ENTREGABLE 4: ORDENACION POR RANGO

Aaron Rojas y David Peirotén

```
/**
*
* ARQUITECTURA DE COMPUTADORES
* 2º Grado en Ingenieria Informatica
* Curso: 2022 - 2023
*
* ENTREGA no.4 <Ordenacion por rango>
*
* EQUIPO : TE - C - 25
* MIEMBROS : Aaron Rojas Gutierrez y David Peiroten Herrero
*
*
*
*/

// includes
#include <stdio.h>
#include <stdlib.h>
#include <device_launch_parameters.h>
#include <time.h>
#include <cuda_runtime.h>

//Prototipos de las funciones s
__host__ void propiedades_Device(int deviceID);

//Funcion que genera un vector con numeros aleatorios
__host__ void generarVector(int *vector1, int N)
{
	srand((int)time(NULL));
	for (int i = 0; i < N; i++)
	{
		vector1[i] = (int)rand() % 30;
	}
}
__host__ int calcula_bloques(int N) {
	int bloques = N / 10;
	if (N % 10 != 0) {
		bloques++;
	}
	return bloques;
}

//Funcion kernel para generar los vectores
__global__ void kernel(int *vector_1, int *vector_2, int *vector_ordenado, int N)
{
	int idGlobal = threadIdx.x + blockDim.x * blockIdx.x;
	int rango = 0;

	for (int i = 0; i < N; i++) {
		if (vector_1[idGlobal] >= vector_1[i]) {
			if (vector_1[idGlobal] > vector_1[i]) {
				rango++;
			}
			else if (vector_1[idGlobal] == vector_1[i] && idGlobal >i) {
				rango++;
			}
```

```
                        else if (vector_1[idGlobal] == vector_1[i] && idGlobal < i) {
                                rango = rango;
                        }
                }

        }
        vector_2[idGlobal] = rango;
        vector_ordenado[rango] = vector_1[idGlobal];
}

int main(int argc, char** argv)
{
        // declaracion de variables
        int *hst_vector1, *hst_vector2, *hst_resultado;
        int *dev_vector1, *dev_vector2, *dev_resultado;
        int numeroThreats, numeroBloques;
        int deviceCount;

        //Cargamos los dispostivios cuda y obtenemos sus datos
        cudaGetDeviceCount(&deviceCount);
        if (deviceCount == 0)
        {
                printf("!!!!!No se han encontrado dispositivos CUDA!!!!!\n");
                return 1;
        }
        else
        {
                printf("Se han encontrado <%d> dispositivos CUDA:\n", deviceCount);
                for (int id = 0; id < deviceCount; id++)
                {
                        propiedades_Device(id);
                }
        }
        //Declarar variables y eventos para monitorizar el tiempi
        cudaEvent_t start;
        cudaEvent_t stop;

        //Creacion de eventos
        cudaEventCreate(&start);
        cudaEventCreate(&stop);

        //Imprimimos los dispostivios y sus propiedades
        time_t fecha;
        time(&fecha);
        printf("*************************************************\n");
        printf("Programa ejecutado el: %s\n", ctime(&fecha));
        printf("Introduce el numero de elementos: ");
        scanf("%d", &numeroThreats);
        getchar();
        numeroBloques = calcula_bloques(numeroThreats);
        printf("> Vector de %d elementos\n", numeroThreats);
        printf("> Lanzamiento con %d bloques de 10 elementos\n", numeroBloques);

        // reserva de memoria en el host
        hst_vector1 = (int*)malloc(numeroThreats * sizeof(int));
        hst_vector2 = (int*)malloc(numeroThreats * sizeof(int));
        hst_resultado = (int*)malloc(numeroThreats * sizeof(int));

        // reserva de memoria en el device
        cudaMalloc((void**)&dev_vector1, numeroThreats * sizeof(int));
        cudaMalloc((void**)&dev_vector2, numeroThreats * sizeof(int));
        cudaMalloc((void**)&dev_resultado, numeroThreats * sizeof(int));
```

```
	// inicializacion del primer vector
	generarVector(hst_vector1, numeroThreats);

	// copiamos el vector 1 en el device
	cudaMemcpy(dev_vector1, hst_vector1, numeroThreats * sizeof(int),
cudaMemcpyHostToDevice);

	// inicializacion del segundo vector y suma
	cudaEventRecord(start, 0);
	kernel << < numeroBloques, 10 >> >(dev_vector1, dev_vector2,
dev_resultado, numeroThreats);
	cudaEventRecord(stop, 0);

	cudaEventSynchronize(stop);
	//Calculo de intentos en milisegundos
	float elapsedTime;

	cudaEventElapsedTime(&elapsedTime, start, stop);

	// recogida de datos desde el device
	cudaMemcpy(hst_vector2, dev_vector2, numeroThreats * sizeof(int),
cudaMemcpyDeviceToHost);
	cudaMemcpy(hst_resultado, dev_resultado, numeroThreats * sizeof(int),
cudaMemcpyDeviceToHost);

	// impresion de resultados
	printf("VECTOR ALEATORIO:\n");
	for (int i = 0; i < numeroThreats; i++)
	{
		printf("%2d ", hst_vector1[i]);
	}
	printf("\n");
	printf("VECTOR RANGOS\n");
	for (int i = 0; i < numeroThreats; i++)
	{
		printf("%2d ", hst_vector2[i]);
	}
	printf("\n");
	printf("VECTOR ORDENADO\n");
	for (int i = 0; i < numeroThreats; i++)
	{
		printf("%2d ", hst_resultado[i]);
	}
	printf("\n");
	printf("*************************************************\n");
	printf("> Tiempo de ejecucion: %f ms\n", elapsedTime);
	printf("*************************************************\n");
	printf("<pulsa [INTRO] para finalizar>");
	getchar();
	// liberacion de recursos
	cudaEventDestroy(start);
	cudaEventDestroy(stop);
	return 0;
}

//Funcion para obtener las caracteristicas del dispositivo
__host__ void propiedades_Device(int deviceID)
{
	cudaDeviceProp deviceProp;
	cudaGetDeviceProperties(&deviceProp, deviceID);
	// calculo del numero de cores (SP)
```

```
int cudaCores = 0;
int SM = deviceProp.multiProcessorCount;
int major = deviceProp.major;
int minor = deviceProp.minor;
const char *archName;
switch (major)
{
case 1:
	//TESLA
	archName = "TESLA";
	cudaCores = 8;
	break;
case 2:
	//FERMI
	archName = "FERMI";
	if (minor == 0)
		cudaCores = 32;
	else
		cudaCores = 48;
	break;
case 3:
	//KEPLER
	archName = "KEPLER";
	cudaCores = 192;
	break;
case 5:
	//MAXWELL
	archName = "MAXWELL";
	cudaCores = 128;
	break;
case 6:
	//PASCAL
	archName = "PASCAL";
	cudaCores = 64;
	break;
case 7:
	//VOLTA(7.0) //TURING(7.5)
	cudaCores = 64;
	if (minor == 0)
		archName = "VOLTA";
	else
		archName = "TURING";
	break;
case 8:
	// AMPERE
	archName = "AMPERE";
	cudaCores = 64;
	break;
default:
	//ARQUITECTURA DESCONOCIDA
	archName = "DESCONOCIDA";
}
int rtV;
cudaRuntimeGetVersion(&rtV);

// presentacion de propiedades
printf("*****************************************************\n");
printf("DEVICE %d: %s\n", deviceID, deviceProp.name);
printf("*****************************************************\n");
printf("> CUDA Toolkit \t: %d.%d\n", rtV / 1000, (rtV % 1000) / 10);
printf("> Arquitectura CUDA \t: %s\n", archName);
printf("> Capacidad de Computo \t: %d.%d\n", major, minor);
```

```
        printf("> No. MultiProcesadores \t: %d\n", SM);
        printf("> No. Nucleos CUDA (%dx%d) \t: %d\n", cudaCores, SM,
cudaCores*SM);
}
```