UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
Facultad de Ingeniería

Introducción a Cómputo Paralelo con CUDA C/C++

LABORATORIO DE INTEL Y CÓMPUTO DE ALTO DESEMPEÑO

Elaboran: Ariel Ulloa Trejo
Jaime Beltrán Rosales
Revisión: Ing. Laura Sandoval Montaño

Temario

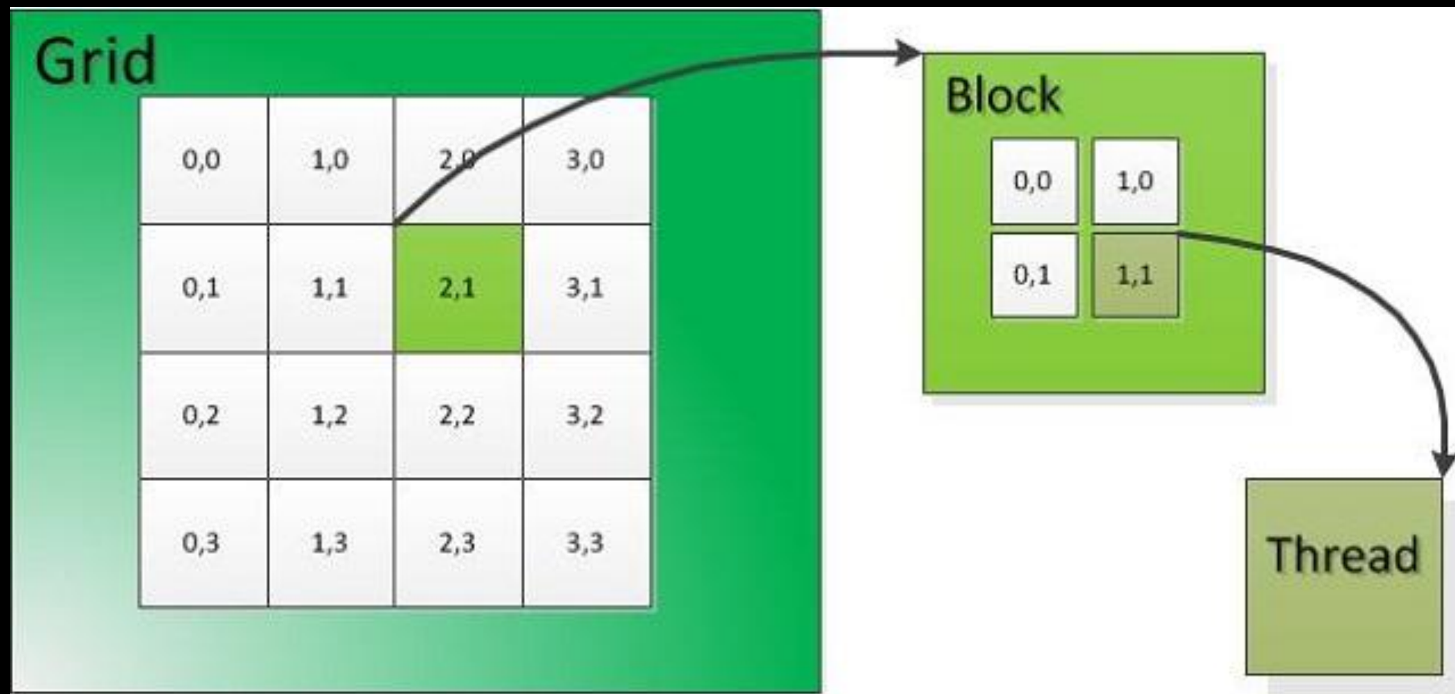
1. Antecedentes
2. El GPU
3. Modelo de programación CUDA
4. Manejo de matrices
5. Memoria compartida

Ariel Ulloa Trejo
Jaime Beltrán Rosales

3 Modelo de programación CUDA

Jerarquía de la memoria

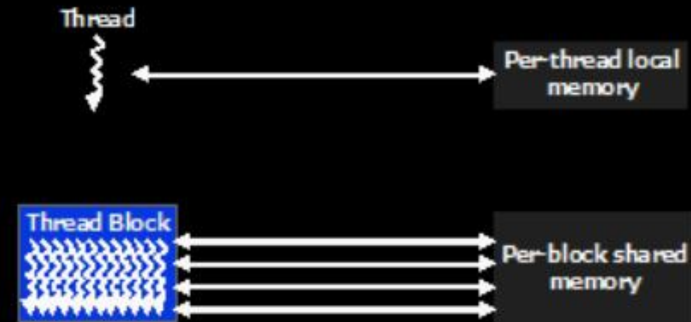
- Una malla (o grid) está conformada por bloques.
- Los bloques a su vez están conformados por hilos.



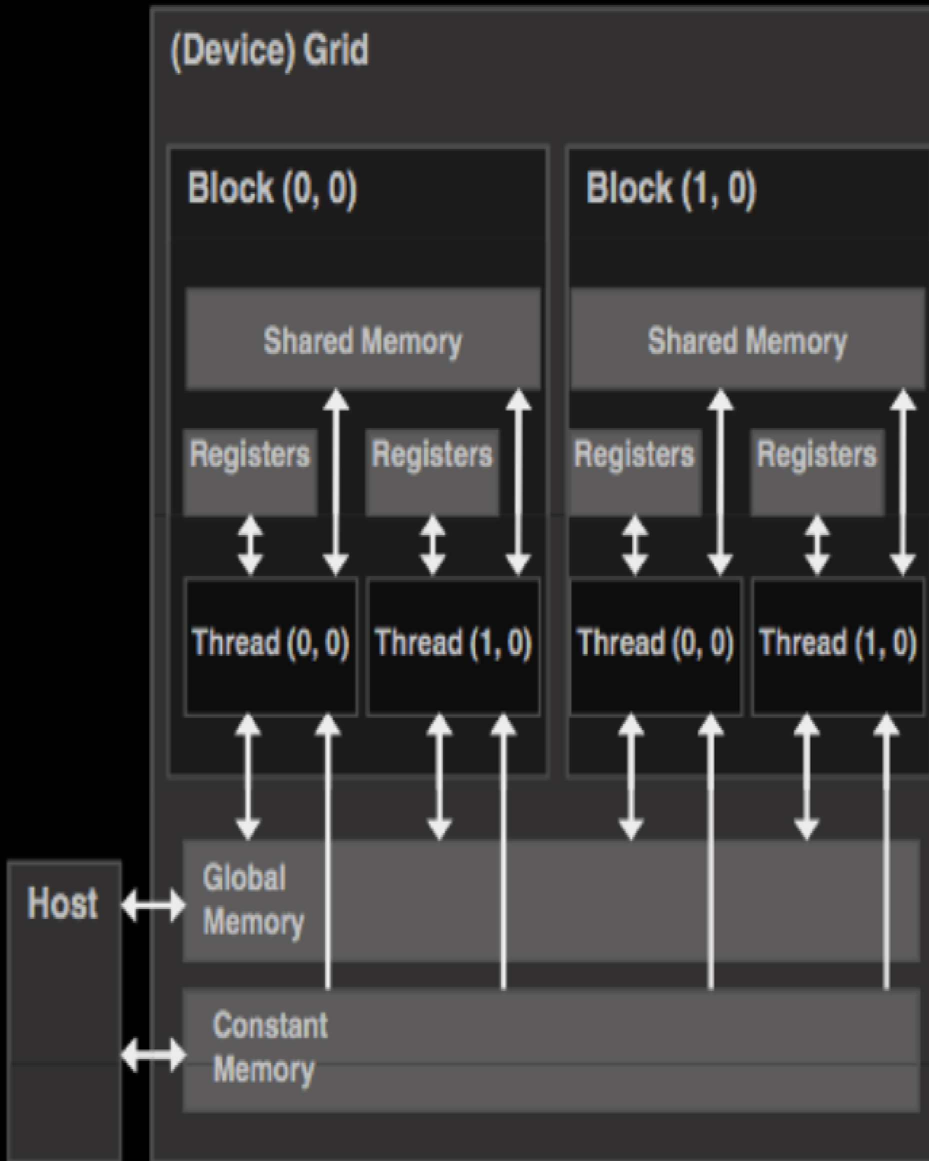
3 Modelo de programación CUDA

Jerarquía de la memoria

- Cada hilo tiene memoria local privada.
- Cada bloque tiene memoria compartida visible a todos los hilos del mismo.
- Todos los hilos tienen acceso a la misma memoria global.



Ariel Ulloa Trejo
Jaime Beltrán Rosales



El dispositivo puede:

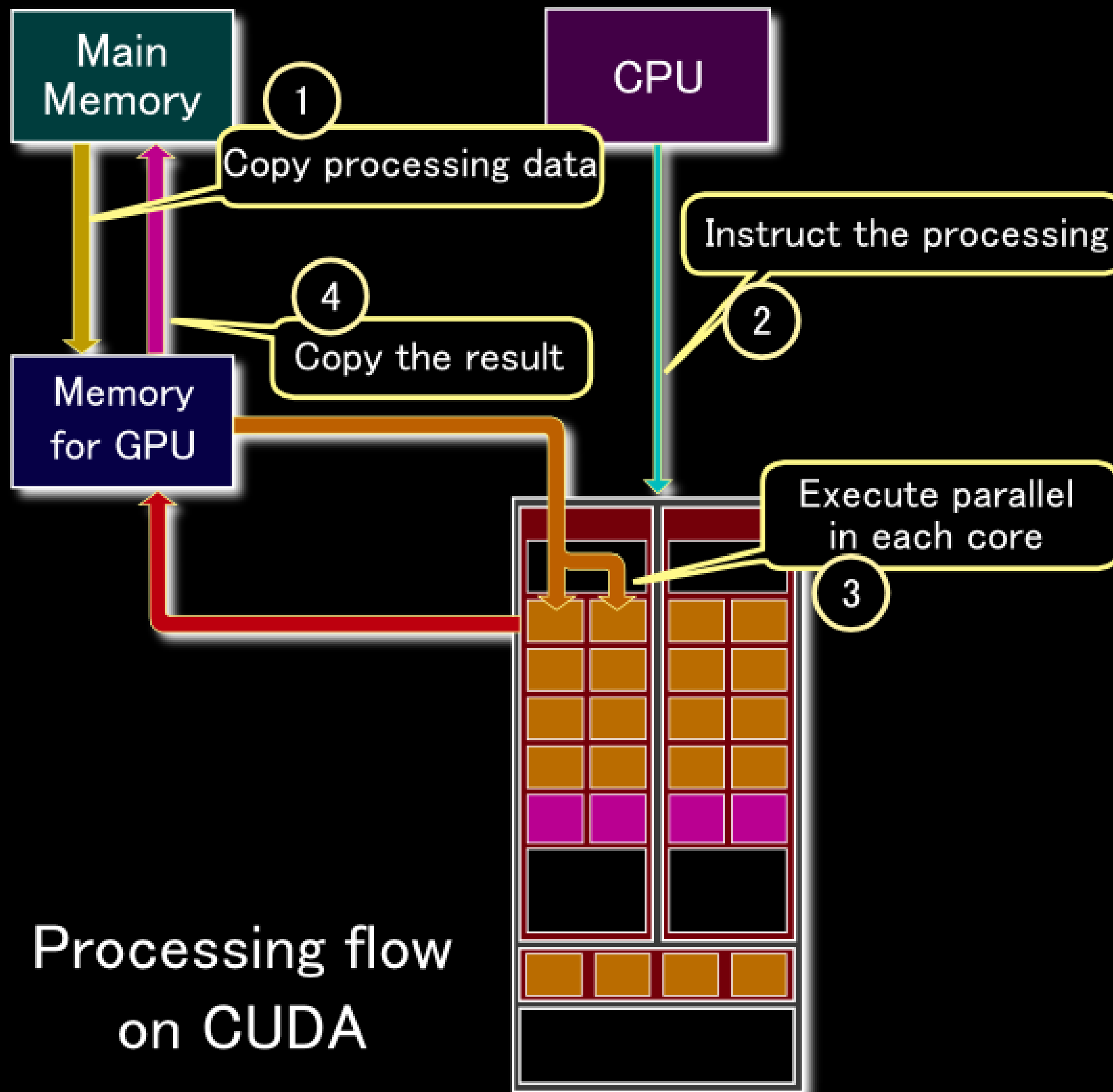
- L/E registros por hilo.
- L/E memoria local por hilo.
- L/E memoria compartida por bloque.
- L/E memoria global por grid.
- Leer memoria constante por grid.

El host (CPU) puede:

- Transferir datos a la memoria global y constante y viceversa.

Ariel Ulloa Trejo
Jaime Beltrán Rosales

3 Modelo de programación CUDA



1. Copiar Datos a procesar.
2. El CPU indica al GPU qué hay que hacer.
3. El código se ejecuta en paralelo.
4. Copiar resultado de GPU a CPU.

Ariel Ulloa Trejo
Jaime Beltrán Rosales

3 Modelo de programación CUDA

cudaMalloc()

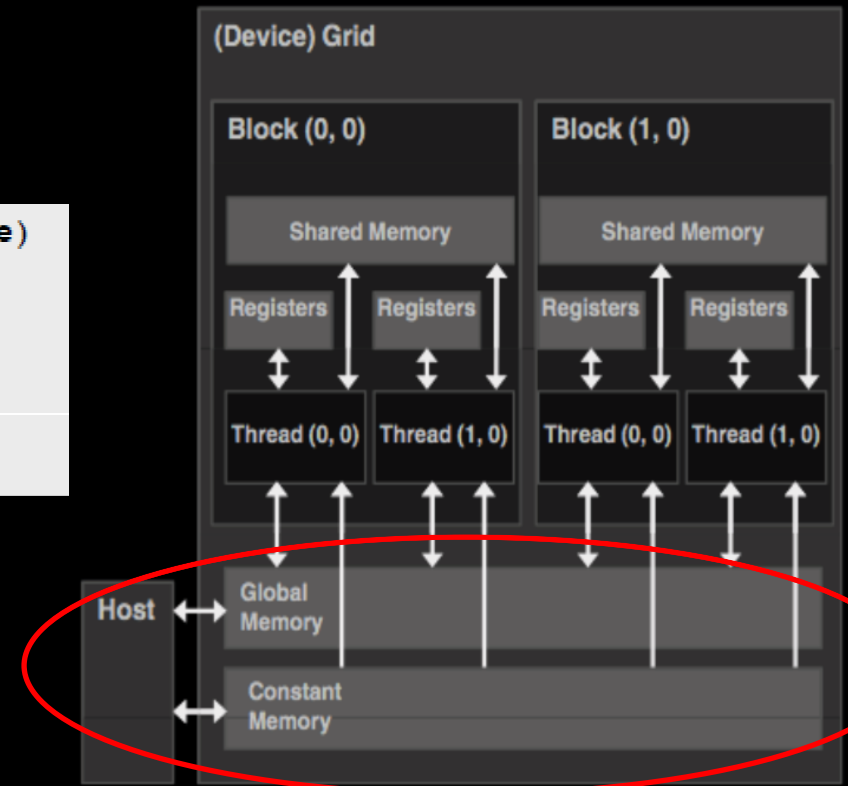
- Función necesaria para asignar memoria en el dispositivo.

cudaFree()

- Es necesario liberar la memoria que ya no se utiliza.

```
cudaError_t   cudaMalloc(void** devPtr, size_t size)
where:
devPtr - Pointer to allocated device memory
size – Requested allocation size in bytes
```

```
cudaError_t cudaFree (void * devPtr)
```



Ariel Ulloa Trejo
Jaime Beltrán Rosales

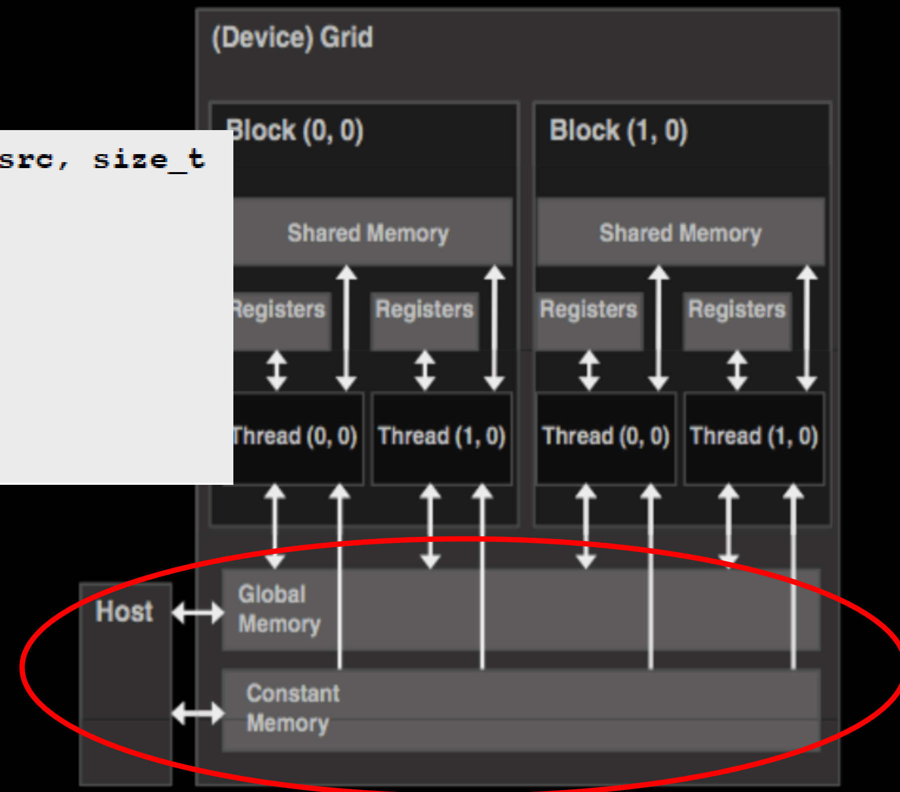
3 Modelo de programación CUDA

cudaMemcpy()

- Función que copia datos entre las memorias:

```
cudaError_t  cudaMemcpy( void * dst, const void * src, size_t
count, enumcudaMemcpyKind kind)
where:
dst - Destination memory address
src – Source memory address
count – Size in bytes to copy
kind – Type of transfer (cudamemcpyHostToDevice,
cudamemcpyDeviceToHost,)
```

- La transferencia es asíncrona (el programa continúa antes de que la copia sea finalizada)



Ariel Ulloa Trejo
Jaime Beltrán Rosales

3 Modelo de programación CUDA

Función Kernel

CUDA C/C++ es una extensión del lenguaje C que permite al programador definir funciones Kernel.

```
__global__ void addKernel( int a, int b, int *c ) {
    *c = a + b;
}

int main(){
    • • •
    addKernel<<<1,1>>>( 7, 6, device_a );
    • • •
    return 0;
}
```

Características

- Es ejecutada por N hilos.
- Cada hilo lo realiza de forma secuencial.
- Se debe especificar la declaración *__global__*
- El número de hilos se define utilizando **<<<…>>>**
- No tiene valor de retorno (void)

Ariel Ulloa Trejo
Jaime Beltrán Rosales

3 Modelo de programación CUDA

Ejercicio

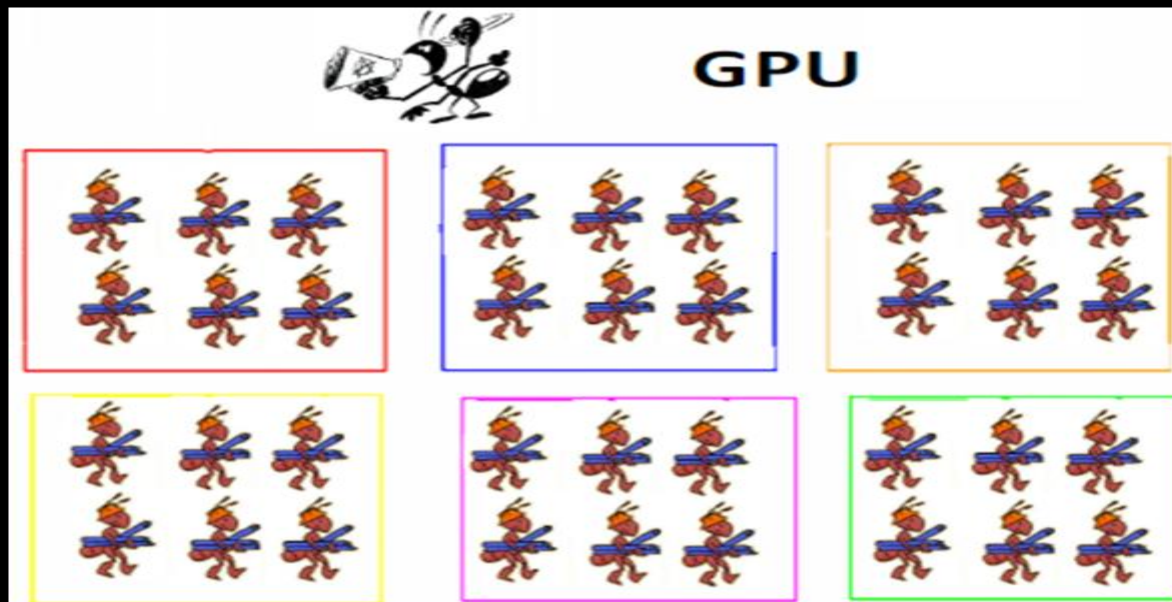
- Hacer un programa que realice la suma de dos enteros.

Ariel Ulloa Trejo
Jaime Beltrán Rosales

3 Modelo de programación CUDA

Configuración de la ejecución:

- La ejecución opera en paralelo a través de hilos.
- Cada hilo tiene su identificador y registros.
- La cantidad de hilos en ejecución depende del hardware.
- **Un grid está compuesto por bloques, y éstos por hilos.**



Ariel Ulloa Trejo
Jaime Beltrán Rosales

3 Modelo de programación CUDA

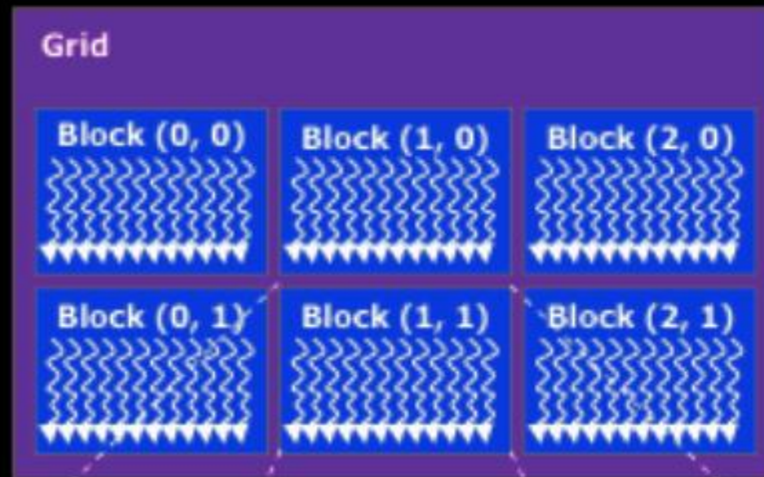
Índices:

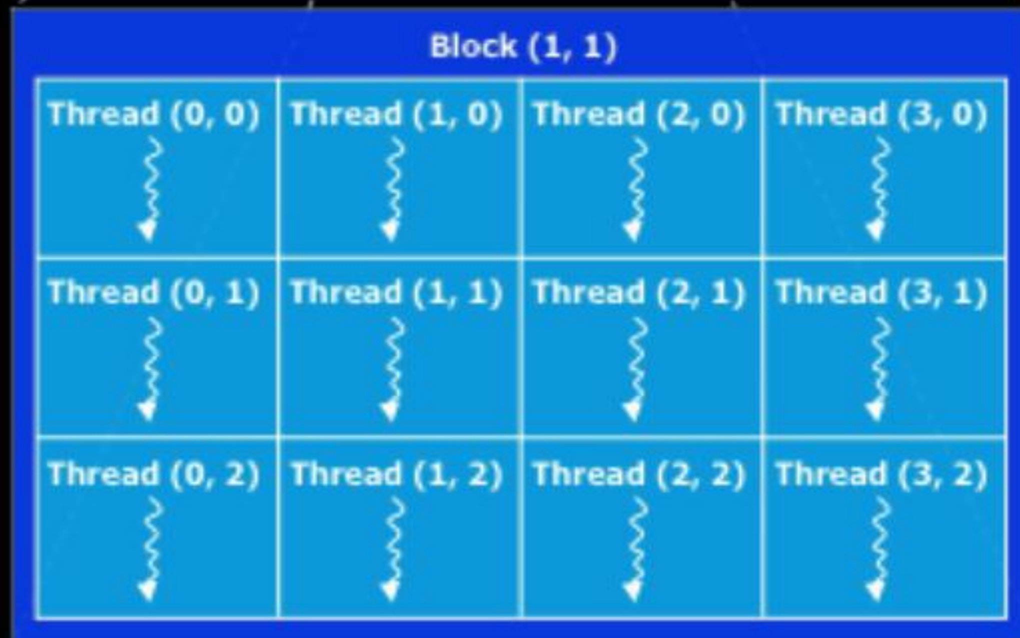
Hilos:

- Todos los hilos tienen índices con el fin de calcular las direcciones de memoria y tomas decisiones de control.
- Los hilos tienen variables para identificarse, ya sea en 1, 2 ó 3 dimensiones (lo que proporciona la facilidad para el manejo de vectores, matrices o volúmenes).

Bloques:

- Tal como los hilos, los bloques (dentro del grid) también tienen variables en 1, 2 ó 3 dimensiones.
- El kernel es copiado en todos los bloques "invocados".
- El número total de hilos es la cantidad de hilos de cada bloque por la cantidad de bloques.





Ariel Ulloa Trejo
Jaime Beltrán Rosales

3 Modelo de programación CUDA

Variables (creadas automáticamente):

- Índice del hilo en x → threadIdx.x
- Índice del hilo en y → threadIdx.y
- Índice del hilo en z → threadIdx.z

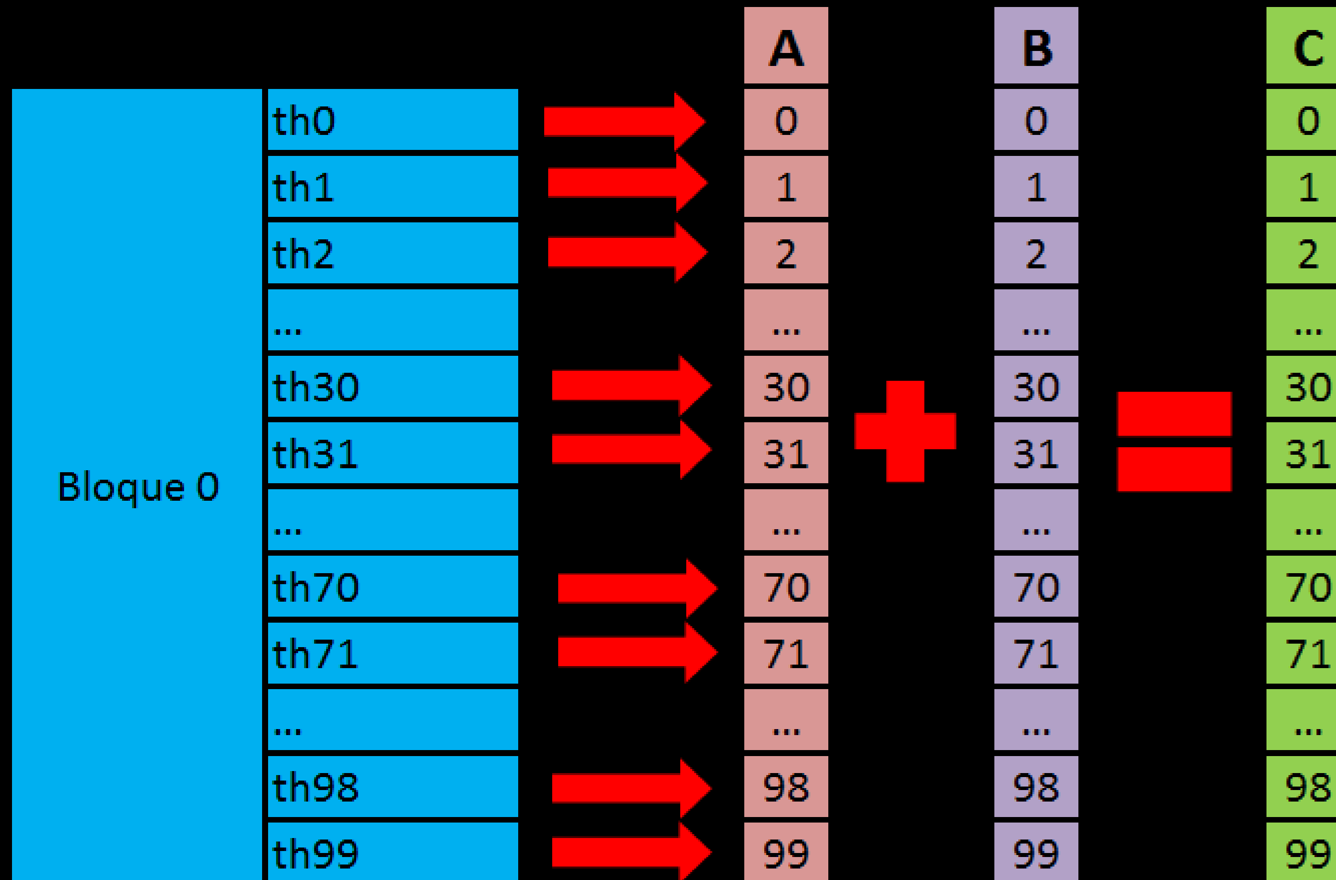
- Índice del bloque en x → blockIdx.x
- Índice del bloque en y → blockIdx.y
- Índice del bloque en z → blockIdx.z

- Número de hilos por bloque en x → blockDim.x
- Número de bloques por grid en x → gridDim.x

Ariel Ulloa Trejo
Jaime Beltrán Rosales

Ejercicio:

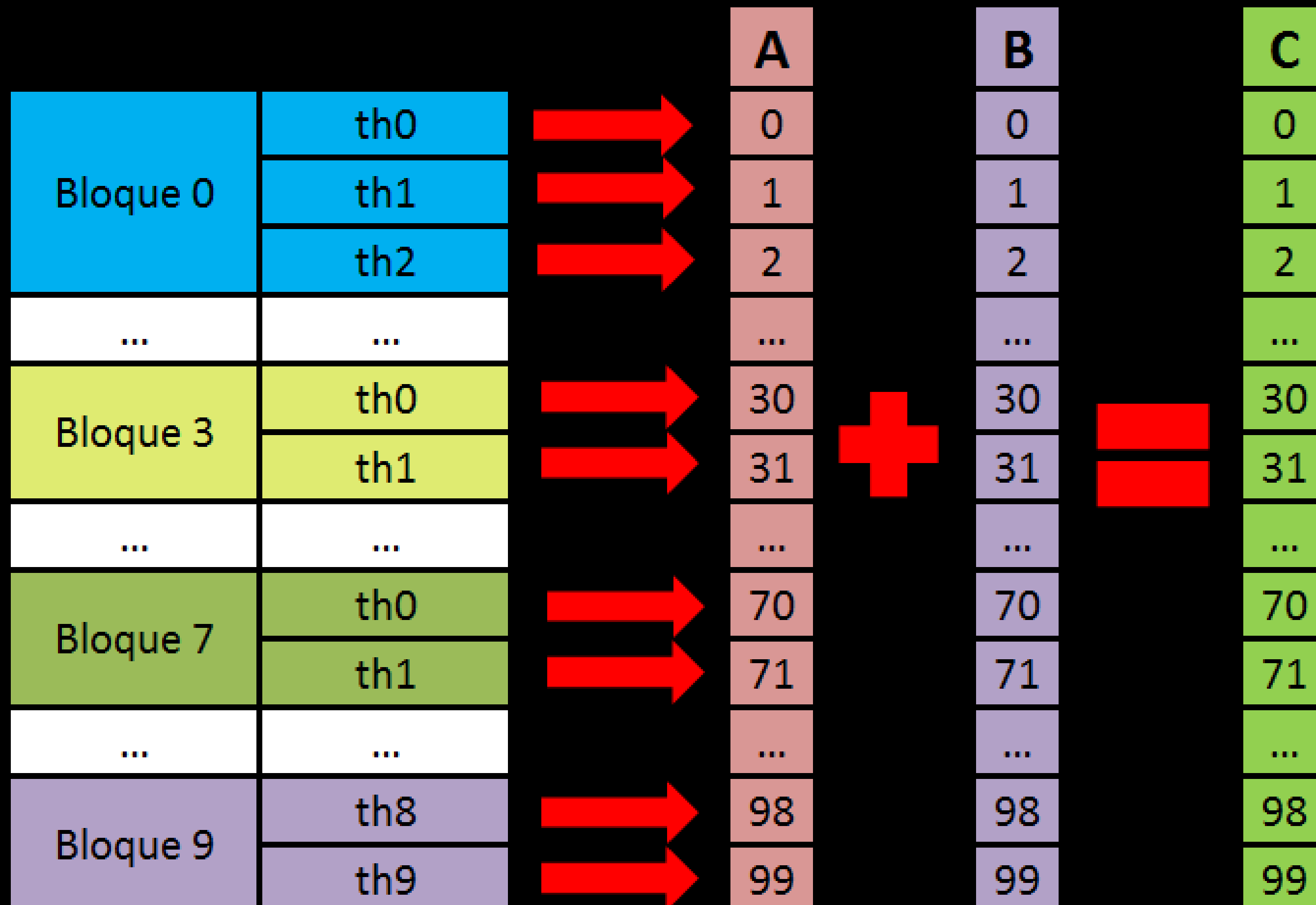
- Sumar dos vectores de tamaño 100 en el GPU usando 1 bloque y 100 hilos.



Ariel Ulloa Trejo
Jaime Beltrán Rosales

Ejercicio:

- Sumar dos vectores de tamaño 100 en el GPU usando 10 bloques y 10 hilos por bloque.



Ariel Ulloa Trejo
Jaime Beltrán Rosales

Solución:

- tid = threadIdx.x + blockDim.x * blockIdx.x;
- Cuyo mapeo es:

BLOCK 5	BLOCK 9
tid= 0 + 5* 10 = 50	tid= 0 + 9* 10 = 90
tid= 1 + 5* 10 = 51	tid= 1 + 9* 10 = 91
tid= 2 + 5* 10 = 52	tid= 2 + 9* 10 = 92
tid= 3 + 5* 10 = 53	tid= 3 + 9* 10 = 93
tid= 4 + 5* 10 = 54	tid= 4 + 9* 10 = 94
tid= 5 + 5* 10 = 55	tid= 5 + 9* 10 = 95
tid= 6 + 5* 10 = 56	tid= 6 + 9* 10 = 96
tid= 7 + 5* 10 = 57	tid= 7 + 9* 10 = 97
tid= 8 + 5* 10 = 58	tid= 8 + 9* 10 = 98
tid= 9 + 5* 10 = 59	tid= 9 + 9* 10 = 99

Arreglos unidimensionales

Ejercicio:

- Sumar dos vectores de tamaño 100 en el GPU usando 2 bloques y 10 hilos por bloque.

Bloque	Hilo
Bloque 0	th0
	...
	th9
Bloque 1	th0
	...
	th9

¿20 hilos para 100 elementos?