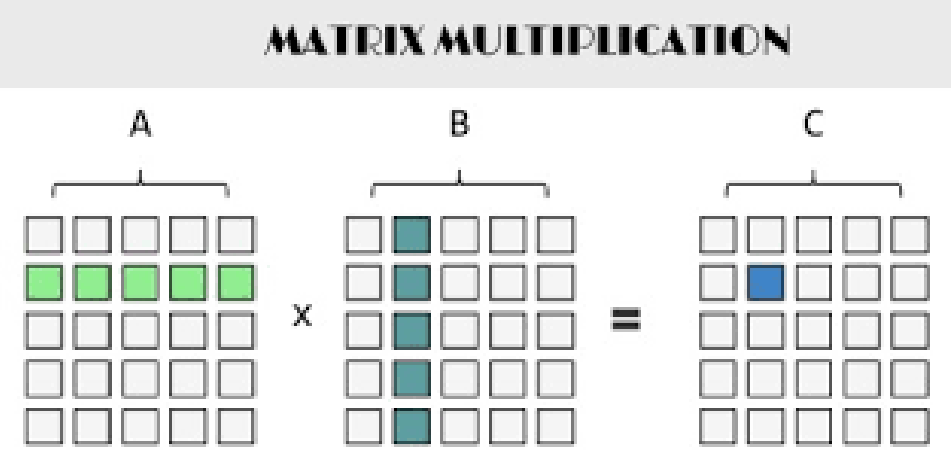
Práctica 12. Multiplicación de Matrices N x M y M x R



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 una matriz tamaño n x m y B una matriz de tamaño m x r, la multiplicación está definida como:

$$A_{n\times m} * B_{m\times r} = AB_{n\times r}$$

$$(AB)_{i,j} = \sum_{k=1}^{m} a_{i,k} + b_{k,j}$$

$$A_{3\times 3} = \begin{bmatrix} 1 & 5 & 2 \\ 0 & -2 & 3 \\ 4 & 1 & 0 \end{bmatrix} \qquad B_{3\times 4} = \begin{bmatrix} 2 & 1 & 3 & 1 \\ -1 & 2 & 0 & 1 \\ 3 & 4 & 0 & 5 \end{bmatrix}$$

$$AB_{3\times 4} = \begin{bmatrix} (1*2)+(5*-1)+(2*3) & (1*1)+(5*2)+(2*4) & (1*3)+(5*0)+(2*0) & (1*1)+(5*1)+(2*5) \\ (0*2)+(-2*-1)+(3*3) & (0*1)+(-2*2)+(3*4) & (0*3)+(-2*0)+(3*0) & (0*1)+(-2*1)+(3*5) \\ (4*2)+(1*-1)+(0*3) & (4*1)+(1*2)+(0*4) & (4*3)+(1*0)+(0*0) & (4*1)+(1*1)+(0*5) \end{bmatrix}$$

$$AB_{3\times 4} = \begin{bmatrix} 2-5+6 & 1+10+8 & 3 & 1+5+10 \\ 2+9 & -4+12 & 0 & -2+15 \\ 8-1 & 4+2 & 12 & 4+1 \end{bmatrix} = \begin{bmatrix} 3 & 19 & 3 & 16 \\ 11 & 8 & 0 & 13 \\ 7 & 6 & 12 & 5 \end{bmatrix}$$

Cálculo de multiplicación de matrices

A

	0	1	2	…	m-1
0					
1					
2					
…					
n-1					

B

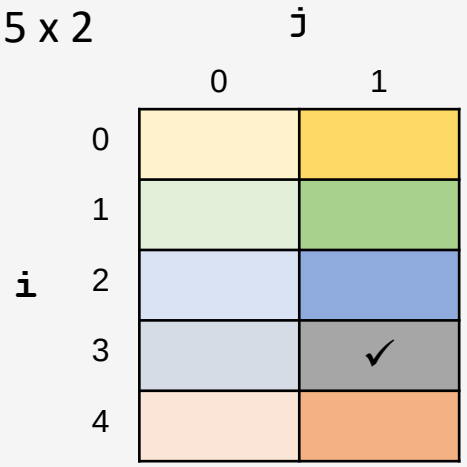
	0	1	2	…	r-1
0					
1					
2					
…					
m-1					

A+B

	0	1	2	…	r-1
0					
1					
2					
…					
n-1					

Elementos de matrices

Matriz 5 x 2

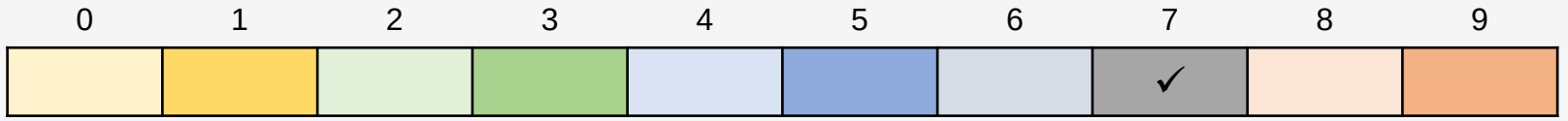


Índice de elementos

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

```
Ejemplo:
mat1[3][1]
```

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



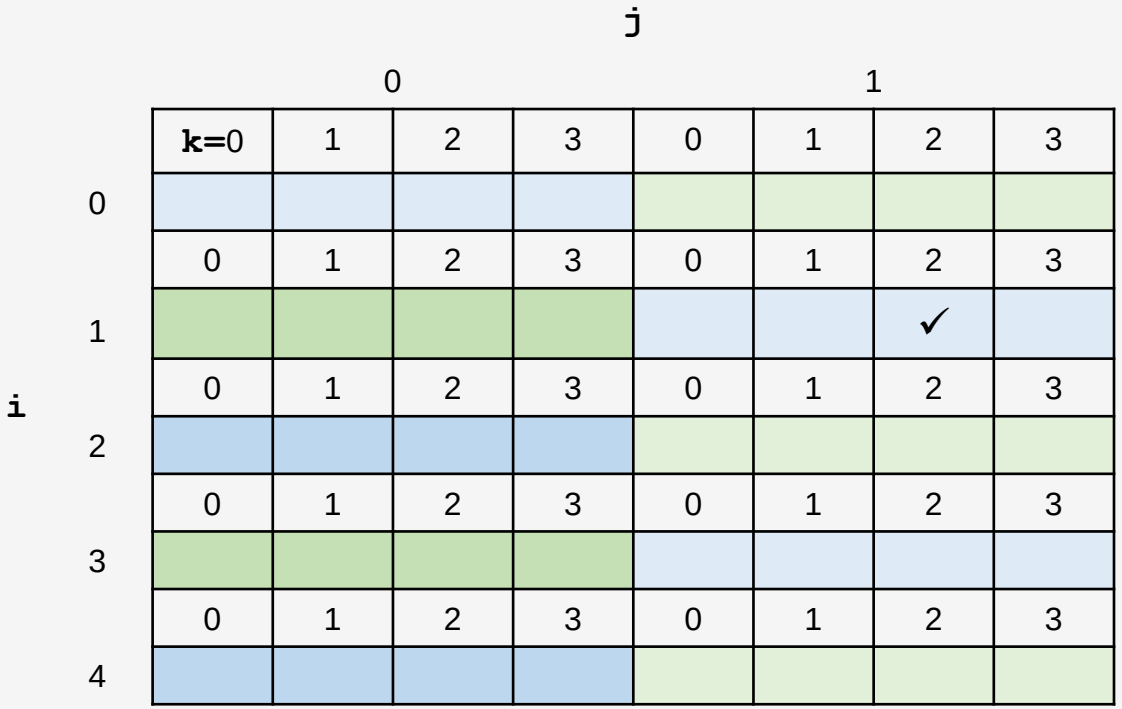
Índice de elementos

```
indice=(i*numCol)+j
```

```
Ejemplo:

mat1[3][1]
indice=(3*2)+1=7
```

Elementos de matrices

Matriz 5 x 2 x 4



Índice de elementos

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

Ejemplo:

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

Elementos de matrices

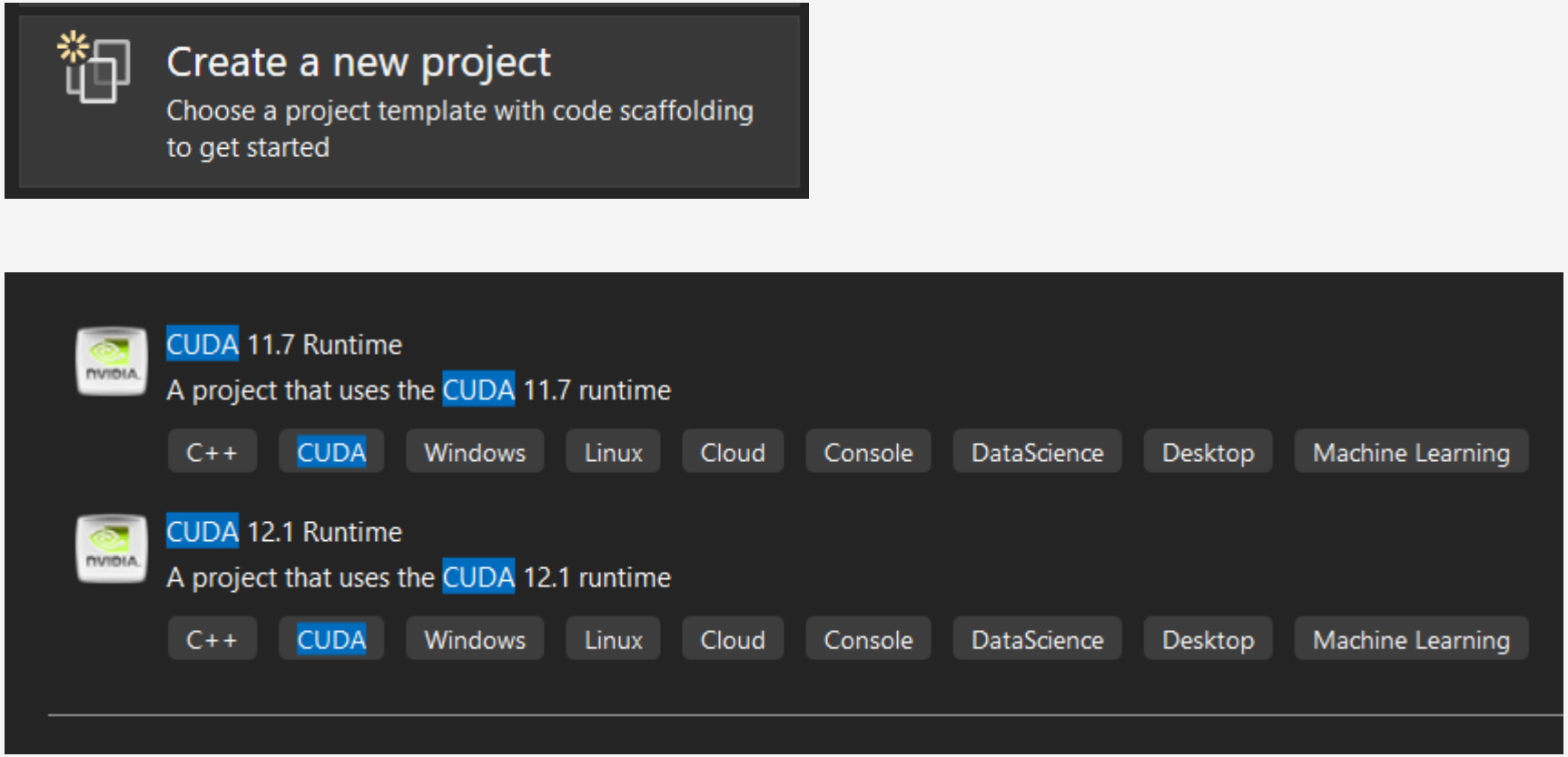
Matriz 5 x 2 x 4 como apuntador a una memoria consecutiva de 40 elementos

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

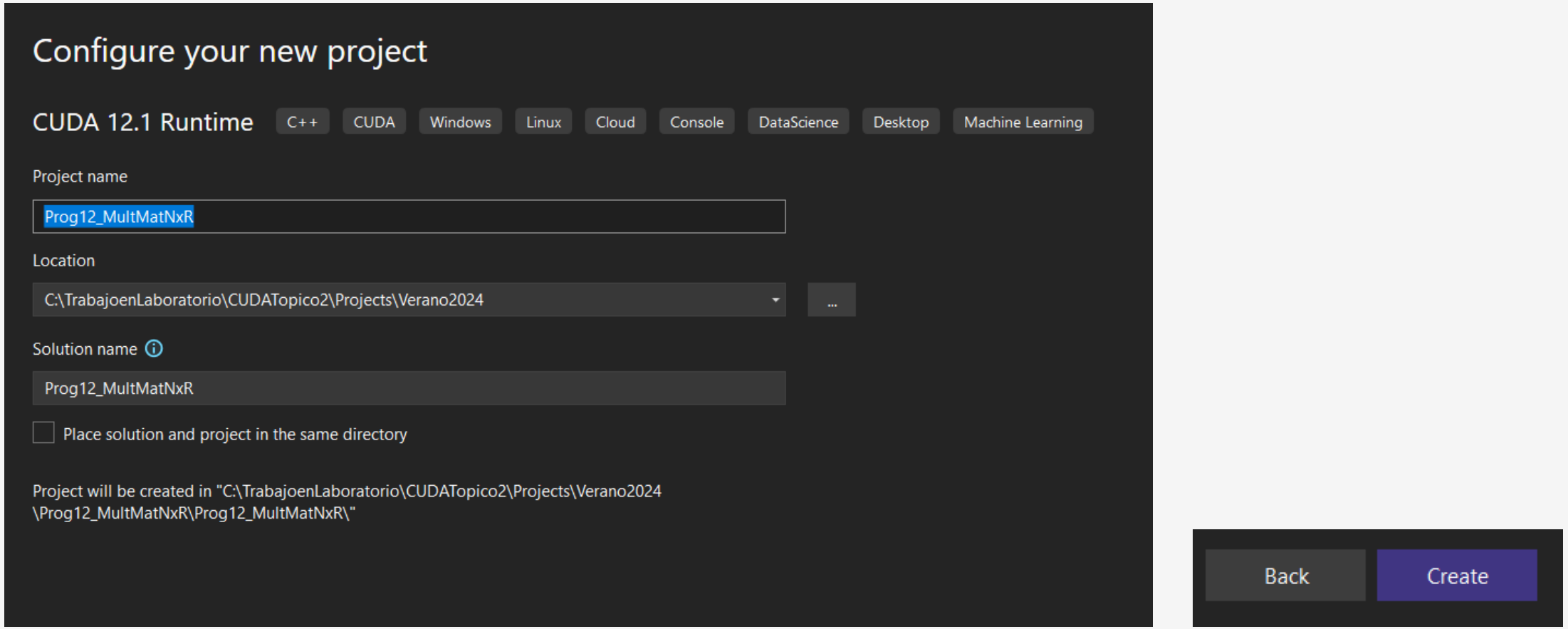
Índice de elementos

```
indice=(i*numCol*numProf)+(j*numProf)+k

Ejemplo:

mat2[1][1][2]
indice=(1*2*4)+(1*4)+2=14
```

Proyecto CUDA



Proyecto CUDA



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 * sizeof(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 * sizeof(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								
A04	widthR	25								
A05	epsilon	0.000001								
A07	maxN	16								
A10	maxM	20								
A15	A	0	1	2	3	4	5	6	7	…	(nxm)-1
		α	γ	ϕ	φ	η	χ	λ	ε	…	τ
B02	B	0	1	2	3	4	5	6	7	…	(mxr)-1
		τ	β	κ	θ	π	δ	ε	υ	…	η
B45	C	0	1	2	3	4	5	6	7	…	(nxr)-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	…	(nxr)-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											

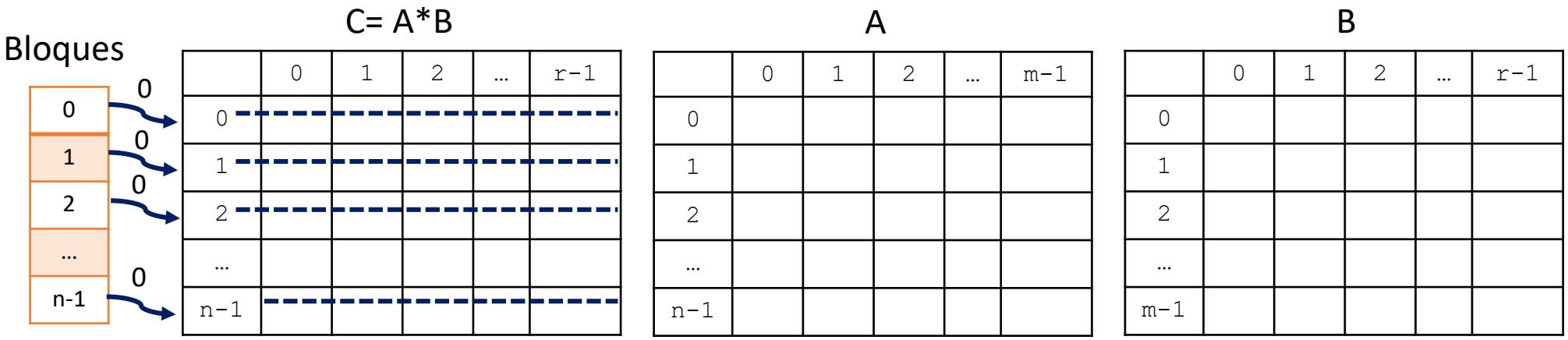
GPU (Device)

J01										
J05										
J10	0	1	2	3	4	5	6	7	…	(nxm)-1
	α	γ	ϕ	φ	η	χ	λ	ε	…	τ
J45	0	1	2	3	4	5	6	7	…	(mxr)-1
	τ	β	κ	θ	π	δ	ε	υ	…	η
J90	0	1	2	3	4	5	6	7	…	(nxr)-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.

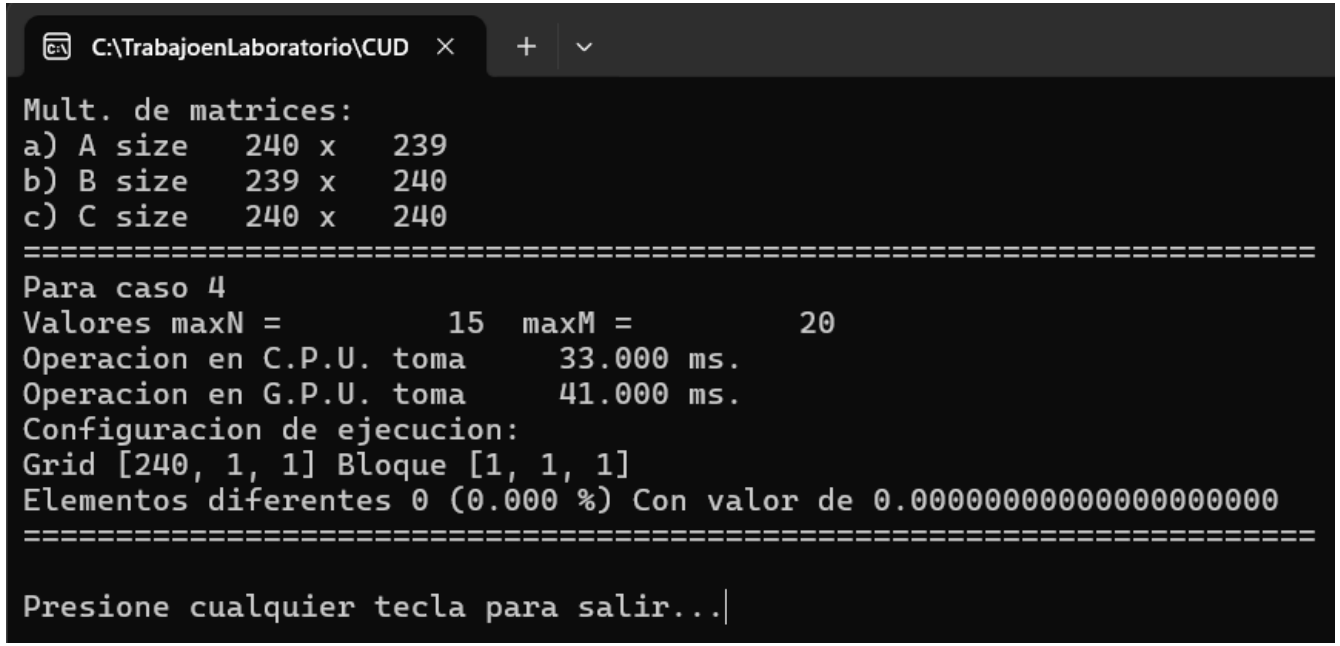


tid=blockIdx.x
primerElem=(blockIdx.x*withR)

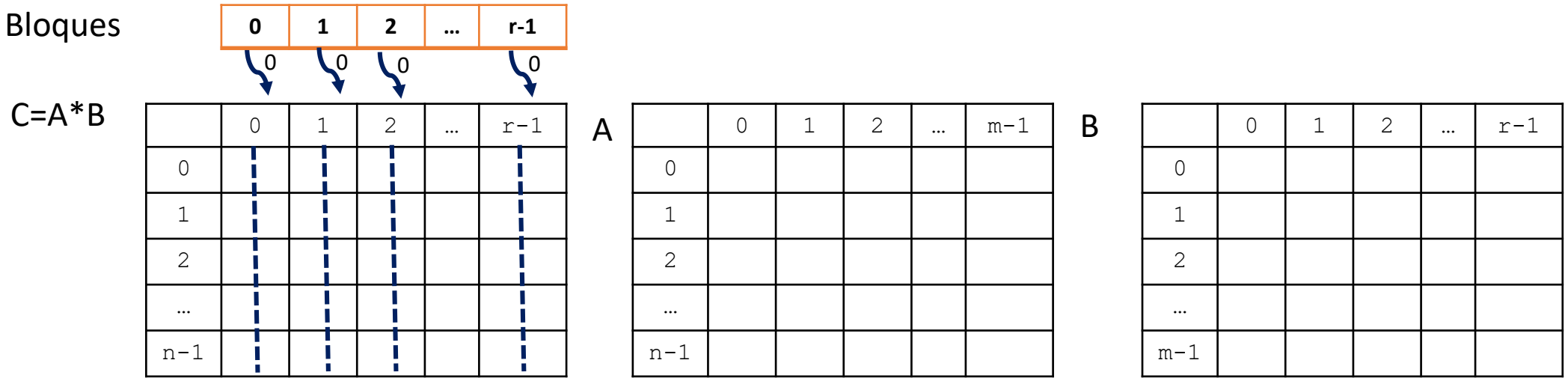
blockIdx.x	**threadIdx.x**	**tid**	**Elementos atendidos**	
0	0	0	$C_{0,0}, C_{0,1}, C_{0,2} \dots C_{0,r-1}$	0,1,2,..(r-1)
1	0	1	$C_{1,0}, C_{1,1}, C_{1,2} \dots C_{1,r-1}$	r,r+1,r+2,…,2r-1
2	0	2	$C_{2,0}, C_{2,1}, C_{2,2} \dots C_{2,r-1}$	2r,2r+1,2r+2,…,3r-1
…	…	…	…	…
n-1	0	n-1	$C_{n-1,0}, C_{n-1,1},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

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

```
#define widthN 240
#define widthM 239
#define widthR 240
#define epsilon float(0.000000001)
…
dim3 dimGrid(widthN);
dim3 dimBlock(1);
…
int fila = blockIdx.x;
float suma = 0;
int tid = fila * widthR;
for (int j = 0; j < widthR; j++){ // columnas de la fila
        suma = 0;
        for (int k = 0; k < widthM; k++){ // elemento
                suma = suma +a[(fila*widthM) + k] * b[(k*widthR) + j];
        }
        c[tid + j] = suma;
}
```



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

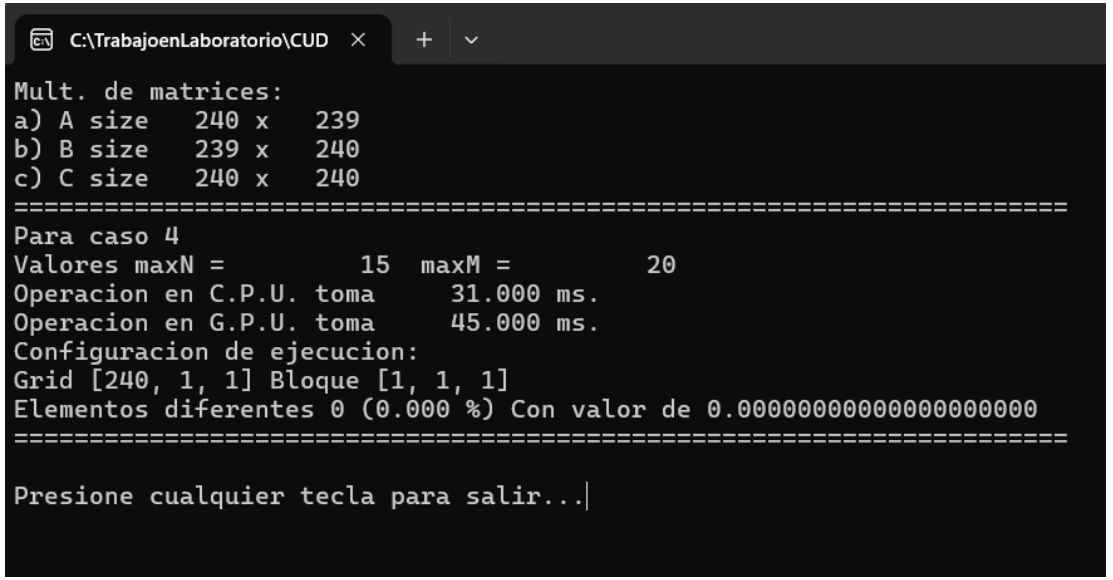


tid=blockIdx.x
primerElem=blockIdx.x

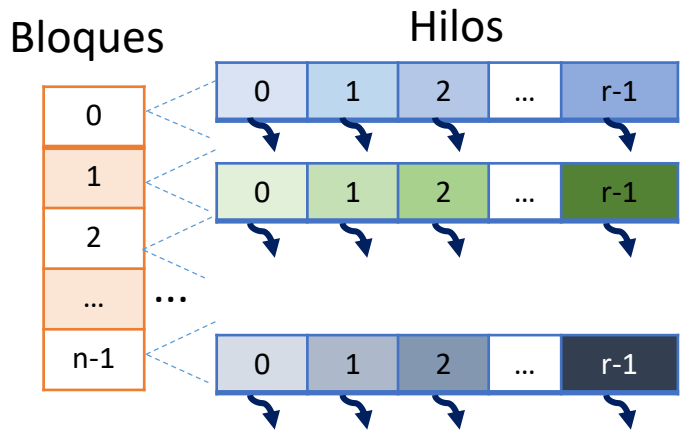
blockIdx.x	**threadIdx.x**	**tid**	**Elementos atendidos**	
0	0	0	$C_{0,0}, C_{1,0}, C_{2,0} \dots C_{n-1,0}$	0,r,2r,…,(n-1)r
1	0	1	$C_{0,1}, C_{1,1}, C_{2,1} \dots C_{n-1,1}$	1,r+1,2r+1,(n-1)r+1
2	0	2	$C_{0,2}, C_{1,2}, C_{2,2} \dots C_{n-1,2}$	2,r+2, 2r+2,…(n-1)r+2
…	…	…	…	…
r-1	0	r-1	$C_{0r-1}, C_{1,r-1},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

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

```
#define widthN 240
#define widthM 239
#define widthR 240
#define epsilon float(0.000000001)
…
dim3 dimGrid(widthR);
dim3 dimBlock(1);
…
int columna = blockIdx.x;
float suma = 0;
int tid = blockIdx.x;
for (int i = 0; i < widthN; i++) {  // filas de la columna
        suma = 0;
        for (int k = 0; k < widthM; k++) { // elemento
                suma = suma + a[(i * widthM) + k] * b[(k * widthR) + columna];
        }
        c[tid] = suma;
        tid = tid + widthR;
}
```



Caso 3. N bloques con R hilos cada uno.



C=A*B

	0	1	2	...	r-1
0					
1					
2					
...					
n-1					

A

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

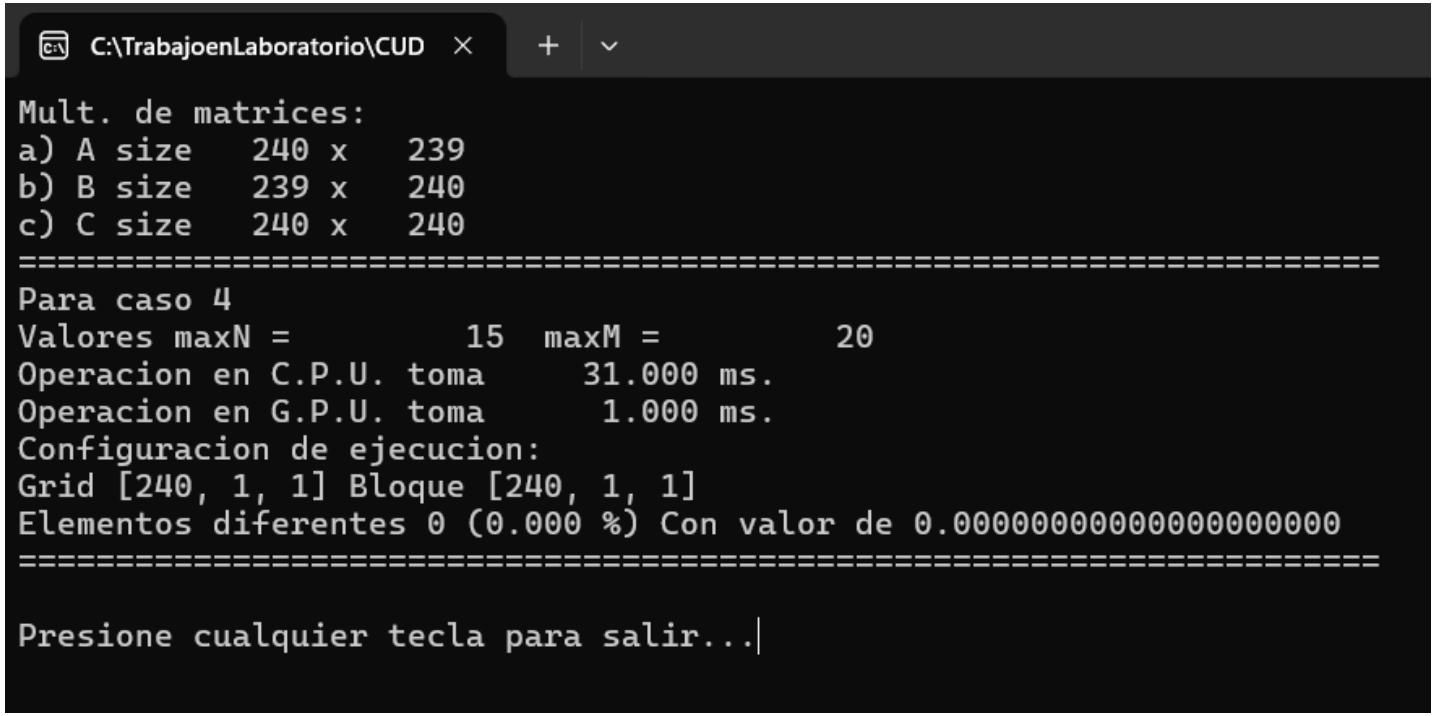
B

	0	1	2	...	r-1
0					
1					
2					
...					
m-1					

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

Caso 3. N bloques con R hilos cada uno.

```
#define widthN 240
#define widthM 239
#define widthR 240
#define epsilon float(0.000000001)
…
dim3 dimGrid(widthN);
dim3 dimBlock(widthR);
…
int fila = blockIdx.x;
int columna = threadIdx.x;
float suma = 0;
int tid = (fila * widthR)+columna;
for (int k = 0; k < widthM; k++){ // elemento
            suma = suma + a[(fila*widthM) + k] * b[(k*widthR) + columna];
}
c[tid] = suma;
```



Caso 4. Bloque 2D con hilos 2D

Hilos

	0	1	2
0	↘	↘	↘
1	↘	↘	↘
2	↘	↘	↘
3	↘	↘	↘

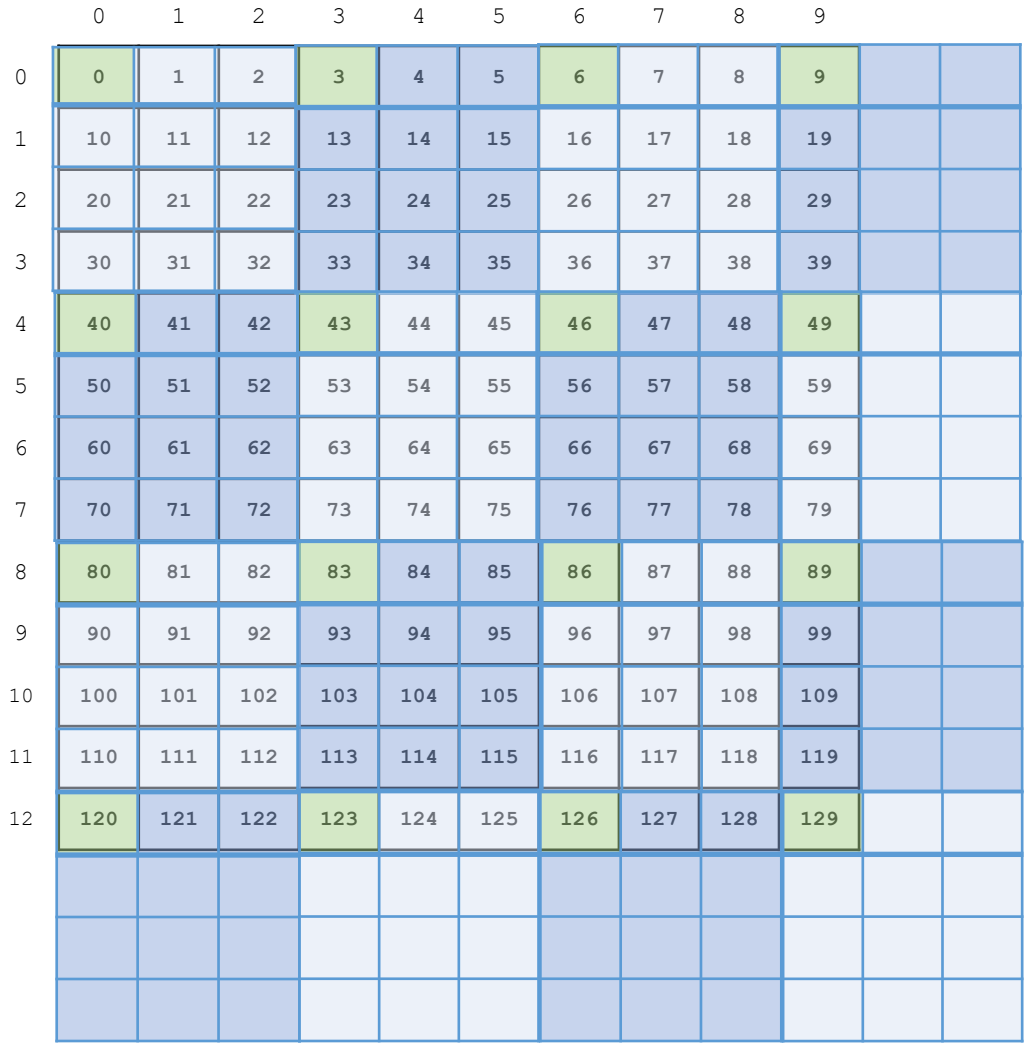
Tamaño de la matriz
13 x 10

maxN=4, maxM=3
Cada bloque tiene maxN x maxM hilos.

Dimensión del grid

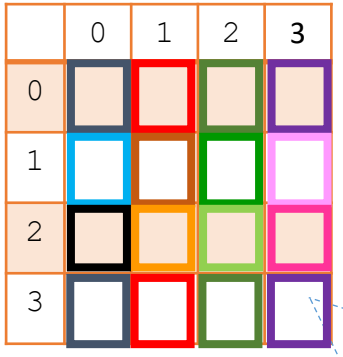
Primera dimensión
N / maxN = 13 / 4 = 3.25 ≈ 4
Segunda dimensión
R / maxM = 10 / 3 = 3.3333 ≈ 4

A * B

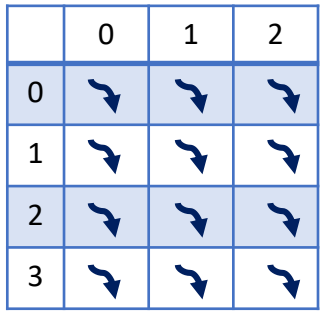


Caso 4. Bloque 2D con hilos 2D

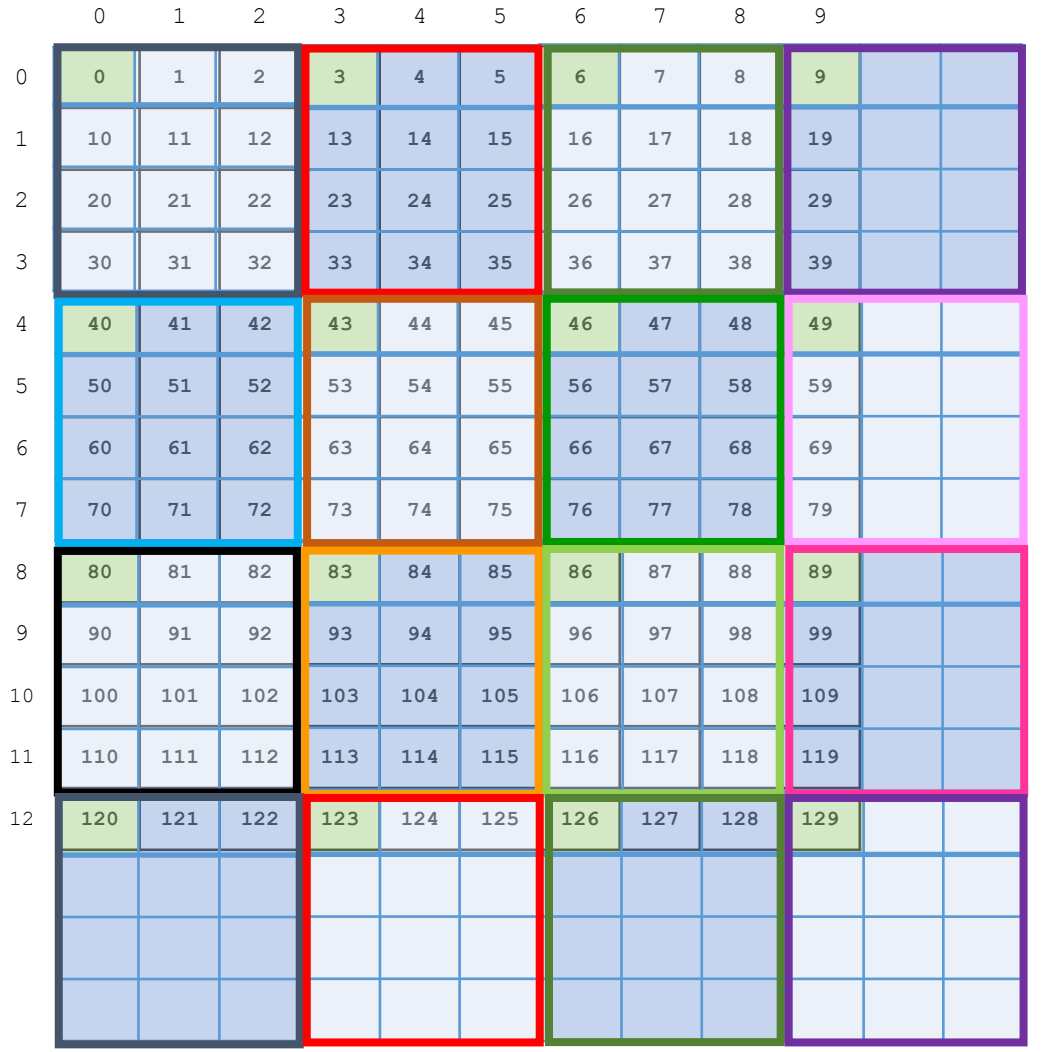
Bloque



Hilos



A * B



maxN=4, maxM=3

Cada bloque tiene maxN x maxM hilos.

filaInicialBloque=(blockIdx.x*blockDim.x)
fila=filaInicialBloque+threadIdx.x
columnaIniciaBloque=(blockIdx.y*blockDim.y)
columna=columnaInicialBloque+threadIdx.y

blockIdx		threadIdx		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

blockIdx		threadIdx		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

blockIdx		threadIdx		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

blockIdx		threadIdx		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

blockIdx		threadIdx		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

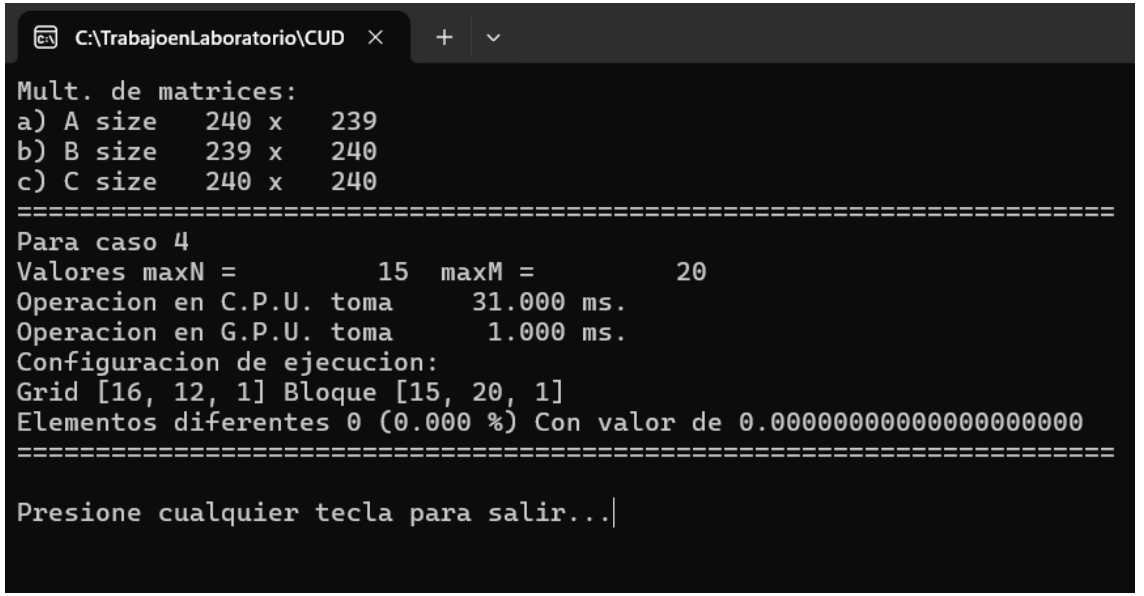
blockIdx		threadIdx		Elemento		
x	y	x	y	fila	col	#
2	2	0	0	8	6	86
2	2	0	1	8	7	87
2	2	0	2	8	8	88
2	2	1	0	9	6	96
2	2	1	1	9	7	97
2	2	1	2	9	8	98
2	2	2	0	10	6	106
2	2	2	1	10	7	107
2	2	2	2	10	8	108
2	2	3	0	11	6	116
2	2	3	1	11	7	117
2	2	3	2	11	8	118
2	3	0	0	8	9	89
2	3	0	1	8	10	90
2	3	0	2	8	11	91
2	3	1	0	9	9	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

blockIdx		threadIdx		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

blockIdx		threadIdx		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 4. Bloque 2D con hilos 2D

```
#define widthN 240
#define widthM 239
#define widthR 240
#define epsilon float(0.000000001)
#define maxN 15
#define maxM 20
…
int numBloquesN = divEntera(widthN , maxN);
int numBloquesR = divEntera(widthR , maxM);
dim3 dimGrid(numBloquesN, numBloquesR);
dim3 dimBlock(maxN, maxM);
…
int fila = (blockIdx.x * blockDim.x) + threadIdx.x;
int columna = (blockIdx.y * * blockDim.y) + threadIdx.y;
if ((fila < widthN) && (columna < widthR)) {
        int tid = (fila*widthR) + columna;
        float suma = 0;
        for (int k = 0; k < widthM; k++) {
                suma = suma + a[(fila*widthM) + k] * b[(k*widthR) + columna];
        }
        c[tid] = suma;
}
```



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**. Addisson 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**. Addisson Wesley. 2011.
- Kirk, David; Hwu, Wen-mei. **Programming Massively Parallel Processors: A Hands-on Approach**. Elsevier. 2010.

Gracias por su atención

DRA.
Sandra Luz
CANCHOLA
MAGDALENO

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