

Universidad del valle de Guatemala  
Departamento de ciencia de la computación  
Programación paralela y distribuida  
MIGUEL NOVELLA LINARES

01/06/2023  
Andres Emilio Quinto - 18288



*Excelencia que trasciende*

**DEL VALLE**  
GRUPO EDUCATIVO

**Hoja de trabajo 3 - CUDA**

## 2.0 hello.cu

Ejecute el programa. Observe cuántas veces se imprime el mensaje y su conexión con la configuración de la llamada al kernel – `hello<<<g,b>>>()`:

```
$ ./hello
```

2.3. Modifique el programa para correr 2 bloques de 1024 hilos. Modificarlo también para que imprima su nombre y carnet. Busque en el despliegue de consola el mensaje del último hilo de la serie (1023).

***(5 PTS) CAPTURA DE PANTALLA DE LA EJECUCIÓN CON 2 BLOQUES DE 1024 HILOS***

```
*/
#include <stdio.h>
#include <cuda.h>

__global__ void hello()
{
    int myID = ( blockIdx.x ) * blockDim.x +
               threadIdx.x;
    if (myID == 1023 || myID == 2047) {
        printf("%d\n", myID);
    }
}

int main()
{
    hello<<<2, 1024>>>();
    cudaThreadSynchronize(); //deprecated
    printf("Hola, soy Andres Q!! 18288\n");
    return 0;
}
```

```
1023
2047
Hola, soy Andres Q!! 18288
```

2.4 Busque en el sitio de Nvidia el Compute Capability de la tarjeta que poseen las máquinas del Laboratorio (o de su computador, en caso tenga tarjeta NVIDIA y estén haciendo todo esto en su compu). Escriba acá el valor de CC y busque la tabla resumen con las características del CC:

Visitando la página web oficial de nvidia en su sección en la cual explican lo de la CC, pude consultar a la tabla de las RTX serie 3000, donde encontré que mi CC es de **7.5** (modelo RTX 3050)

## NVIDIA Quadro and NVIDIA RTX Mobile GPUs

GPU	Compute Capability
RTX A5000	8.6
RTX A4000	8.6
RTX A3000	8.6
RTX A2000	8.6
RTX 5000	7.5
RTX 4000	7.5
RTX 3000	7.5

*Imagen 1, Compute capability de mi gráfica rtx 3050*

Referencia de la web:

[CUDA GPUs - Compute Capability | NVIDIA Developer](#)

2.5 Modifique el programa para correr 1 bloque de 2048 hilos. Coloque una captura de pantalla de la salida y busque en la tabla de CC el siguiente dato:

**(5 PTS) Maximum number of threads per block:**

**(10 PTS) CAPTURA DE PANTALLA DE LA EJECUCIÓN CON 1 BLOQUE DE 2048 HILOS**

**(10 PTS) EXPLIQUE EN POCAS PALABRAS EL RESULTADO**

Citando lo que dice la pagina de nvidia: *There is a limit to the number of threads per block, since all threads of a block are expected to reside on the same streaming multiprocessor core and must share the limited memory resources of that core. On current GPUs, a thread block may contain up to 1024 threads*

Es decir el maximum number of threads per block es 1024 para mi grafica.

**the code:**

```
#include <stdio.h>
#include <cuda.h>

#define BLOCK_SIZE 2048

__global__ void hello()
{
    int myID = (blockIdx.x * blockDim.x) + threadIdx.x;
    if (myID < BLOCK_SIZE) {
        printf("%d\n", myID);
    }
}

int main()
{
    hello<<<1, BLOCK_SIZE>>>>();
    cudaDeviceSynchronize();
    printf("Hola, soy Andres Q!! 18288\n");
    return 0;
}
```

**the output:**

```
javier@javier-BELL: ~/Documents/040_11vo_Semestre/FANALCEN/Proyectos/nucleo : ./hello
Hola, soy Andres Q!! 18288
```

Si intentas ejecutar un solo bloque con 2048 hilos en una GPU con este límite, se producirá un error y la ejecución del programa no será correcta o esperada.

### 3.1 Hello2.cu

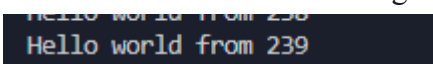
Descargue, compile y ejecute hello2.cu. Observe la relación de la configuración de la llamada al kernel con la geometría de los hilos y el resultado. Escriba la respuesta a los dos enunciados:

#### i. **Máximo ID de los hilos:**

En este caso, las dimensiones de la cuadrícula son (4, 3, 2) y las dimensiones del bloque son (10). Entonces:

Máximo ID de los hilos =  $(4 * 3 * 2 * 10) - 1 = 239$

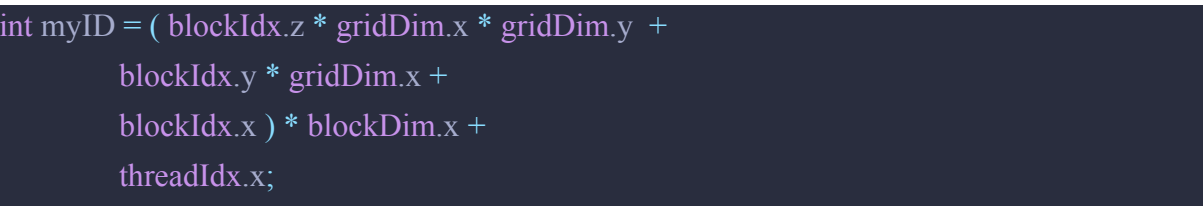
En este caso efectivamente tengo de output: ID 239

A screenshot of a terminal window showing the output of a program. The text "Hello world from 239" is displayed in a light blue font on a dark background.

#### ii. **Ejecución de los hilos en orden:**

La ejecución de los hilos en orden depende de cómo se organizan los hilos en la cuadrícula y en el bloque, y cómo se asignan a los hilos en el dispositivo CUDA. En este caso, se utiliza un bucle de tres dimensiones para recorrer los bloques en la cuadrícula y un bucle de una dimensión para recorrer los hilos dentro de cada bloque.

El orden de ejecución de los hilos se determina por la forma en que se calcula myID, que es el identificador de cada hilo. En este caso, myID se calcula como:

A screenshot of C++ code in a dark-themed editor. The code defines the calculation of myID based on grid dimensions, block dimensions, and thread indices.

```
int myID = ( blockIdx.z * gridDim.x * gridDim.y +  
            blockIdx.y * gridDim.x +  
            blockIdx.x ) * blockDim.x +  
            threadIdx.x;
```

Esto significa que los hilos se ejecutarán en el siguiente orden:

1. El primer hilo (myID = 0) se ejecutará en el hilo con el índice más bajo en la cuadrícula y el bloque.
2. Los hilos se ejecutarán secuencialmente dentro de cada bloque, desde el hilo con threadIdx.x = 0 hasta threadIdx.x = 9.
3. Luego, se pasará al siguiente bloque dentro de la cuadrícula y los hilos dentro de ese bloque se ejecutarán secuencialmente.

Este proceso continuará hasta que se ejecuten todos los hilos en la cuadrícula. Es decir en orden ascendente.

```
Hello world from 230
Hello world from 231
Hello world from 232
Hello world from 233
Hello world from 234
Hello world from 235
Hello world from 236
Hello world from 237
Hello world from 238
Hello world from 239
```

3.2 Observe que la fórmula genérica para cálculo del ID global está en los comentarios. Modifique el programa para que imprima también su nombre y carnet. Luego, realice la siguiente modificación al programa (al inicio del main) y use la fórmula genérica para derivar el nuevo cálculo de ID:

```
Hello world from thread 242, my name is Andres Quinto and my student ID is 18288
Hello world from thread 243, my name is Andres Quinto and my student ID is 18288
Hello world from thread 244, my name is Andres Quinto and my student ID is 18288
Hello world from thread 245, my name is Andres Quinto and my student ID is 18288
Hello world from thread 246, my name is Andres Quinto and my student ID is 18288
Hello world from thread 247, my name is Andres Quinto and my student ID is 18288
Hello world from thread 248, my name is Andres Quinto and my student ID is 18288
Hello world from thread 249, my name is Andres Quinto and my student ID is 18288
Hello world from thread 250, my name is Andres Quinto and my student ID is 18288
Hello world from thread 251, my name is Andres Quinto and my student ID is 18288
Hello world from thread 252, my name is Andres Quinto and my student ID is 18288
Hello world from thread 253, my name is Andres Quinto and my student ID is 18288
Hello world from thread 254, my name is Andres Quinto and my student ID is 18288
Hello world from thread 255, my name is Andres Quinto and my student ID is 18288
```

**(10 PTS) FÓRMULA PARA CALCULAR EL ID GLOBAL Y SALIDA DE PANTALLA**

**fórmula:**

```
{
    int myID = ( blockIdx.z * gridDim.x * gridDim.y +
                blockIdx.y * gridDim.x +
                blockIdx.x ) * blockDim.x +
                threadIdx.x;

    printf ("Hello world from thread %i, my name is Andres Quinto and my student ID is
18288\n", myID);
}
```

**(5 PTS) CAPTURA DE PANTALLA DE LA NUEVA CONFIGURACIÓN (buscar el mensaje impreso por el hilo con el máximo ID global)**

**hilo maximo 255**

Esta salida muestra el ID global de cada hilo, comenzando desde 0 y aumentando secuencialmente. Junto con eso, se muestra el nombre "Andres Quinto" y el carné "18288" para cada hilo.

```
Hello world from thread 252, my name is Andres Quinto and my student ID is 18288
Hello world from thread 253, my name is Andres Quinto and my student ID is 18288
Hello world from thread 254, my name is Andres Quinto and my student ID is 18288
Hello world from thread 255, my name is Andres Quinto and my student ID is 18288
```

3.3 Revise nuevamente la información del Compute Capability respecto a las dimensiones máximas de hilos-bloque en x, y, & z para una grilla. Cree una configuración para lanzar exitosamente el kernel para procesar 100,000 datos. (Sugerencia: busque una configuración que lance como mínimo 100,000 hilos. Modifique el kernel para que imprima el mensaje únicamente si es el ID global máximo)

***(10 PTS) MOSTRAR SU CONFIGURACIÓN USADA***

***(5 PTS) FÓRMULA PARA CALCULAR EL ID GLOBAL Y SALIDA DE PANTALLA***

***(5 PTS) CAPTURA DE PANTALLA CORRIENDO EL CODIGO CON LA NUEVA CONFIGURACIÓN (con el mensaje impreso por el hilo con el máximo ID global)***

Configuración:

**Cuadrícula (Grid): dim3 g(313, 1)**

**La cuadrícula tiene 313 bloques en la dimensión x y 1 bloque en la dimensión y.**

**Bloque (Block): dim3 b(320, 1)**

**Cada bloque tiene 320 hilos en la dimensión x y 1 hilo en la dimensión y.**

Usamos la fórmula de:

```
int myID = (blockIdx.z * gridDim.x * gridDim.y +
            blockIdx.y * gridDim.x +
            blockIdx.x) * blockDim.x +
            threadIdx.x;

if(myID == 99999) { // ID global máximo para 100,000 hilos (99999 = 100,000 - 1)
    printf("Hello world from thread %i (ID global máximo), my name is Andres Quinto and my student ID is 18288\n", myID);
}
```

salida en consola:

```
Hello world from thread 99999 (ID global máximo), my name is Andres Quinto and my student ID is 18288
```



4.2 vectorAdd.cu Modifique el programa para que utilice memoria compartida en la suma de vectores, por ahora de forma estática (vectores de tamaño fijo/conocido a priori). Asegurese que los resultados sean correctos.

**(10 PTS) SALIDA DE PANTALLA CORRIENDO EL PROGRAMA CON MEMORIA COMPARTIDA ESTÁTICA DE FORMA CORRECTA.**

```
vector3/hdt3temp$ nvcc vectorAdd.cu -o vectorAdd
vector3/hdt3temp$ ./vectorAdd
vector3/hdt3temp$
```

4.3 Modifique el programa para que el usuario indique el tamaño de los vectores. Utilice memoria compartida en la suma de vectores, esta vez de forma dinámica para acomodar ese cambio. Asegurese que los resultados sean correctos.

**(10 PTS) SALIDA DE PANTALLA CORRIENDO EL PROGRAMA CON MEMORIA COMPARTIDA DINÁMICA DE FORMA CORRECTA.**

```
Enter the size of the vectors: 10000
Time elapsed: 0.105792 ms
```

4.5 tabla de tiempos y probando todas las versiones de memoria:

**(10 PTS) COMPARACIÓN DE TIEMPOS DE EJECUCIÓN DE LOS 3 PROGRAMAS ORDENADOS EN UNA TABLA, ASÍ COMO LOS COMENTARIOS Y DISCUSIÓN SOBRE LOS RESULTADOS.**

N	Memoria Global ms	Memoria Dinámica ms	Memoria Estática ms
1	1.927	2.480	2.093
2	2.834	2.140	2.154
3	2.814	3.403	2.693
4	2.062	2.491	2.057
5	2.777	2.242	3.121

Promedio	2.4828	2.5512	2.4326
----------	--------	--------	--------

Comentarios y discusión:

**Memoria Global:** Este es el tipo más simple de memoria en la GPU. Se aloja fuera de los bloques de hilos y tiene una latencia relativamente alta para acceder a los datos.

**Memoria Compartida Estática:** La memoria compartida es más rápida que la memoria global y puede ser utilizada para compartir datos entre los hilos dentro de un mismo bloque. En este caso, estás definiendo el tamaño de la memoria compartida en tiempo de compilación.

**Memoria Compartida Dinámica:** Al igual que la memoria compartida estática, la memoria compartida dinámica es más rápida que la memoria global y se puede utilizar para compartir datos entre hilos dentro de un mismo bloque. La diferencia es que el tamaño de la memoria compartida se define en tiempo de ejecución.

Es de notar que, en este caso, la memoria global parece ser ligeramente más rápida en promedio que los métodos de memoria compartida. Esto es contrario a lo que uno esperaría intuitivamente, ya que la memoria compartida generalmente tiene menor latencia que la memoria global.

Sin embargo, hay que tener en cuenta que el beneficio de la memoria compartida se manifiesta principalmente en situaciones donde hay un alto grado de acceso a los datos compartidos entre los hilos. En el código, cada hilo realiza su cálculo de forma independiente, por lo que no se benefician significativamente del uso de la memoria compartida.

Por último, se debe tener en cuenta que la arquitectura específica de la GPU, el tamaño de los vectores y el número de bloques e hilos utilizados pueden influir en estos resultados. En ciertos escenarios, podrías observar una mejora significativa al usar memoria compartida.

En resumen, aunque los tiempos de ejecución de los tres métodos son similares en este caso, es posible que veamos diferencias más grandes en escenarios diferentes, especialmente cuando los cálculos son más complejos y/o requieren una comunicación significativa entre hilos.

Anexos de las corridas:

Global:

```
Global Memory
Execution Time: 1.927 ms
./mnt/d/Documents/UVG/11vo Semestre/PALELA/Proyecto3/hdt3temp$ ./vectorAddF
Global Memory
Execution Time: 2.834 ms
./mnt/d/Documents/UVG/11vo Semestre/PALELA/Proyecto3/hdt3temp$ ./vectorAddF
Global Memory
Execution Time: 2.814 ms
./mnt/d/Documents/UVG/11vo Semestre/PALELA/Proyecto3/hdt3temp$ ./vectorAddF
Global Memory
Execution Time: 2.062 ms
./mnt/d/Documents/UVG/11vo Semestre/PALELA/Proyecto3/hdt3temp$ ./vectorAddF
Global Memory
Execution Time: 2.777 ms
./mnt/d/Documents/UVG/11vo Semestre/PALELA/Proyecto3/hdt3temp$
```

Static Shared:

```
./mnt/d/Documents/UVG/11vo Semestre/PALELA/Proyecto3/hdt3temp$ ./vectorAddF
Static Shared Memory
Execution Time: 2.093 ms
./mnt/d/Documents/UVG/11vo Semestre/PALELA/Proyecto3/hdt3temp$ ./vectorAddF
Static Shared Memory
Execution Time: 2.154 ms
./mnt/d/Documents/UVG/11vo Semestre/PALELA/Proyecto3/hdt3temp$ ./vectorAddF
Static Shared Memory
Execution Time: 2.693 ms
./mnt/d/Documents/UVG/11vo Semestre/PALELA/Proyecto3/hdt3temp$ ./vectorAddF
Static Shared Memory
Execution Time: 2.057 ms
```

```
Static Shared Memory
Execution Time: 3.121 ms
```

Dynamic:

```
./mnt/d/Documents/UVG/11vo Semestre/PALELA/Proyecto3/hdt3temp$ ./vectorAddF
Dynamic Shared Memory
Milliseconds: 2.480 ms
Seconds: 2.000 s
./mnt/d/Documents/UVG/11vo Semestre/PALELA/Proyecto3/hdt3temp$ ./vectorAddF
Dynamic Shared Memory
Milliseconds: 2.140 ms
Seconds: 2.000 s
```

```
./mnt/d/Documents/UVG/11vo Semestre/PALELA/Proyecto3/hdt3temp$ ./vectorAddF
Dynamic Shared Memory
Milliseconds: 3.403 ms
Seconds: 3.000 s
./mnt/d/Documents/UVG/11vo Semestre/PALELA/Proyecto3/hdt3temp$ ./vectorAddF
Dynamic Shared Memory
Milliseconds: 2.491 ms
Seconds: 2.000 s
./mnt/d/Documents/UVG/11vo Semestre/PALELA/Proyecto3/hdt3temp$ ./vectorAddF
Dynamic Shared Memory
Milliseconds: 2.242 ms
Seconds: 2.000 s
```