CUDA en el que lances un kernel con un solo bloque y un solo hilo. El kernel deberá de resolver una ecuación de segundo grado de la forma:

$$a \cdot x^2 + b \cdot x + c = 0$$

Cuyas soluciones son:

$$x_1 = -\frac{b}{2a} + \frac{\sqrt{b^2 - 4ac}}{2a}$$

$$x_2 = -\frac{b}{2a} - \frac{\sqrt{b^2 - 4ac}}{2a}$$

Para la implementación, deberás considerar los siguientes requisitos:

1. Pedir los coeficientes de la ecuación al usuario.
2. El programa deberá mostrar las soluciones de la ecuación desde main. En caso de que no exista solución, se seberá mostrar un mensaje diciendo que la solución no existe.
3. Las soluciones no deben ser impresas desde el kernel.
4. El kernel deberá estar definido de la siguiente forma:

```
__global__ void formula_general(double* dev_abc, double* dev_x1x2, bool* dev_error)
```

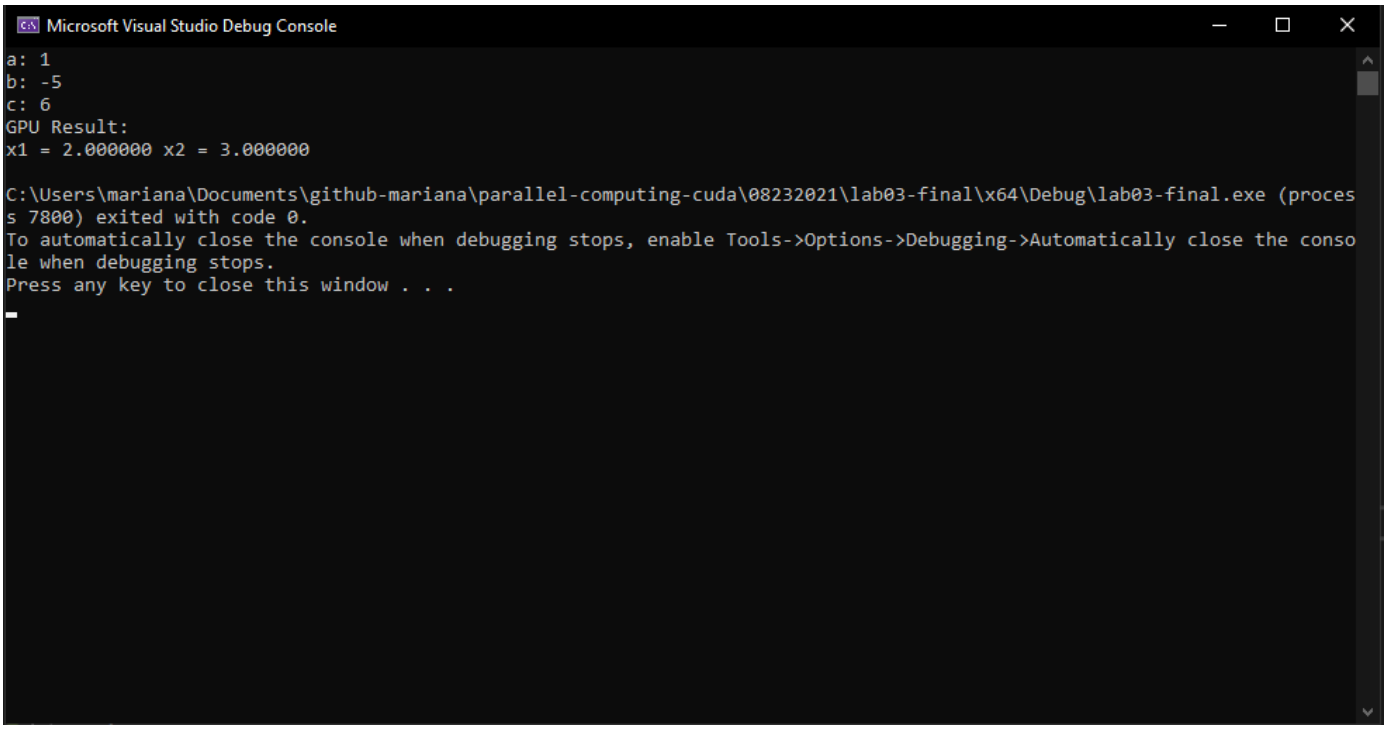
IMPLEMENTACIÓN

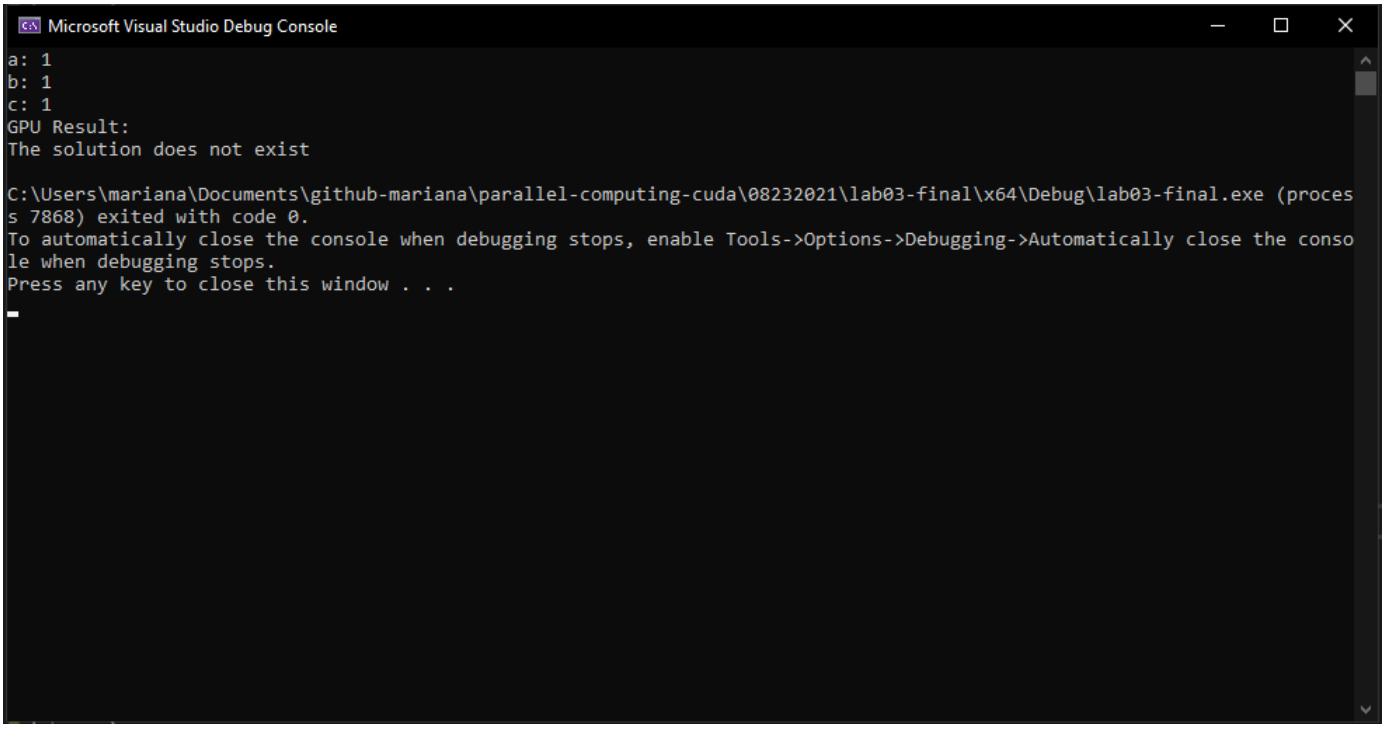
Agrega el código de tu implementación aquí.

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

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

__global__ void formula_general(double* dev_abc, double* dev_x1x2, bool* dev_error)
{
    double root = (dev_abc[1] * dev_abc[1]) - (4 * dev_abc[0] * dev_abc[2]);
    if (root < 0) {
        *dev_error = true;
    }
    else {
        *dev_error = false;
        dev_x1x2[0] = ((-1 * dev_abc[1] - sqrt(root)) / (2 * dev_abc[0]));
        dev_x1x2[1] = ((-1 * dev_abc[1] + sqrt(root)) / (2 * dev_abc[0]));
    }
}

int main() {
    double* n_host = (double*)malloc(sizeof(double) * 3);
    double* x1x2_host = (double*)malloc(sizeof(double) * 2);
    bool* error_host = (bool*)malloc(sizeof(bool));

    double* n_dev;
    double* x1x2_dev;
    bool* error_dev;
    cudaMalloc((void**)&n_dev, sizeof(double) * 3);
    cudaMalloc((void**)&x1x2_dev, sizeof(double) * 2);
    cudaMalloc((void**)&error_dev, sizeof(bool));

    for (int i = 0; i < 3; i++) {
        printf("%c: ", char(i + 97));
        scanf("%lf", &n_host[i]);
    }

    x1x2_host[0] = 0;
    x1x2_host[1] = 0;
    *error_host = false;

    cudaMemcpy(n_dev, n_host, sizeof(double) * 3, cudaMemcpyHostToDevice);
    cudaMemcpy(x1x2_dev, x1x2_host, sizeof(double) * 2, cudaMemcpyHostToDevice);
    cudaMemcpy(error_dev, error_host, sizeof(bool), cudaMemcpyHostToDevice);

    formula_general <<< 1, 1 >>> (n_dev, x1x2_dev, error_dev);

    cudaMemcpy(error_host, error_dev, sizeof(bool), cudaMemcpyDeviceToHost);
    cudaMemcpy(x1x2_host, x1x2_dev, sizeof(double) * 2, cudaMemcpyDeviceToHost);
    if (*error_host) {
        printf("GPU Result:\n");
        printf("The solution does not exist\n");
    }
    else {
        printf("GPU Result:\n");
        printf("x1 = %lf x2 = %lf\n", x1x2_host[0], x1x2_host[1]);
    }
}
```

RESULTADOS

Agrega la imagen de la consola con el despliegue de los resultados obtenidos.





CONCLUSIONES

Escribe tus observaciones y conclusiones.
Esta práctica me gustó bastante, ya que fue una especie de cierre al proceso de aprendizaje de lanzar un kernel. Lo realicé de forma más fluida y sabiendo para qué y cómo manipular la memoria. Se concluye además que se puede utilizar punteros tipo bool, cosa que en lo personal tenía la duda, ya que en el lenguaje c no está definido el tipo, pero en el lenguaje c++ sí lo está, y al tratarse de CUDA con soporte para c/c++, era una duda latente. Aprendí bastante sobre todo de espacios de memoria, ya que me ocurrió un error donde yo quería imprimir el resultado (tipo double) como si fuera un flotante, y se imprimía algo extraño, debido a que la computadora lee por bytes y un double necesita más memoria al leer para imprimirla que un float, por lo que tardé en saber que el error era en la impresión y no en la lógica del programa.