UNIVERSIDAD PANAMERICANA

Universidad Panamericana

Campus Guadalajara

Escuela de ingenierías

Fundamentos de programación en paralelo

REPORTE DE PRÁCTICA

IDENTIFICACIÓN DE LA PRÁCTICA

Práctica	**2**	**Nombre de la práctica**	**Transferencia de datos**
Fecha	**11/08/2021**	**Nombre del profesor**	**Alma Nayeli Rodríguez Vázquez**
Nombre del estudiante		**Mariana Ávalos Arce**	

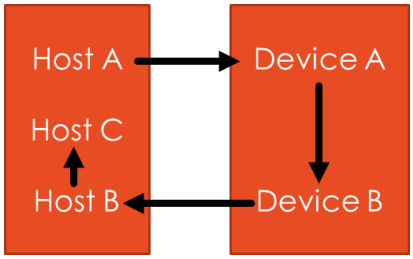
OBJETIVO

El objetivo de esta práctica consiste en transferir datos entre el host y el device utilizando los diferentes tipos de transferencia.

PROCEDIMIENTO

Realiza la implementación siguiendo estas instrucciones.

Realiza un programa en C/C++ utilizando CUDA en el que transfieras un vector de 8 números enteros (inicializados con valores arbitrarios) siguiendo el flujo de transferencia de la figura.



Imprime los valores almacenados en los vectores Host A y Host C para comprobar que son iguales.

IMPLEMENTACIÓN

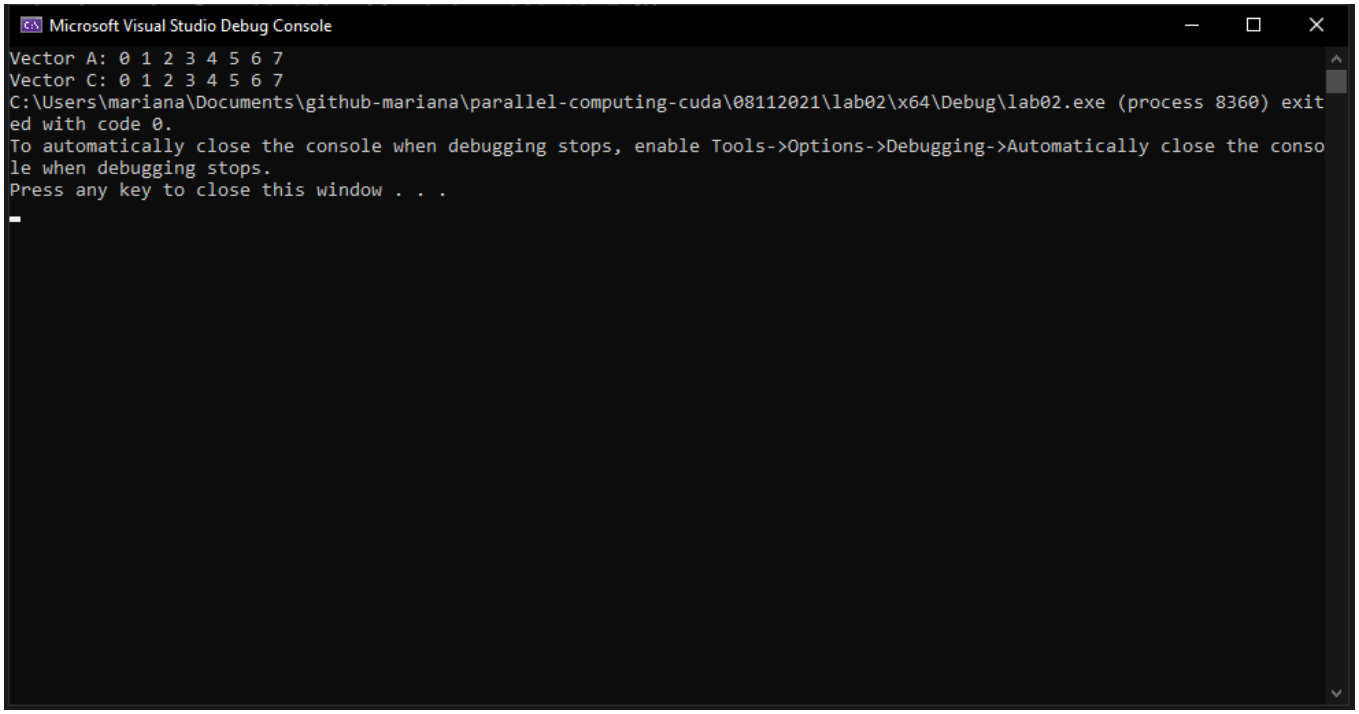
Agrega el código de tu implementación aquí.

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

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

int main()
{
    int N = 8;
    int* host_vectorA;
    int* host_vectorB;
```

```
    int* host_vectorC;

    int* device_vectorA;
    int* device_vectorB;
    int* device_vectorC;

    host_vectorA = (int*)malloc(sizeof(int) * N);
    host_vectorB = (int*)malloc(sizeof(int) * N);
    host_vectorC = (int*)malloc(sizeof(int) * N);

    cudaMalloc((void**)&device_vectorA, sizeof(int) * N);
    cudaMalloc((void**)&device_vectorB, sizeof(int) * N);

    for (int i = 0; i < N; i++) {
        host_vectorA[i] = i;
    }


    cudaMemcpy(device_vectorA, host_vectorA, sizeof(int) * N,
cudaMemcpyHostToDevice);
    cudaMemcpy(device_vectorB, device_vectorA, sizeof(int) * N,
cudaMemcpyDeviceToDevice);
    cudaMemcpy(host_vectorB, device_vectorB, sizeof(int) * N,
cudaMemcpyDeviceToHost);
    cudaMemcpy(host_vectorC, host_vectorB, sizeof(int) * N,
cudaMemcpyHostToHost);

    printf("Vector A: ");
    for (int i = 0; i < N; i++) {
        printf("%d ", host_vectorA[i]);
    }

    printf("\nVector C: ");
    for (int i = 0; i < N; i++) {
        printf("%d ", host_vectorC[i]);
    }

    free(host_vectorA);
    free(host_vectorB);
    free(host_vectorC);
    cudaFree(device_vectorA);
    cudaFree(device_vectorB);

    return 0;
}
```

RESULTADOS

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



CONCLUSIONES

Escribe tus observaciones y conclusiones.

Se concluye al final de esta práctica que efectivamente la memoria del CPU y la memoria del GPU está separada por completo, lo que tiene una no lo tiene la otra. Y de ahí salen las funciones de CUDA para copiar variables o memoria desde el Host al Device, por ejemplo. Al principio, no le vi mucho sentido al usar los tipos de transferencia conocidos como HostToHost o DeviceToDevice, pues si se trata de acciones entre memoria común, pues se aplican las normas de programación simple (ciclos) para copiar memoria o asignarla. Sin embargo, jamás hubiera pensado que estas funciones de CUDA evitan el uso de ciclos para copiar memoria de arreglos o matrices, lo cual optimiza el código y nos quita esa tarea engorrosa de programar ciclos hasta triples cada que queremos copiar información desde estructuras de datos. Me pareció bastante productiva esta práctica para ejemplificar cada tipo de transferencia de datos.