UNIVERSIDAD PANAMERICANA

Universidad Panamericana

Campus Guadalajara

Escuela de ingenierías

Fundamentos de programación en paralelo

REPORTE DE PRÁCTICA

IDENTIFICACIÓN DE LA PRÁCTICA

Práctica	**7**	**Nombre de la práctica**	**Manejo de errores**
Fecha	**15/09/2021**	**Nombre del profesor**	**Alma Nayeli Rodríguez Vázquez**
Nombre del estudiante		**Mariana Ávalos Arce**	

OBJETIVO

El objetivo de esta práctica consiste incluir el manejo de errores en la implementación de un kernel que invierta los elementos de un vector.

PROCEDIMIENTO

Realiza la implementación siguiendo estas instrucciones.

Realiza un programa en C/C++ utilizando CUDA en el que implementes un kernel que invierta el orden de los elementos de un vector de enteros generados aleatoriamente y que guarda los valores invertidos en otro vector considerando los siguientes requerimientos:

- 32 hilos
- Un bloque de una dimensión
- El kernel debe ser como el siguiente:
  __global__ void flipVector(int* vector, int* flippedVector)
- Incluir manejo de errores usando la siguiente función:
  __host__ void check_CUDA_Error(const char* mensaje)

IMPLEMENTACIÓN

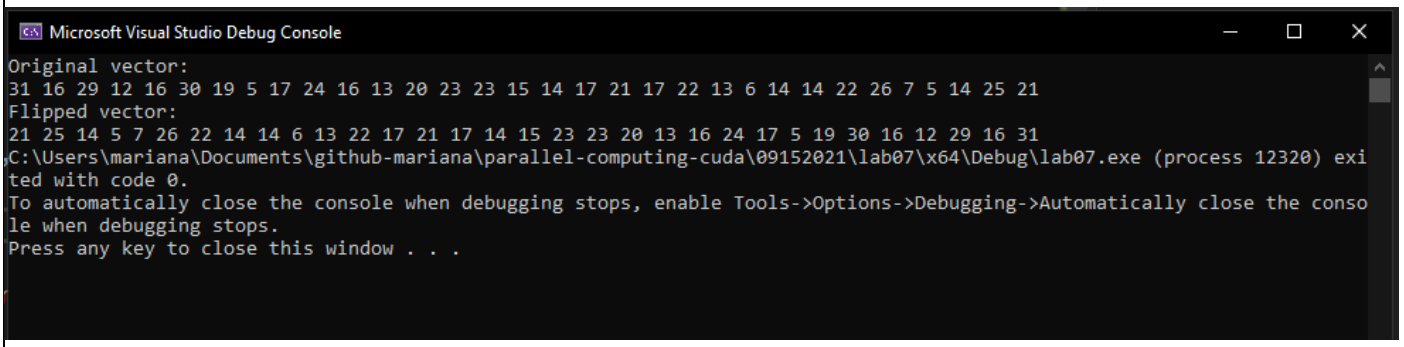
Agrega el código de tu implementación aquí.

```
#include "cuda_runtime.h"
#include "device_launch_parameters.h"
#include <stdio.h>
#include <stdlib.h>

#include <stdlib.h> /* srand, rand */
#include <time.h> /* time */

__host__ void checkCUDAError(const char* msg) {
	cudaError_t error;
	cudaDeviceSynchronize();
	error = cudaGetLastError();
	if (error != cudaSuccess) {
		printf("ERROR %d: %s (%s)\n", error, cudaGetErrorString(error), msg);
	}
}

__global__ void flipVector(int* vector, int* flippedVector) {
```

```
	int gId = threadIdx.x + blockIdx.x * blockDim.x;
	flippedVector[(blockDim.x - 1) - gId] = vector[gId];
}

int main() {
	const int vectorSize = 32;
	int* vector = (int*)malloc(sizeof(int) * vectorSize);
	int* flippedVector = (int*)malloc(sizeof(int) * vectorSize);

	int* devVector, * devFlippedVector;
	cudaMalloc((void**)&devVector, sizeof(int) * vectorSize);
	checkCUDAError("cudaMalloc: devVector");
	cudaMalloc((void**)&devFlippedVector, sizeof(int) * vectorSize);
	checkCUDAError("cudaMalloc: devFlippedVector");

	srand(time(NULL));
	printf("Original vector: \n");
	for (int i = 0; i < vectorSize; i++) {
		int num = rand() % vectorSize + 1;
		vector[i] = num;
		printf("%d ", vector[i]);
	}

	cudaMemcpy(flippedVector, vector, sizeof(int) * vectorSize, cudaMemcpyHostToHost);
	checkCUDAError("cudaMemcpy: vector -> flippedVector, Host -> Host");
	cudaMemcpy(devVector, vector, sizeof(int) * vectorSize, cudaMemcpyHostToDevice);
	checkCUDAError("cudaMemcpy: vector -> devVector, Host -> Device");
	cudaMemcpy(devFlippedVector, flippedVector, sizeof(int) * vectorSize,
cudaMemcpyHostToDevice);
	checkCUDAError("cudaMemcpy: flippedVector -> devFlippedVector, Host -> Device");

	dim3 grid(1);
	dim3 block(vectorSize);

	flipVector << < grid, block >> > (devVector, devFlippedVector);
	checkCUDAError("kernel: flipVector");

	cudaMemcpy(flippedVector, devFlippedVector, sizeof(int) * vectorSize,
cudaMemcpyDeviceToHost);
	checkCUDAError("cudaMemcpy: devFlippedVector -> flippedVector, Device -> Host");

	printf("\nFlipped vector: \n");
	for (int i = 0; i < vectorSize; i++) {
		printf("%d ", flippedVector[i]);
	}
}
```

RESULTADOS

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



CONCLUSIONES

Escribe tus observaciones y conclusiones.

Bastante útil resulta el manejo de errores, sobre todo el manejo de memoria (copia de memoria), ya que copiar bits recae en el programador y puede fallar. Además, esta práctica en específico fue interesante usar el blockDim para obtener el 32 y así invertir el arreglo, pues es un uso más allá del cálculo del globalId. En este manejo de índices también pueden pasar errores y por eso es bastante útil checar el error después del lanzamiento del kernel, en este ejercicio en particular.