

### PRÁCTICA 3 – MANEJO DE MEMORIA

FECHA: 07/03/2022

GRUPO: 4CDV1

EQUIPO: NETPOWER

Integrantes:

Alcibar Zubillaga Julián
De Luna Ocampo Yanina

### OBJETIVOS

**Parte 1:** Correr y analizar el programa 04\_memo.cu

**Parte 2:** Correr y analizar el programa 05\_suma.cu

### ESCENARIO

El modelo de programación en CUDA asume que se está corriendo sobre un sistema compuesto por un CPU (host) y un GPU (device), cada uno con su espacio de memoria propio. Un kernel sólo puede operar sobre la memoria del GPU. De ahí, que sea necesario usar funciones específicas para reservar la memoria en el GPU y para transferir datos de ésta a la memoria del CPU

En esta práctica deberás correr un programa que comparte datos entre la memoria del CPU y la del GPU.

**Nota:** Se utilizará un servidor externo para realizar la práctica. Es necesario contar con un programa cliente ssh y credenciales de acceso al servidor externo.

### RECURSOS NECESARIOS PARA REALIZAR LA PRÁCTICA

- 1 computadora personal
- Acceso a internet
- 1 programa cliente SSH\* (Putty: <https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html>)

## INTRODUCCIÓN

La programación en CUDA está basada en tres abstracciones básicas: una jerarquía de grupos de hilos, otra de tipos de memoria y barreras de sincronización. La estructura que conforma la jerarquía de hilos que se ejecutan en el dispositivo NVIDIA (device) se agrupan en tres elementos: mallas, bloques e hilos.

El uso de las memorias es un aspecto fundamental en la programación de GPUs. El uso incorrecto de la jerarquía de memorias disponible en las GPUs puede ocasionar que no consigamos acelerar el algoritmo que estamos implementando de forma paralela, sino que además su tiempo de ejecución sea superior a una implementación secuencial en la CPU. Por ello, la elección del tipo de memoria más adecuada, así como el acceso a los datos en memoria, es realmente importante. Por mucha capacidad de cómputo que tenga nuestra GPU o CPU, si la velocidad de acceso/escritura de las memorias no es suficiente, toda esta capacidad de cómputo estará desaprovechada [2].

Con ayuda de PuTTY, que es un emulador de terminal que admite varios protocolos de red tal como ssh. Esto permite correr comandos UNIX en tu servidor el cual no está disponible cuando te conectas usando un cliente FTP [1].



### PARTE 1: CORRER Y ANALIZAR EL PROGRAMA 04\_MEMO.CU

UTILIZA UN CLIENTE SSH PARA CONECTARTE AL SERVIDOR LAMBDA 01.

(TE PUEDES CONECTAR AL SERVIDOR LAMBDA DE LA MISMA FORMA QUE LO HICISTE EN LA PRÁCTICA 01).

Dentro del servidor Lambda. Acceder a la carpeta “hpc\_4AV1

Copiaremos el programa 04\_memo.cu a nuestra carpeta de trabajo, ejecuta el siguiente comando:

```
cp ~/hpc_4AV1/unidad01/04_memo.cu ~/hpc_4AV1/estudiantes/<tu_carpeta_de_trabajo>/u01
```

Reemplaza <tu\_carpeta\_de\_trabajo> por el nombre de tu carpeta de trabajo. Este comando copiará el programa “04\_memo.cu” a la subcarpeta u01 de tu carpeta de trabajo. Accede a esa subcarpeta y abre el programa, usa el comando:

```
nano 04_memo.cu
```

Revisa el programa y analízalo.

Para reservar espacio en la memoria global del GPU y poder acceder a ella desde el host se utiliza la función *cudaMalloc()*, que tiene un comportamiento similar a la correspondiente función estándar de C:

```
cudaMalloc(void **devPtr, size_t size);
```

El primer argumento es un doble puntero y corresponde a la dirección de memoria del puntero (devPtr) en donde se va a almacenar la dirección de la memoria reservada en el GPU. Por otro lado, el segundo argumento es la cantidad de memoria expresada en bytes que se va a reservar.

Una vez reservado el espacio de memoria global en el GPU, el siguiente paso es transferir datos entre esta memoria y la del CPU. En este caso se puede usar la función *cudaMemcpy()*.

```
cudaMemcpy(void *dst, void *src, size_t count, cudaMemcpyKind kind);
```

El primer parámetro (dst) corresponde al puntero con la dirección de destino de los datos, el segundo (src) es el puntero con la dirección de origen (donde se encuentran los datos que se quieren copiar), count es el número de bytes se van a transferir y kind es el tipo de transferencia que se va a realizar:

- **cudaMemcpyHostToHost** CPU al CPU
- **cudaMemcpyHostToDevice** CPU al GPU
- **cudaMemcpyDeviceToHost** GPU al CPU
- **cudaMemcpyDeviceToDevice** GPU al GPU

Compila y ejecuta el programa, usa los siguientes comandos:

```
nvcc 04_memo.cu -o 04_memo.x
```

.04\_memo.x

INCLUYE AQUÍ LA CAPTURA DE PANTALLA CON EL RESULTADO DEL PROGRAMA EJECUTÁNDOSE

#### A) CAPTURAS ALCIBAR ZUBILLAGA JULIÁN

```
bcruz@labdatos01: ~/hpc_4AV1/estudiantes/Alcibar/
GNU nano 4.8                                04 memo.cu
// includes
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <cuda_runtime.h>
#define N 8

// MAIN: rutina principal ejecutada en el CPU
int main(int argc, char** argv)
{
    // declaracion
    float *hst_matriz; //Espacio de memoria en el CPU
    float *dev_matriz; //Espacio de memoria en el GPU
    // reserva en el CPU
    hst_matriz = (float*)malloc( N*N*sizeof(float) );
    // Reserva en el GPU
    cudaMalloc( (void**)&dev_matriz, N*N*sizeof(float) );
    // inicializacion de datos
    srand ( (int)time(NULL) );
    for (int i=0; i<N*N; i++)
    {
        hst_matriz[i] = (float)( rand() % 10 );
    }

    // copia de datos
    cudaMemcpy(dev_matriz, hst_matriz, N*N*sizeof(float), cudaMemcpyHostToDevice);

    // salida
    cudaFree( dev_matriz );

    printf("\npulsa INTRO para finalizar...");
    fflush(stdin);

    char tecla = getchar();
    return 0;
}
```

Captura de pantalla con la ejecución del programa

#### B) CAPTURAS DE LUNA OCAMPO YANINA

```
bcruz@labdatos01: ~/hpc_4AV1/estudiantes/DelunaY
GNU nano 4.8                                04 memo.cu
// includes
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <cuda_runtime.h>
#define N 8

// MAIN: rutina principal ejecutada en el CPU
int main(int argc, char** argv)
{
    // declaracion
    float *hst_matriz; //Espacio de memoria en el CPU
    float *dev_matriz; //Espacio de memoria en el GPU
    // reserva en el CPU
    hst_matriz = (float*)malloc( N*N*sizeof(float) );
    // Reserva en el GPU
    cudaMalloc( (void**)&dev_matriz, N*N*sizeof(float) );
    // inicializacion de datos
    srand ( (int)time(NULL) );
    for (int i=0; i<N*N; i++)
    {
        hst_matriz[i] = (float)( rand() % 10 );
    }

    // copia de datos
    cudaMemcpy(dev_matriz, hst_matriz, N*N*sizeof(float), cudaMemcpyHostToDevice);

    // salida
    cudaFree( dev_matriz );

    printf("\npulsa INTRO para finalizar...");
    fflush(stdin);

    char tecla = getchar();
    return 0;
}
```

Captura de pantalla con la ejecución del programa

Copiaremos el programa 05\_suma.cu a nuestra carpeta de trabajo, ejecuta el siguiente comando:

```
cp ~/hpc_4AV1/unidad01/05_suma.cu ~/hpc_4AV1/estudiantes/<tu_carpeta_de_trabajo>/u01
```

Reemplaza <tu\_carpeta\_de\_trabajo> por el nombre de tu carpeta de trabajo. Este comando copiará el programa “05\_suma.cu” a la subcarpeta u01 de tu carpeta de trabajo. Accede a esa subcarpeta y abre el programa, usa el comando:

```
nano 05_suma.cu
```

Revisa el programa y analízalo.

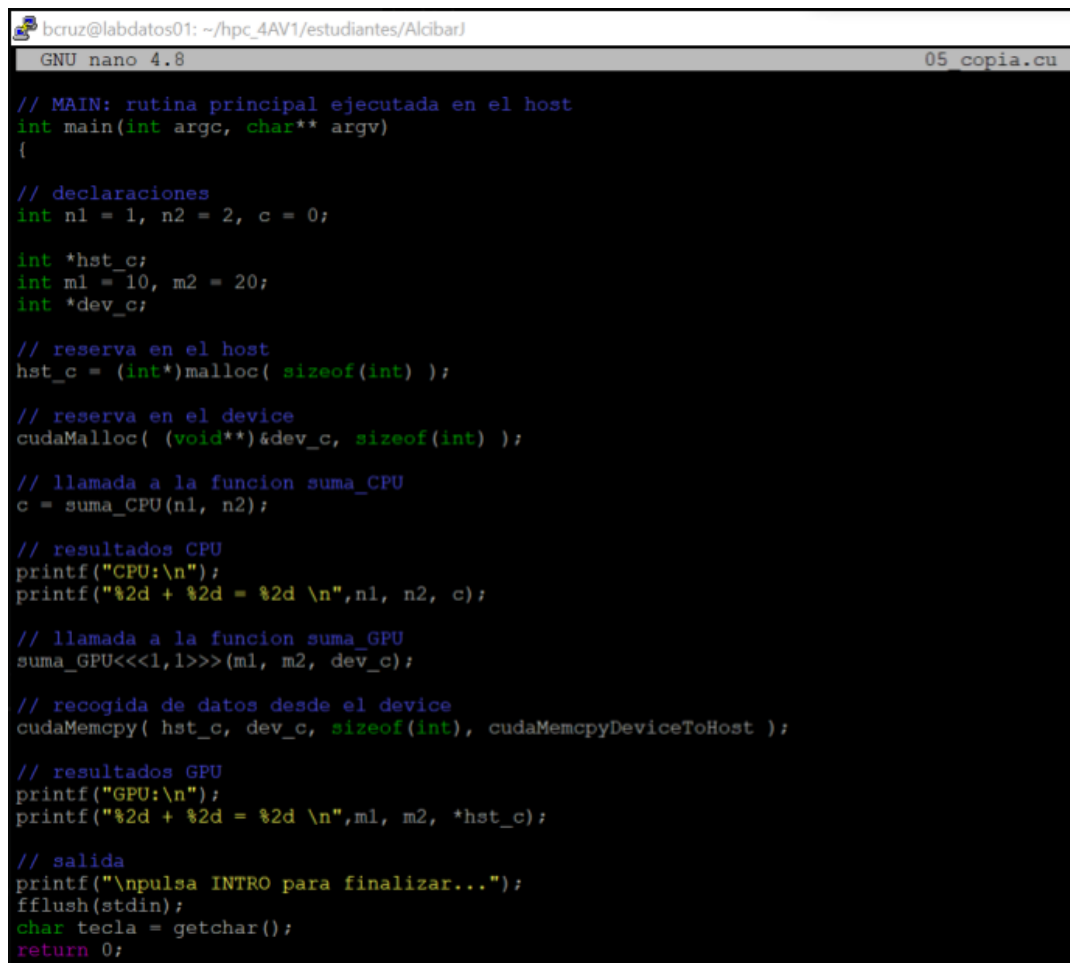
Compila y ejecuta el programa, usa los siguientes comandos:

```
nvcc 05_suma.cu -o 05_suma.x
```

```
./05_suma.x
```

INCLUYE AQUÍ LA CAPTURA DE PANTALLA CON EL RESULTADO DEL PROGRAMA EJECUTÁNDOSE

### c) CAPTURAS ALCIBAR ZUBILLAGA JULIÁN



```
bacruz@labdatos01: ~/hpc_4AV1/estudiantes/Alcibar/
GNU nano 4.8                                05_copia.cu

// MAIN: rutina principal ejecutada en el host
int main(int argc, char** argv)
{

// declaraciones
int n1 = 1, n2 = 2, c = 0;

int *hst_c;
int m1 = 10, m2 = 20;
int *dev_c;

// reserva en el host
hst_c = (int*)malloc( sizeof(int) );

// reserva en el device
cudaMalloc( (void*)&dev_c, sizeof(int) );

// llamada a la funcion suma_CPU
c = suma_CPU(n1, n2);

// resultados CPU
printf("CPU:\n");
printf("%2d + %2d = %2d \n",n1, n2, c);

// llamada a la funcion suma_GPU
suma_GPU<<<1,1>>>(m1, m2, dev_c);

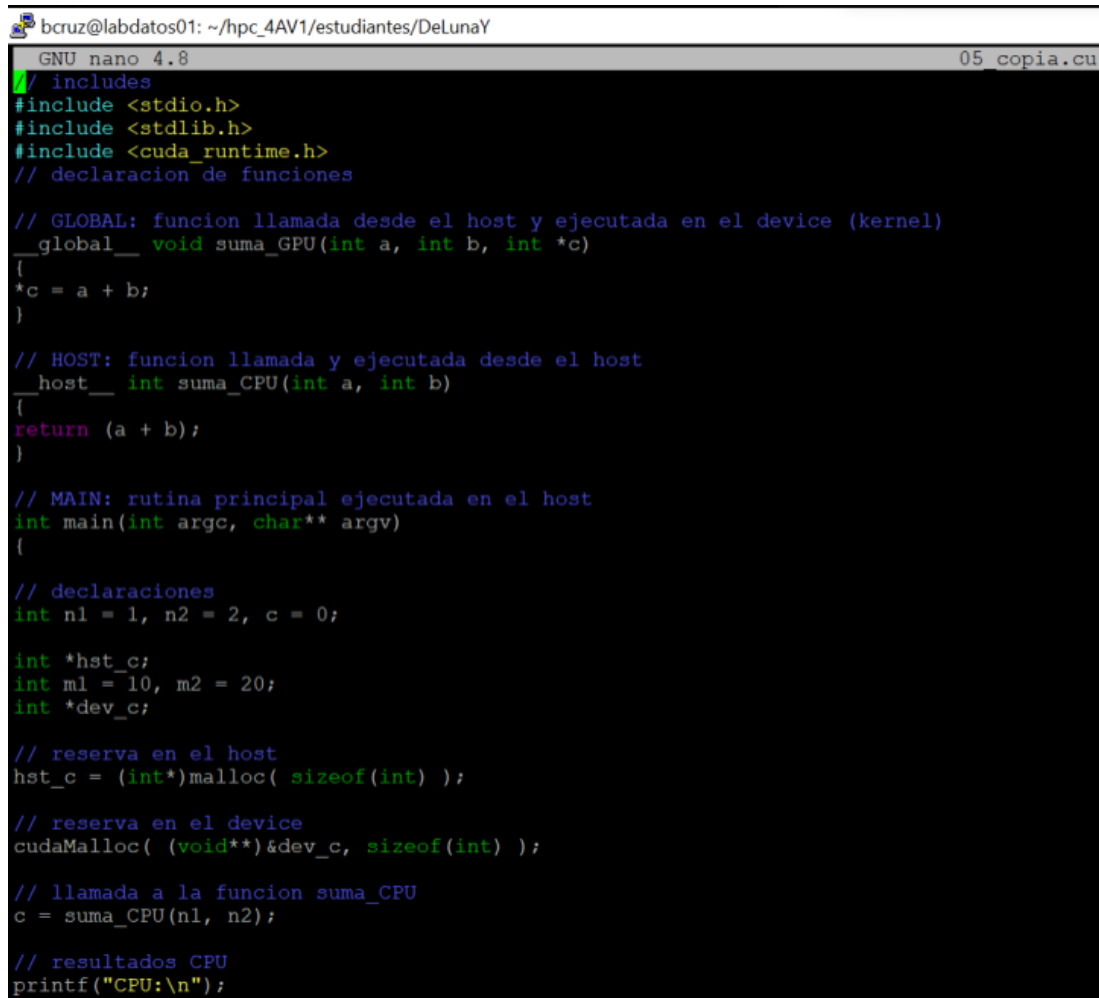
// recogida de datos desde el device
cudaMemcpy( hst_c, dev_c, sizeof(int), cudaMemcpyDeviceToHost );

// resultados GPU
printf("GPU:\n");
printf("%2d + %2d = %2d \n",m1, m2, *hst_c);

// salida
printf("\npulsa INTRO para finalizar...");
fflush(stdin);
char tecla = getchar();
return 0;
```

Captura de pantalla con la ejecución del programa

## d) CAPTURAS DE LUNA OCAMPO YANINA



```
GNU nano 4.8 05_copia.cu
// includes
#include <stdio.h>
#include <stdlib.h>
#include <cuda_runtime.h>
// declaracion de funciones

// GLOBAL: funcion llamada desde el host y ejecutada en el device (kernel)
__global__ void suma_GPU(int a, int b, int *c)
{
    *c = a + b;
}

// HOST: funcion llamada y ejecutada desde el host
__host__ int suma_CPU(int a, int b)
{
    return (a + b);
}

// MAIN: rutina principal ejecutada en el host
int main(int argc, char** argv)
{
    // declaraciones
    int n1 = 1, n2 = 2, c = 0;

    int *hst_c;
    int m1 = 10, m2 = 20;
    int *dev_c;

    // reserva en el host
    hst_c = (int*)malloc( sizeof(int) );

    // reserva en el device
    cudaMalloc( (void**)&dev_c, sizeof(int) );

    // llamada a la funcion suma_CPU
    c = suma_CPU(n1, n2);

    // resultados CPU
    printf("CPU:\n");
```

Captura de pantalla con la ejecución del programa

## PARTE 3: CUESTIONARIO

## ¿QUÉ CAMBIARÍAS DEL PROGRAMA PARA QUE SUME ARREGLOS EN LUGAR DE ENTEROS?

En la parte de los argumentos, en vez de pasar como parámetros los números enteros, pasaremos arreglos mediante arreglos los cuales nos servirán para que no estemos haciendo tantas copias, por último con un for iteramos los arreglos y por último sumaremos de uno en una cada uno de esto.

Evidencia de impresión del código:

a) Impresión Alcibar Zubillaga Julián

```
(base) bcruz@labdatos01:~/hpc_4AV1/estudiantes/AlcibarJ$ nano 05_copia.cu
(base) bcruz@labdatos01:~/hpc_4AV1/estudiantes/AlcibarJ$ nvcc 05_copia.cu -o 05_copia.x
(base) bcruz@labdatos01:~/hpc_4AV1/estudiantes/AlcibarJ$ ./05_copia.x
CPU:
 1 + 2 = 3
GPU:
10 + 20 = 30
pulsa INTRO para finalizar...
```

### b) Impresión De Luna Ocampo Yanina

```
(base) bcruz@labdatos01:~/hpc_4AV1/estudiantes/DeLunaY$ nvcc 05_copia.cu -o 05_copia.x
(base) bcruz@labdatos01:~/hpc_4AV1/estudiantes/DeLunaY$ ./05_copia.x
CPU:
 1 + 2 = 3
GPU:
10 + 20 = 30
pulsa INTRO para finalizar...
```

---

#### DE LA PREGUNTA ANTERIOR ¿QUÉ PARALELIZARÍAS EN EL CASO ANTERIOR?

Podemos paralelizar la impresión de la suma también, ya que si de una vez se realiza la operación, de una vez se puede imprimir para que este proceso no se haga en serie después de que se pasen los datos de un lado a otro.

### CONCLUSIONES

Recordamos algunas capacidades de aprovechamiento de las GPU, como los son: grandes vectores de datos, paralelismo de grano fino SIMD y por mencionar una más, la baja latencia en operaciones en punto flotante. Estas, como todo, han ido evolucionando, empezamos con los ordenadores, los cuales no tenían tarjetas gráficas, tiempo después aparecen para ayudarnos a la presentación en pantalla, de ahí nos pasamos a la industria de videojuegos, hasta que en 2004, surge la idea de utilizarlas para la computación de alto desempeño, en 2007, NVIDIA, explicada en la introducción de este documento, ve oportunidad en el mercado y desarrolla por fin, CUDA, que es una tecnología que permite obtener grandes rendimientos para problemas con un alto paralelismo. Hay que tener claro su funcionamiento para saber si es adecuado y obtener el mayor rendimiento posible. Con esto podemos decir que programar en CUDA es fácil pero no lo es para obtener rendimiento. Actualmente NVIDIA tiene un gran éxito en la computación GPGPU, las supercomputadoras top utilizan clústers de GPUs para aumentar su potencia computacional.

### BIBLIOGRAFÍA

1. *Security check.* (2022). Descargar PuTTY. <https://help.dreamhost.com/hc/es/articles/215464538-Descargar-PuTTY#:~:text=PuTTY%20es%20un%20emulador%20de,conectas%20usando%20un%20cliente%20FTP>.
2. Inicio. Departamento de Tecnología Informática y Computación. [https://www.dtic.ua.es/jgpu11/material/sesion1\\_jgpu11.pdf](https://www.dtic.ua.es/jgpu11/material/sesion1_jgpu11.pdf) (accedido el 8 de marzo de 2022).

### CONSIDERACIONES FINALES

Descarga el documento antes de llenarlo.

Este documento se debe llenar en equipo, aunque la práctica la deben hacer TODOS los integrantes de este.

Después de llenar el documento, guárdalo como PDF y envíalo a través de la plataforma TEAMS, en la pestaña de tareas correspondiente. Solamente lo tiene que subir uno de los integrantes. Pero deben incluir TODOS los nombres de los integrantes del equipo en la primera página.

Queda estrictamente prohibido cualquier tipo de plagio a otros equipos o grupos. En caso de que ocurra, se anulará la práctica y se descontarán dos puntos a los equipos involucrados.