

Práctica 11. Suma de matrices N x M

$$\begin{bmatrix} 2 & 6 & 5 \\ -1 & 2 & 3 \\ 0 & 1 & 5 \\ 7 & 9 & 10 \end{bmatrix}_{4 \times 3} + \begin{bmatrix} 2 & 5 & 7 \\ 2 & 1 & -4 \\ 5 & 6 & 8 \\ 4 & -2 & 3 \end{bmatrix}_{4 \times 3} = \begin{bmatrix} 4 & 11 & 12 \\ 1 & 3 & -1 \\ 5 & 7 & 13 \\ 11 & 7 & 13 \end{bmatrix}_{4 \times 3}$$

U.A.Q. Fac. de Informática

Dra. Sandra Luz Canchola Magdaleno

Correo: sandra.canchola@uaq.mx

Dra. Reyna Moreno Beltrán

Correo: reyna.moreno@uaq.mx



Sean A y B matrices de dimensiones $n \times m$, la suma está definida como:

$$(A + B)_{i,j} = a_{i,j} + b_{i,j}$$

$$A_{n \times m} + B_{n \times m} = \begin{bmatrix} a_{11} + b_{11} & a_{12} + b_{12} & \dots & a_{1m} + b_{1m} \\ a_{21} + b_{21} & a_{22} + b_{22} & \dots & a_{2m} + b_{2m} \\ \dots & \dots & \dots & \dots \\ a_{n1} + b_{n1} & a_{n2} + b_{n2} & \dots & a_{nm} + b_{nm} \end{bmatrix}$$

Cálculo de suma de matrices $n \times m$

A

	0	1	2	...	m-1
0					
1					
2					
...					
n-1					

B

	0	1	2	...	m-1
0					
1					
2					
...					
n-1					

A+B

	0	1	2	...	m-1
0					
1					
2					
...					
n-1					

Elementos de matrices

Matriz 5 x 2

i	j	
	0	1
0		
1		
2		
3		✓
4		

Índice de elementos

`matriz[i][j]`

Ejemplo:

`mat1[3][1]`

Matriz 5 x 2 como apuntador a una memoria consecutiva de 10 elementos

0	1	2	3	4	5	6	7	8	9
							✓		

Índice de elementos

$\text{indice} = (i * \text{numCol}) + j$

Ejemplo:

`mat1[3][1]`

$\text{indice} = (3 * 2) + 1 = 7$

Elementos de matrices

Matriz 5 x 2 x 4

		j							
		0				1			
i	k=0	1	2	3	0	1	2	3	
	0								
	0	1	2	3	0	1	2	3	
	1						✓		
	0	1	2	3	0	1	2	3	
	2								
	0	1	2	3	0	1	2	3	
3									
0	1	2	3	0	1	2	3		
4									

Índice de elementos

`matriz[i][j][k]`

Ejemplo:

`mat2[1][1][2]`

Elementos de matrices

Matriz 5 x 2 x 4 como apuntador a una memoria consecutiva de 40 elementos

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	...	36	37	38	39
														✓		...				

Índice de elementos

$$\text{indice} = (i * \text{numCol} * \text{numProf}) + (j * \text{numProf}) + k$$

Ejemplo:

`mat2[1][1][2]`

$$\text{indice} = (1 * 2 * 4) + (1 * 4) + 2 = 14$$

Proyecto CUDA



Create a new project

Choose a project template with code scaffolding to get started



CUDA 11.7 Runtime

A project that uses the CUDA 11.7 runtime

C++

CUDA

Windows

Linux

Cloud

Console

DataScience

Desktop

Machine Learning



CUDA 12.1 Runtime

A project that uses the CUDA 12.1 runtime

C++

CUDA

Windows

Linux

Cloud

Console

DataScience

Desktop

Machine Learning

Proyecto CUDA

Configure your new project

CUDA 12.1 Runtime

C++

CUDA

Windows

Linux

Cloud

Console

DataScience

Desktop

Machine Learning

Project name

Prog11_SumaMatNxM

Location

C:\TrabajoLaboratorio\CUDATopico2\Projects\Verano2024

...

Solution

Create new solution

Solution name ⓘ

Prog11_SumaMatNxM

☐

Place solution and project in the same directory

Project will be created in "C:\TrabajoLaboratorio\CUDATopico2\Projects\Verano2024\Prog11_SumaMatNxM\Prog11_SumaMatNxM"

Back

Create

Operaciones de memoria (CPU)

- **malloc.**- Reserva un bloque de memoria de un tamaño definido de bytes, retornando un apuntador al inicio de dicho bloque. El contenido de dicho bloque no se inicializa por lo que es indeterminado. Ejemplo:

```
void* malloc (size_t size);  
buffer = (char*) malloc (sizeof(char)*100);
```

- **memset.**- asigna valores en secciones de memoria. Ejemplo:
Memset(variable, valor_a_asignar, tamaño_de_memoria)
Donde: tamaño_de_memoria se define como n * sizeof(tipo)

- **memcpy.**- Copia el contenido de un bloque de memoria referenciado por un apuntador a otro apuntador. Ejemplo:

```
void* memcpy( void* dest, const void* src, std::size_t count );  
memcpy (ptrDest, ptrOrigen, sizeof(int)*100);
```

- **free.**- Liberar la memoria reservada con el comando malloc. Ejemplo:
free(pointerName);
free(array2);

Operaciones de memoria (GPU)

- **cudaMalloc**.- asigna una sección de memoria en GPU de acuerdo con el espacio solicitado.

Ejemplo:

```
cudaMalloc((void**) &apuntador, tamaño_de_memoria)
```

Donde: tamaño_de_memoria se define como $n * \text{sizeof}(\text{tipo})$

- **cudaMemset**.- asigna valores en secciones de memoria.

Ejemplo:

```
Memset(apuntador, valor_a_asignar, tamaño_de_memoria)
```

Donde: tamaño_de_memoria se define como $n * \text{sizeof}(\text{tipo})$

- **cudaMemcpy**.- copia memoria hacia y desde el device.

Ejemplo:

```
cudaMemcpy(destino, origen, tamaño_de_memoria, indicador_flujo_de_inf)
```

Donde Indicador= cudaMemcpyHostToDevice, cudaMemcpyDeviceToHost, cudaMemcpyDeviceToDevice

- **cudaFree**.- libera la memoria reservada por un apuntador.

Ejemplo:

```
cudaFree (apuntador)
```

Memoria

CPU (Host)

A01	widthN	50									
A03	widthM	60									
A05	epsilon	0.000001									
A07	maxN	16									
A10	maxM	20									
A15	A	0	1	2	3	4	5	6	7	...	(n×m) - 1
		α	γ	φ	φ	η	χ	λ	ε	...	τ
B02	B	0	1	2	3	4	5	6	7	...	(n×m) - 1
		τ	β	κ	θ	π	δ	ε	υ	...	η
B45	C	0	1	2	3	4	5	6	7	...	(n×m) - 1
		α + τ	γ + β	φ + κ	φ + θ	η + π	χ + δ	λ + ε	ε + υ	...	τ + η
C30	C_host	0	1	2	3	4	5	6	7	...	(n×m) - 1
		α + τ	γ + β	φ + κ	φ + θ	η + π	χ + δ	λ + ε	ε + υ	...	τ + η
E10	dev_A	J10									
F20	dev_B	J45									
G05	dev_C	J90									
H16											
H20											

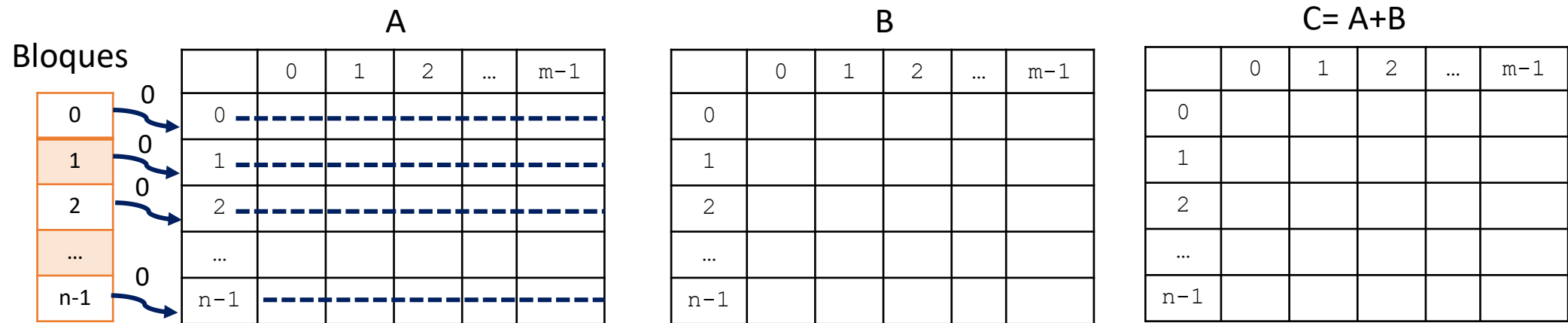
GPU (Device)

J01											
J05											
J10		0	1	2	3	4	5	6	7	...	(n×m) - 1
		α	γ	φ	φ	η	χ	λ	ε	...	τ
J45		0	1	2	3	4	5	6	7	...	(n×m) - 1
		τ	β	κ	θ	π	δ	ε	υ	...	η
J90		0	1	2	3	4	5	6	7	...	(n×m) - 1
		α + τ	γ + β	φ + κ	φ + θ	η + π	χ + δ	λ + ε	ε + υ	...	τ + η
K01											
K05											
K10											
K15											
K20											
K30											
L07											
L10											

Memoria reservada

Apuntador

Caso 1. N bloques con hilo único.
Cada hilo calcula el resultado de una fila completa.



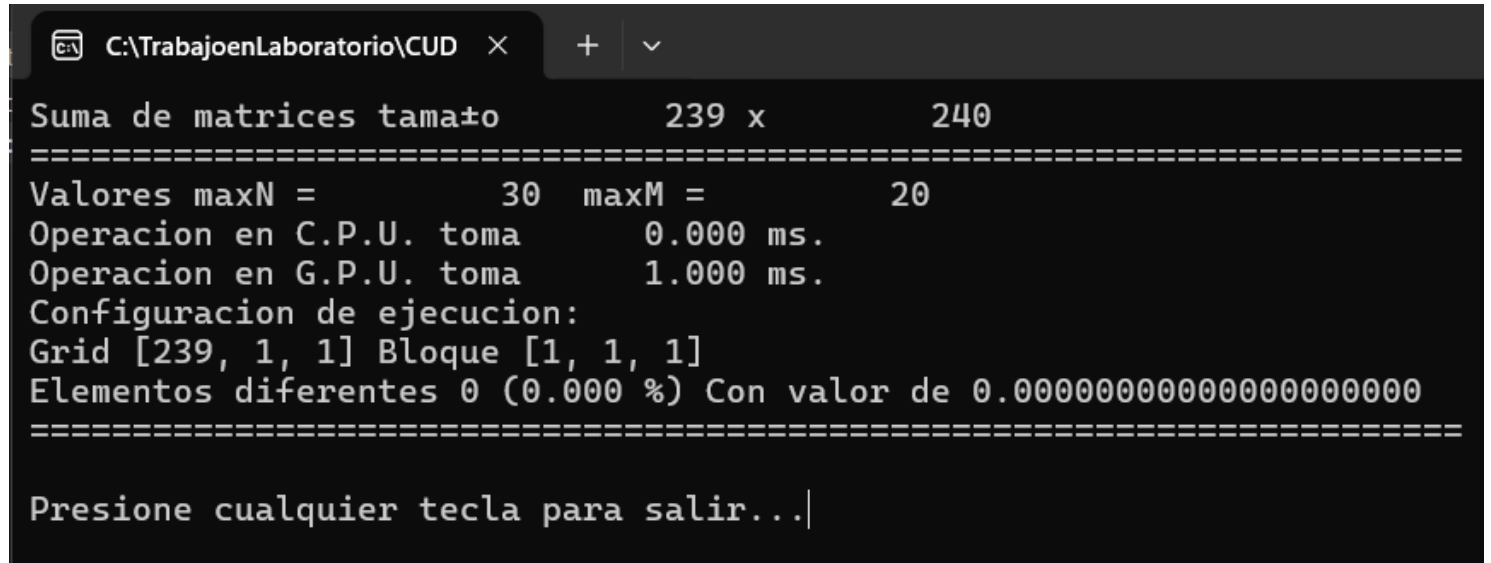
$tid = blockIdx.x$
 $primerElem = (blockIdx.x * n)$

blockIdx.x	threadIdx.x	tid	Elementos atendidos	
0	0	0	$C_{0,0}, C_{0,1}, C_{0,2} \dots C_{0,m-1}$	$0, 1, 2, \dots (m-1)$
1	0	1	$C_{1,0}, C_{1,1}, C_{1,2} \dots C_{1,m-1}$	$m, m+1, m+2, \dots, 2m-1$
2	0	2	$C_{2,0}, C_{2,1}, C_{2,2} \dots C_{2,m-1}$	$2m, 2m+1, 2m+2, \dots, 3m-1$
...
n-1	0	n-1	$C_{n-1,0}, C_{n-1,1}, C_{n-1,2} \dots C_{n-1,m-1}$	$(n-1)m, (n-1)m+1, (n-1)m+2, \dots, n * m - 1$

Caso 1. N bloques con hilo único.

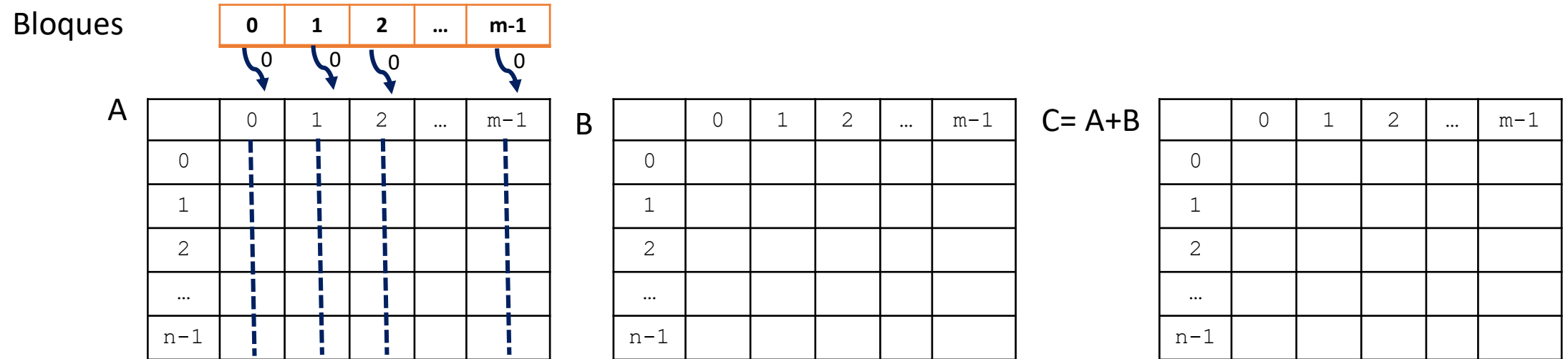
Cada hilo calcula el resultado de una fila completa.

```
#define widthN 239
#define widthM 240
#define epsilon float(0.0000001)
...
dim3 dimGrid(widthN);
dim3 dimBlock(1);
...
int fila = blockIdx.x;
int tid = fila * widthM;
for (int i = 0; i < widthM; i++){
    c[tid + i] = a[tid + i] + b[tid + i];
}
```



```
C:\TrabajoLaboratorio\CUD x + v
Suma de matrices tamaño      239 x      240
=====
Valores maxN =          30  maxM =          20
Operacion en C.P.U. toma      0.000 ms.
Operacion en G.P.U. toma      1.000 ms.
Configuracion de ejecucion:
Grid [239, 1, 1] Bloque [1, 1, 1]
Elementos diferentes 0 (0.000 %) Con valor de 0.00000000000000000000
=====
Presione cualquier tecla para salir...|
```

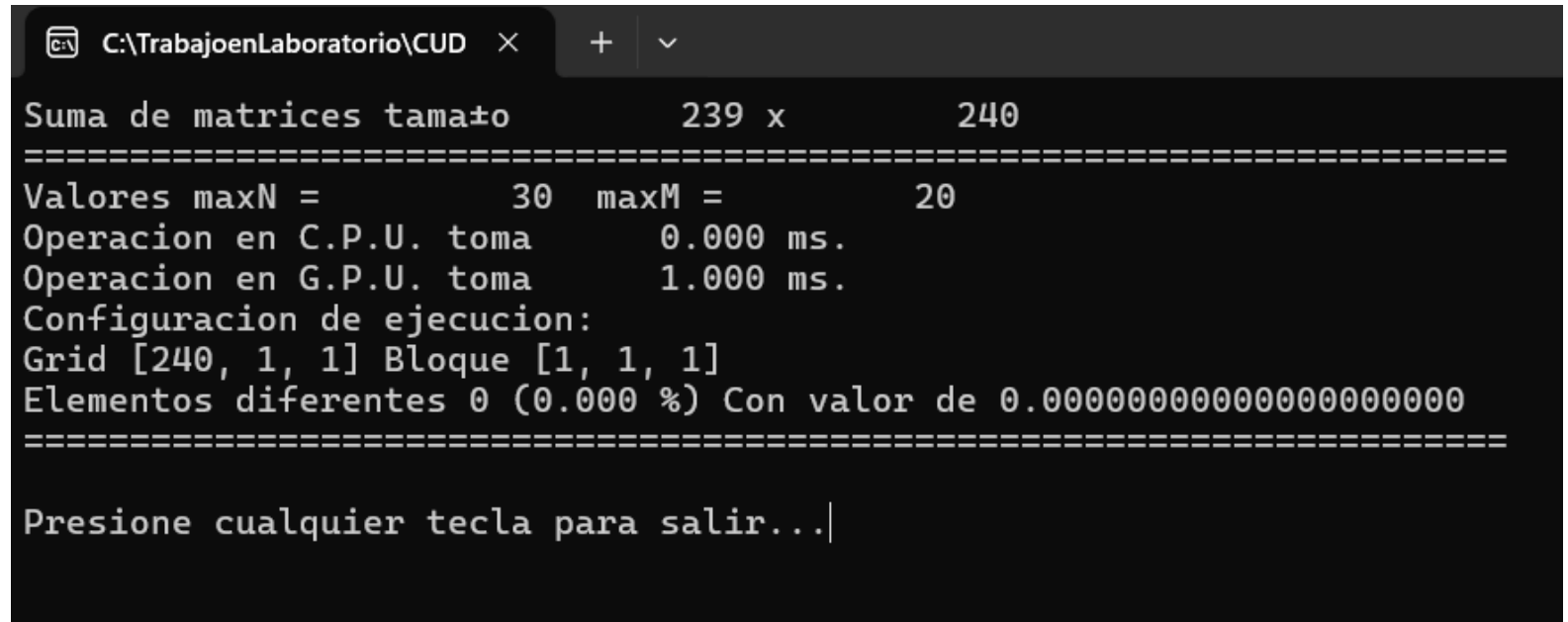
Caso 2. N bloques con hilo único.
Cada hilo calcula el resultado de una columna completa.



tid=blockIdx.x
primerElem=blockIdx.x

Caso 2. N bloques con hilo único.
Cada hilo calcula el resultado de una columna completa.

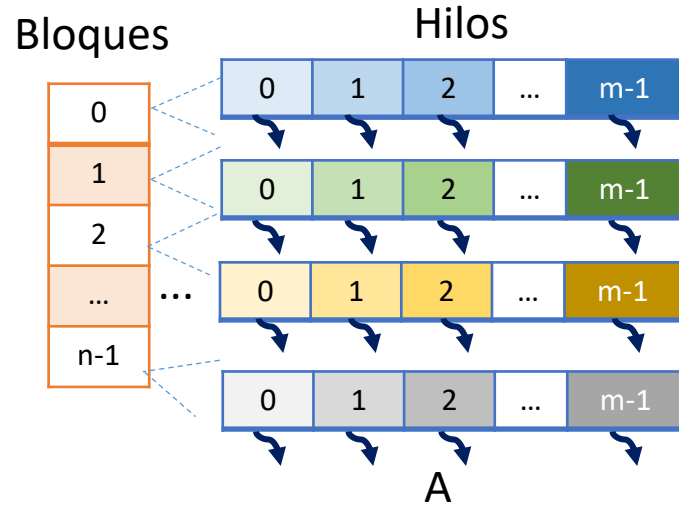
```
#define widthN 239
#define widthM 240
#define epsilon float(0.0000001)
...
dim3 dimGrid(widthM);
dim3 dimBlock(1);
...
int tid = blockIdx.x;
for (int i = 0; i < widthN; i++){
    c[tid + (i * widthM)] = a[tid + (i * widthM)] + b[tid + (i * widthM)];
}
```



The screenshot shows a terminal window titled "C:\TrabajoLaboratorio\CUD" with a dark background and light-colored text. The output of the program is as follows:

```
Suma de matrices tamaño      239 x      240
=====
Valores maxN =          30  maxM =          20
Operacion en C.P.U. toma      0.000 ms.
Operacion en G.P.U. toma      1.000 ms.
Configuracion de ejecucion:
Grid [240, 1, 1] Bloque [1, 1, 1]
Elementos diferentes 0 (0.000 %) Con valor de 0.000000000000000000000000
=====
Presione cualquier tecla para salir...|
```


Caso 3. N bloques con M hilos cada uno.



	0	1	2	...	m-1
0					
1					
2					
...					
n-1					

B

	0	1	2	...	m-1
0					
1					
2					
...					
n-1					

C= A+B

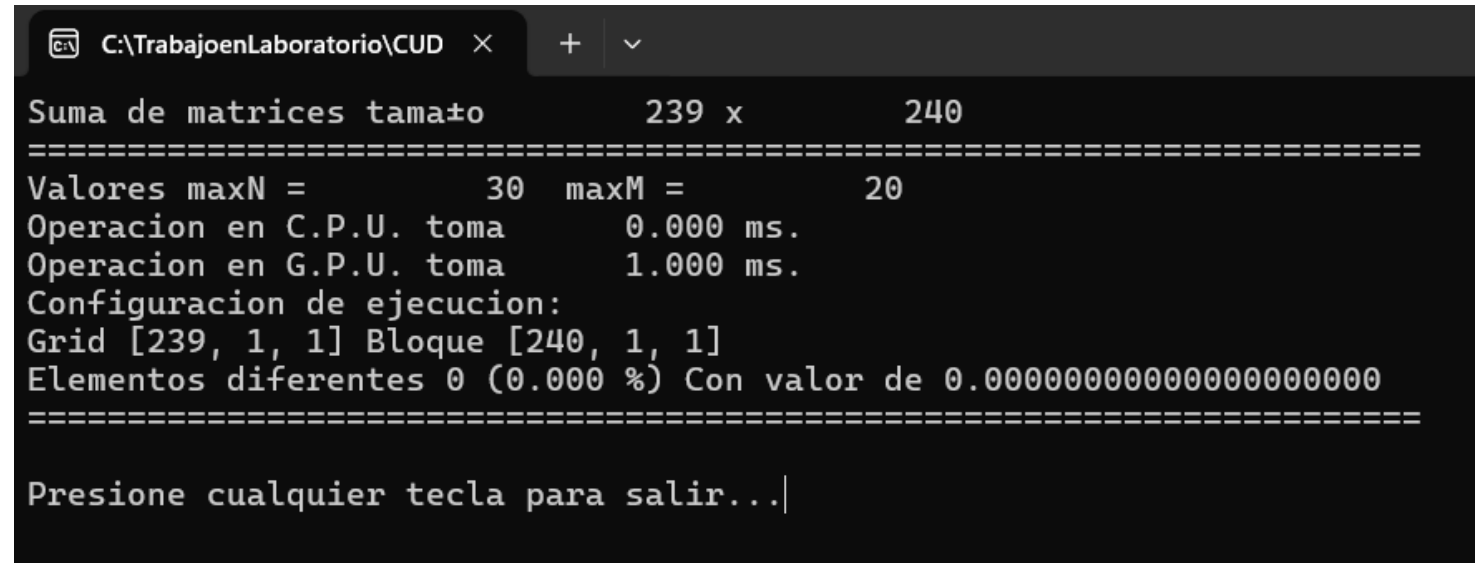
	0	1	2	...	m-1
0					
1					
2					
...					
n-1					

$tid = (blockIdx.x * blockDim.x) + threadIdx.x$

blockIdx.x	threadIdx.x	tid	Elementos atendidos	
0	0	0	$C_{0,0}, C_{1,0}, C_{2,0} \dots C_{n-1,0}$	$0, m, 2m, \dots, (n-1)m$
1	0	1	$C_{0,1}, C_{1,1}, C_{2,1} \dots C_{n-1,1}$	$1, m+1, 2m+1, \dots, (n-1)m+1$
2	0	2	$C_{0,2}, C_{1,2}, C_{2,2} \dots C_{n-1,2}$	$2, m+2, 2m+2, \dots, (n-1)m+2$
...
n-1	0	n-1	$C_{0m-1}, C_{1,m-1}, C_{2,m-1} \dots C_{n-1,m-1}$	$m-1, 2m-1, \dots, (n \times m) - 1$

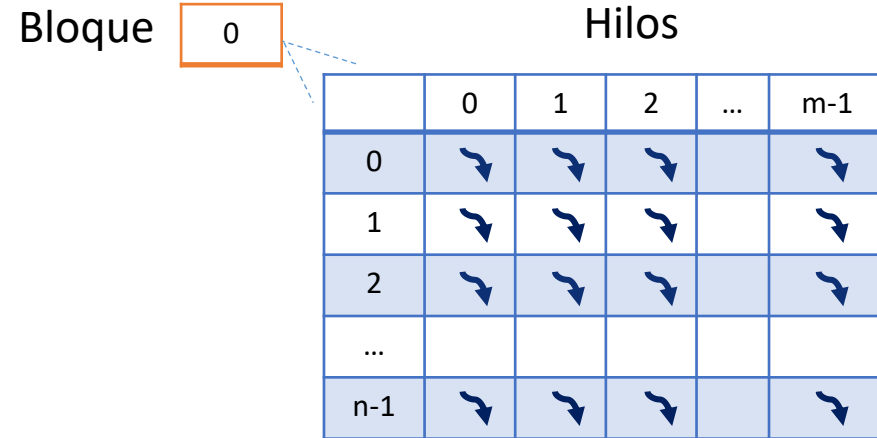
Caso 3. N bloques con M hilos cada uno.

```
#define widthN 239
#define widthM 240
#define epsilon float(0.0000001)
...
dim3 dimGrid(widthN); // por cada renglon
dim3 dimBlock(widthM); // por cada columna
...
int tid = (blockIdx.x*blockDim.x)+threadIdx.x;
c[tid] = a[tid] + b[tid];
```



```
C:\TrabajoLaboratorio\CUD x + v
Suma de matrices tamaño      239 x      240
=====
Valores maxN =      30  maxM =      20
Operacion en C.P.U. toma      0.000 ms.
Operacion en G.P.U. toma      1.000 ms.
Configuracion de ejecucion:
Grid [239, 1, 1] Bloque [240, 1, 1]
Elementos diferentes 0 (0.000 %) Con valor de 0.000000000000000000000000
=====
Presione cualquier tecla para salir...|
```

Caso 4. Un bloque con NxM hilos



A

	0	1	2	...	m-1
0					
1					
2					
...					
n-1					

B

	0	1	2	...	m-1
0					
1					
2					
...					
n-1					

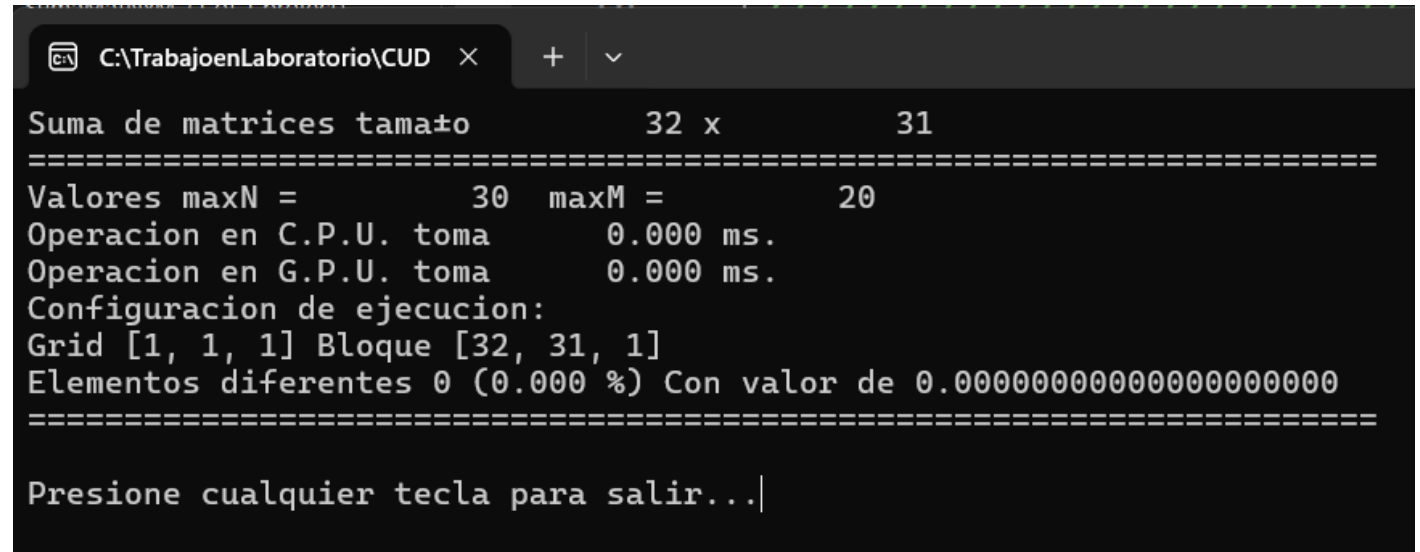
C= A+B

	0	1	2	...	m-1
0					
1					
2					
...					
n-1					

$tid = (threadIdx.x * blockDim.y) + threadIdx.y$

Caso 4. Un bloque con NxM hilos













```
#define widthN 32
#define widthM 31
#define epsilon float(0.0000001)
...
dim3 dimGrid(1);
dim3 dimBlock(widthN, widthM); // por cada elemento
...
int tid = (threadIdx.x*blockDim.y) + threadIdx.y;
c[tid] = a[tid] + b[tid];
```



```
C:\TrabajoLaboratorio\CUD x + v
Suma de matrices tamaño 32 x 31
=====
Valores maxN = 30 maxM = 20
Operacion en C.P.U. toma 0.000 ms.
Operacion en G.P.U. toma 0.000 ms.
Configuracion de ejecucion:
Grid [1, 1, 1] Bloque [32, 31, 1]
Elementos diferentes 0 (0.000 %) Con valor de 0.00000000000000000000
=====
Presione cualquier tecla para salir...|
```

Caso 5. Bloque 2D con hilos 2D

Hilos

	0	1	2
0			
1			
2			
3			

Tamaño de la matriz
13 x 10

maxN=4, maxM=3

Cada bloque tiene $\text{maxN} \times \text{maxM}$ hilos.

Dimensión del grid

Primera dimensión

$$N / \max N = 13 / 4 = 3.25 \approx 4$$

Segunda dimensión

$$M / \max M = 10 / 3 = 3.3333 \approx 4$$

A

[illegible]

Caso 5. Bloque 2D con hilos 2D

Bloque

	0	1	2	3
0				
1				
2				
3				

Hilos

	0	1	2
0			
1			
2			
3			

maxN=4, maxM=3

Cada bloque tiene maxN x maxM hilos.

$\text{filaInicialBloque} = (\text{blockIdx.x} * \text{blockDim.x})$

$\text{fila} = \text{filaInicialBloque} + \text{threadIdx.x}$

$\text{columnaIniciaBloque} = (\text{blockIdx.y} * \text{blockDim.y})$

$\text{columna} = \text{columnaIniciaBloque} + \text{threadIdx.y}$

A

	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	10	11	12	13	14	15	16	17	18	19
2	20	21	22	23	24	25	26	27	28	29
3	30	31	32	33	34	35	36	37	38	39
4	40	41	42	43	44	45	46	47	48	49
5	50	51	52	53	54	55	56	57	58	59
6	60	61	62	63	64	65	66	67	68	69
7	70	71	72	73	74	75	76	77	78	79
8	80	81	82	83	84	85	86	87	88	89
9	90	91	92	93	94	95	96	97	98	99
10	100	101	102	103	104	105	106	107	108	109
11	110	111	112	113	114	115	116	117	118	119
12	120	121	122	123	124	125	126	127	128	129

blockI dx		thread Idx		Elemento		
x	y	x	y	fila	col	#
0	0	0	0	0	0	0
0	0	0	1	0	1	1
0	0	0	2	0	2	2
0	0	1	0	1	0	10
0	0	1	1	1	1	11
0	0	1	2	1	2	12
0	0	2	0	2	0	20
0	0	2	1	2	1	21
0	0	2	2	2	2	22
0	0	3	0	3	0	30
0	0	3	1	3	1	31
0	0	3	2	3	2	32
0	1	0	0	0	3	3
0	1	0	1	0	4	4
0	1	0	2	0	5	5
0	1	1	0	1	3	13
0	1	1	1	1	4	14
0	1	1	2	1	5	15
0	1	2	0	2	3	23
0	1	2	1	2	4	24
0	1	2	2	2	5	25
0	1	3	0	3	3	33
0	1	3	1	3	4	34
0	1	3	2	3	5	35

blockI dx		thread Idx		Elemento		
x	y	x	y	fila	col	#
0	2	0	0	0	6	6
0	2	0	1	0	7	7
0	2	0	2	0	8	8
0	2	1	0	1	6	16
0	2	1	1	1	7	17
0	2	1	2	1	8	18
0	2	2	0	2	6	26
0	2	2	1	2	7	27
0	2	2	2	2	8	28
0	2	3	0	3	6	36
0	2	3	1	3	7	37
0	2	3	2	3	8	38
0	3	0	0	0	9	9
0	3	0	1	0	10	10
0	3	0	2	0	11	11
0	3	1	0	1	9	19
0	3	1	1	1	10	20
0	3	1	2	1	11	21
0	3	2	0	2	9	29
0	3	2	1	2	10	30
0	3	2	2	2	11	31
0	3	3	0	3	9	39
0	3	3	1	3	10	40
0	3	3	2	3	11	41

blockI dx		thread Idx		Elemento		
x	y	x	y	fila	col	#
1	0	0	0	4	0	40
1	0	0	1	4	1	41
1	0	0	2	4	2	42
1	0	1	0	5	0	50
1	0	1	1	5	1	51
1	0	1	2	5	2	52
1	0	2	0	6	0	60
1	0	2	1	6	1	61
1	0	2	2	6	2	62
1	0	3	0	7	0	70
1	0	3	1	7	1	71
1	0	3	2	7	2	72
1	1	0	0	4	3	43
1	1	0	1	4	4	44
1	1	0	2	4	5	45
1	1	1	0	5	3	53
1	1	1	1	5	4	54
1	1	1	2	5	5	55
1	1	2	0	6	3	63
1	1	2	1	6	4	64
1	1	2	2	6	5	65
1	1	3	0	7	3	73
1	1	3	1	7	4	74
1	1	3	2	7	5	75

blockI dx		thread Idx		Elemento		
x	y	x	y	fila	col	#
1	2	0	0	4	6	46
1	2	0	1	4	7	47
1	2	0	2	4	8	48
1	2	1	0	5	6	56
1	2	1	1	5	7	57
1	2	1	2	5	8	58
1	2	2	0	6	6	66
1	2	2	1	6	7	67
1	2	2	2	6	8	68
1	2	3	0	7	6	76
1	2	3	1	7	7	77
1	2	3	2	7	8	78
1	3	0	0	4	9	49
1	3	0	1	4	10	50
1	3	0	2	4	11	51
1	3	1	0	5	9	59
1	3	1	1	5	10	60
1	3	1	2	5	11	61
1	3	2	0	6	9	69
1	3	2	1	6	10	70
1	3	2	2	6	11	71
1	3	3	0	7	9	79
1	3	3	1	7	10	80
1	3	3	2	7	11	81

blockI dx		thread Idx		Elemento		
x	y	x	y	fila	col	#
2	0	0	0	8	0	80
2	0	0	1	8	1	81
2	0	0	2	8	2	82
2	0	1	0	9	0	90
2	0	1	1	9	1	91
2	0	1	2	9	2	92
2	0	2	0	10	0	100
2	0	2	1	10	1	101
2	0	2	2	10	2	102
2	0	3	0	11	0	110
2	0	3	1	11	1	111
2	0	3	2	11	2	112
2	1	0	0	8	3	83
2	1	0	1	8	4	84
2	1	0	2	8	5	85
2	1	1	0	9	3	93
2	1	1	1	9	4	94
2	1	1	2	9	5	95
2	1	2	0	10	3	103
2	1	2	1	10	4	104
2	1	2	2	10	5	105
2	1	3	0	11	3	113
2	1	3	1	11	4	114
2	1	3	2	11	5	115

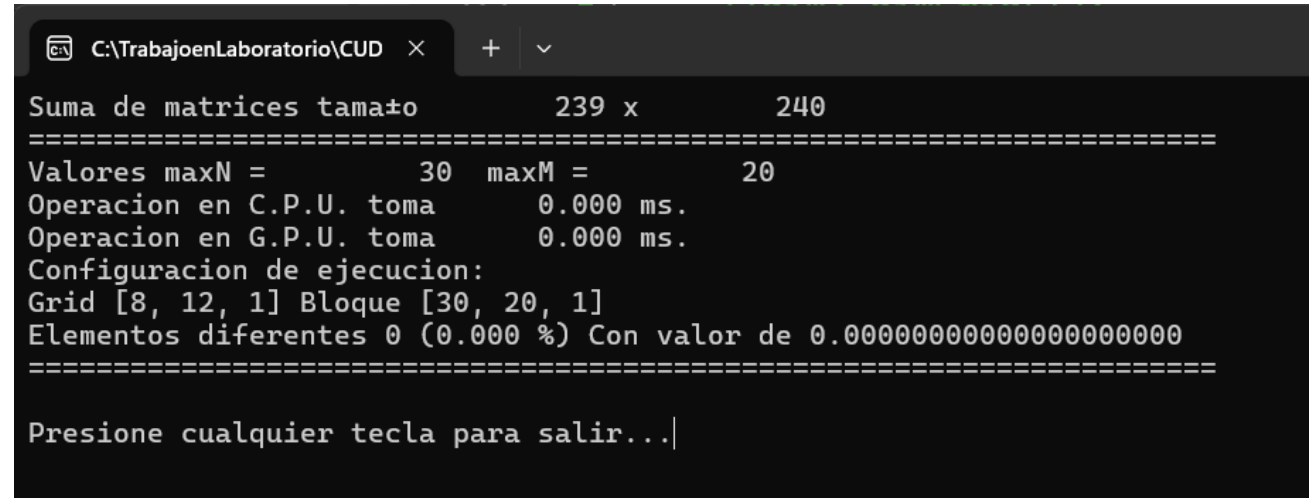
blockI dx		thread Idx		Elemento		
x	y	x	y	fila	col	#
2	2	0	0	8	6	86
2	2	0	1	8	7	87
2	2	0	2	8	8	88
2	2	1	0	9	6	96
2	2	1	1	9	7	97
2	2	1	2	9	8	98
2	2	2	0	10	6	106
2	2	2	1	10	7	107
2	2	2	2	10	8	108
2	2	3	0	11	6	116
2	2	3	1	11	7	117
2	2	3	2	11	8	118
2	3	0	0	8	9	89
2	3	0	1	8	10	90
2	3	0	2	8	11	91
2	3	1	0	9	9	99
2	3	1	1	9	10	100
2	3	1	2	9	11	101
2	3	2	0	10	9	109
2	3	2	1	10	10	110
2	3	2	2	10	11	111
2	3	3	0	11	9	119
2	3	3	1	11	10	120
2	3	3	2	11	11	121

blockI dx		thread Idx		Elemento		
x	y	x	y	fila	col	#
3	0	0	0	12	0	120
2	0	0	1	12	1	121
2	0	0	2	12	2	122
2	0	1	0	13	0	130
2	0	1	1	13	1	131
2	0	1	2	13	2	132
2	0	2	0	14	0	140
2	0	2	1	14	1	141
2	0	2	2	14	2	142
2	0	3	0	15	0	150
2	0	3	1	15	1	151
2	0	3	2	15	2	152
3	1	0	0	12	3	123
2	1	0	1	12	4	124
2	1	0	2	12	5	125
2	1	1	0	13	3	133
2	1	1	1	13	4	134
2	1	1	2	13	5	135
2	1	2	0	14	3	143
2	1	2	1	14	4	144
2	1	2	2	14	5	145
2	1	3	0	15	3	153
2	1	3	1	15	4	154
2	1	3	2	15	5	155

blockI dx		thread Idx		Elemento		
x	y	x	y	fila	col	#
3	2	0	0	12	6	126
2	2	0	1	12	7	127
2	2	0	2	12	8	128
2	2	1	0	13	6	136
2	2	1	1	13	7	137
2	2	1	2	13	8	138
2	2	2	0	14	6	146
2	2	2	1	14	7	147
2	2	2	2	14	8	148
2	2	3	0	15	6	156
2	2	3	1	15	7	157
2	2	3	2	15	8	158
3	3	0	0	12	9	129
2	3	0	1	12	10	130
2	3	0	2	12	11	131
2	3	1	0	13	9	139
2	3	1	1	13	10	140
2	3	1	2	13	11	141
2	3	2	0	14	9	149
2	3	2	1	14	10	150
2	3	2	2	14	11	151
2	3	3	0	15	9	159
2	3	3	1	15	10	160
2	3	3	2	15	11	161

Caso 5. Bloque 2D con hilos 2D

```
#define widthN 239
#define widthM 240
#define epsilon float(0.0000001)
#define maxN 30
#define maxM 20
...
int numBloquesN = divEntera(widthN , maxN);
int numBloquesM = divEntera(widthM , maxM);
dim3 dimGrid(numBloquesN, numBloquesM);
dim3 dimBlock(maxN, maxM);
...
int fila = (blockIdx.x * blockDim.x) + threadIdx.x;
int columna = (blockIdx.y * blockDim.y) + threadIdx.y;
if ((fila < widthN) && (columna < widthM)) {
    int tid = (fila * widthM) + columna;
    c[tid] = a[tid] + b[tid];
}
```



```
C:\TrabajoLaboratorio\CUD x + v
Suma de matrices tamaño      239 x      240
=====
Valores maxN =          30  maxM =          20
Operacion en C.P.U. toma      0.000 ms.
Operacion en G.P.U. toma      0.000 ms.
Configuracion de ejecucion:
Grid [8, 12, 1] Bloque [30, 20, 1]
Elementos diferentes 0 (0.000 %) Con valor de 0.00000000000000000000
=====
Presione cualquier tecla para salir...|
```

Bibliografía

- Documentación **CUDA C++ Programming Guide** NVIDIA. 2024
<https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html>
- Sitio **CUDA Toolkit Documentation** NVIDIA, 2024.
<https://docs.nvidia.com/cuda/index.html>
- Storti, Duane; Yurtoglu, Mete. **CUDA for Engineers: An Introduction to High-Performance Parallel Computing**. Addison Wesley. 2015.
- Cheng, John; Grossman, Max; McKercher. **Professional CUDA C Programming**. Edit. Wrox. 2014.
- Sanders, Jason; Kandrot, Edward. **CUDA by Example: An Introduction to General-Purpose GPU Programming**. Addison Wesley. 2011.
- Kirk, David; Hwu, Wen-mei. **Programming Massively Parallel Processors: A Hands-on Approach**. Elsevier. 2010.

Gracias por su atención



**U.A.Q. Fac. de Informática
Campus Juriquilla**

**Dra. Sandra Luz Canchola Magdaleno
sandra.canchola@uaq.mx
Cel. 442-1369270**

**Dra. Reyna Moreno Beltrán
reyna.moreno@uaq.mx**