Práctica 07. Producto Punto

Sea $u \cdot v$ el producto punto entre dos vectores y θ el ángulo entre elllos

$$u \cdot v = \sum_{n} u_i * v_i$$

$$u \cdot v = \|u\|\|v\|\cos(\theta)$$

$$\cos(\theta) = \frac{u \cdot v}{\|u\|\|v\|}$$

$si\ (u \cdot v)\ \ es$

$u \cdot v = 0, entonces\ \theta = 90°\ \ \therefore son\ ortogonales$

$u \cdot v < 0\ \ , entonces\ \theta > 90°$

$u \cdot v > 0\ \ , entonces\ \theta < 90°$

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

Sea $u \cdot v$ el producto punto entre dos vectores y θ el ángulo entre elllos

$$u \cdot v = \sum_n u_i * v_i$$

$si\ (u \cdot v)\ \ es$

$u \cdot v = 0, entonces\ \theta = 90°\ \ \therefore son\ ortogonales$

$u \cdot v < 0\ \ , entonces\ \theta > 90°$

$u \cdot v > 0\ \ , entonces\ \theta < 90°$

$$u \cdot v = \|u\|\|v\|\cos(\theta)$$

$$\cos(\theta) = \frac{u \cdot v}{\|u\|\|v\|}$$

$\vec{u} = (1{,}5, -3)$ $\quad \|\vec{u}\| = \sqrt{1^2 + 5^2 + (-3)^2} = \sqrt{1 + 25 + 9} = \sqrt{35}$

$\vec{v} = (5{,}10{,}8)$ $\quad \|\vec{v}\| = \sqrt{5^2 + 10^2 + 8^2} = \sqrt{25 + 100 + 64} = \sqrt{189} = 3\sqrt{21}$

$\vec{u} \cdot \vec{v} = (1 * 5) + (5 * 10) + (-3 * 8) = 5 + 50 - 24 = 31$ $\quad \therefore\ \ \theta < 90°$

$$\cos(\theta) = \frac{u \cdot v}{\|u\|\|v\|} = \frac{31}{\sqrt{35} * 3\sqrt{21}} = \frac{31}{21\sqrt{15}} = 0.381150$$

$\theta = 67.595019°$

Producto punto de vectores n-Dimensionales

a

0	1	2	…	n-1
a_0	a_1	a_2		a_{n-1}

b

0	1	2	…	n-1
b_0	b_1	b_2		b_{n-1}

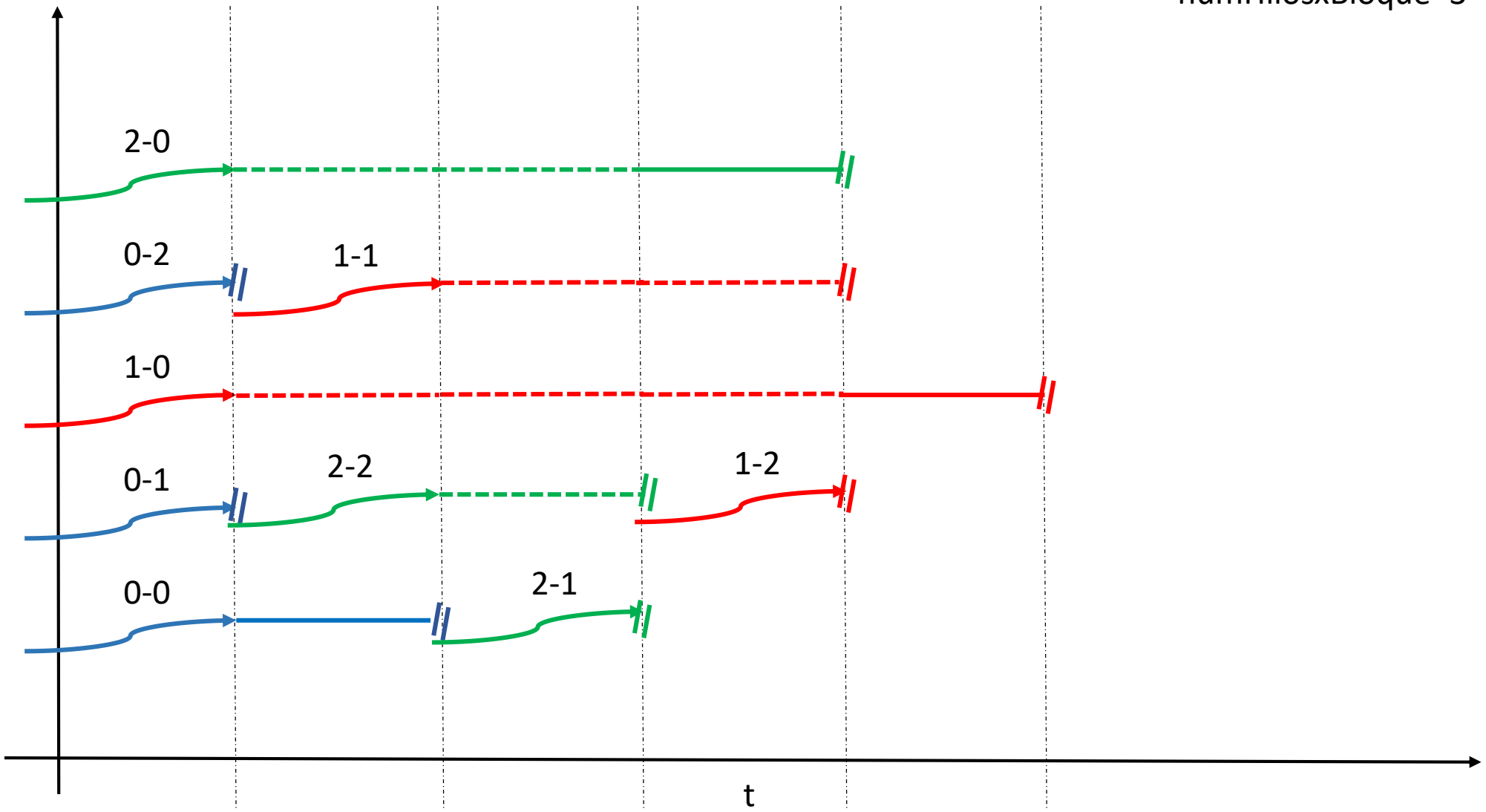
c=a*b

0	1	2	…	n-1
$a_0 * b_0$	$a_1 * b_1$	$a_2 * b_2$		$a_{n-1} * b_{n-1}$

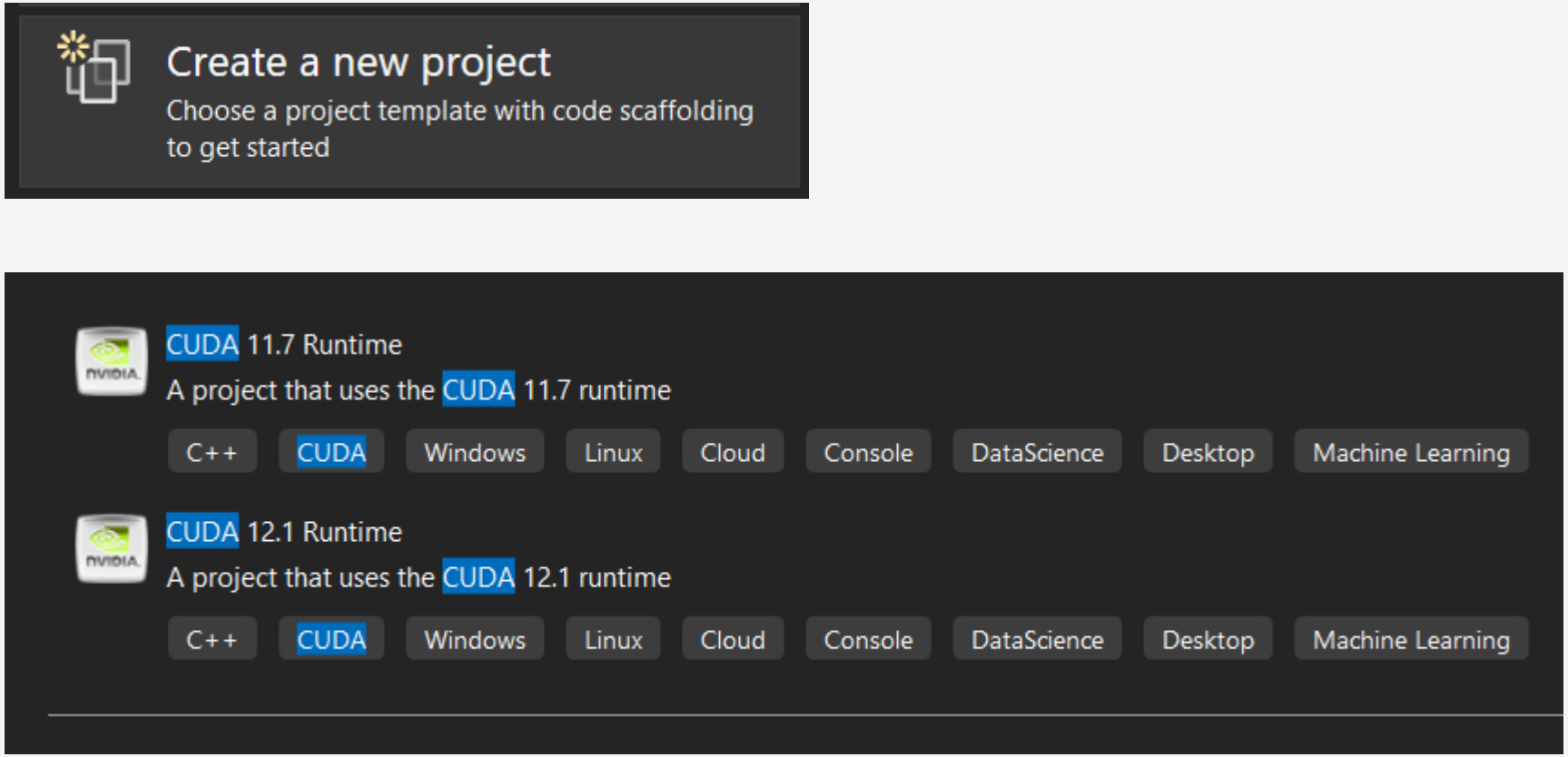
$$\vec{a} \cdot \vec{b} = \sum_{n} a_i * b_i$$

Sincronización de hilos

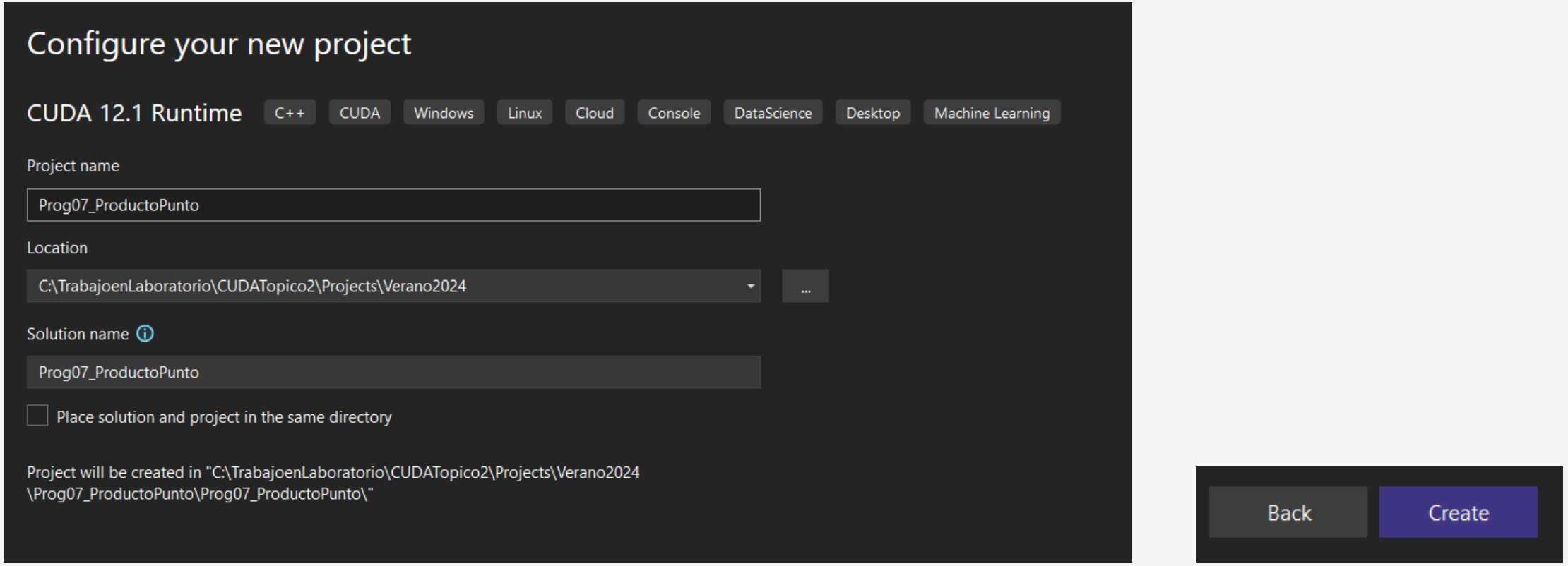
numBloques=3
numHilosxBloque=3



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	length	50								
A05	hilosxBloque	1024								
A10	a	α	ϕ	η	λ	τ	κ	π	ε	…	ω
A15	b	χ	γ	φ	θ	ι	ϖ	υ	β	…	δ
A20	gpu_axb	α * χ	ϕ * γ	η * φ	λ * θ	τ * ι	κ * ϖ	π * υ	ε * β	…	ω * δ
B01	cpu_axb	α * χ	ϕ * γ	η * φ	λ * θ	τ * ι	κ * ϖ	π * υ	ε * β	…	ω * δ
B10	gpu_axb_parcial	Σ_0		Σ_1		Σ_2		…		Σ_k	
B31	cpu_axb_parcial	Σ_0		Σ_1		Σ_2		…		Σ_k	
B45	dev_a	J10									
B80	dev_b	J45									
C30	dev_axb	J90									
E07	dev_axb_parcial	K01									
E10	dev_suma1	K05									
F20	dev_suma2	K10									
G05	sumaCPU	Σcpu									
H16	sumaGPU1	Σ1									
H20	sumaGPU2	Σ2									

GPU (Device)

J01										
J05										
J10		α	ϕ	η	λ	τ	κ	π	ε	…	ω
J45		χ	γ	φ	θ	ι	ϖ	υ	β	…	δ
J90		α * χ	ϕ * γ	η * φ	λ * θ	τ * ι	κ * ϖ	π * υ	ε * β	…	ω * δ
K01		Σ_0		Σ_1		Σ_2		…		Σ_k	
K05		Σ1									
K10		Σ2									
K15											
K20											
K30											
L07											
L10											

Memoria reservada

Apuntador

sandra.canchola@uaq.mx

Ejemplo:

HilosporBloque=3
NumBloques=4

a

0	1	2	3	4	5	6	7	8	9
5	6	7	5	4	2	1	0	3	-1

b

0	1	2	3	4	5	6	7	8	9
2	1	-1	2	4	8	7	-3	2	1

c=a*b

0	1	2	3	4	5	6	7	8	9
10	6	-7	10	16	16	7	0	6	-1

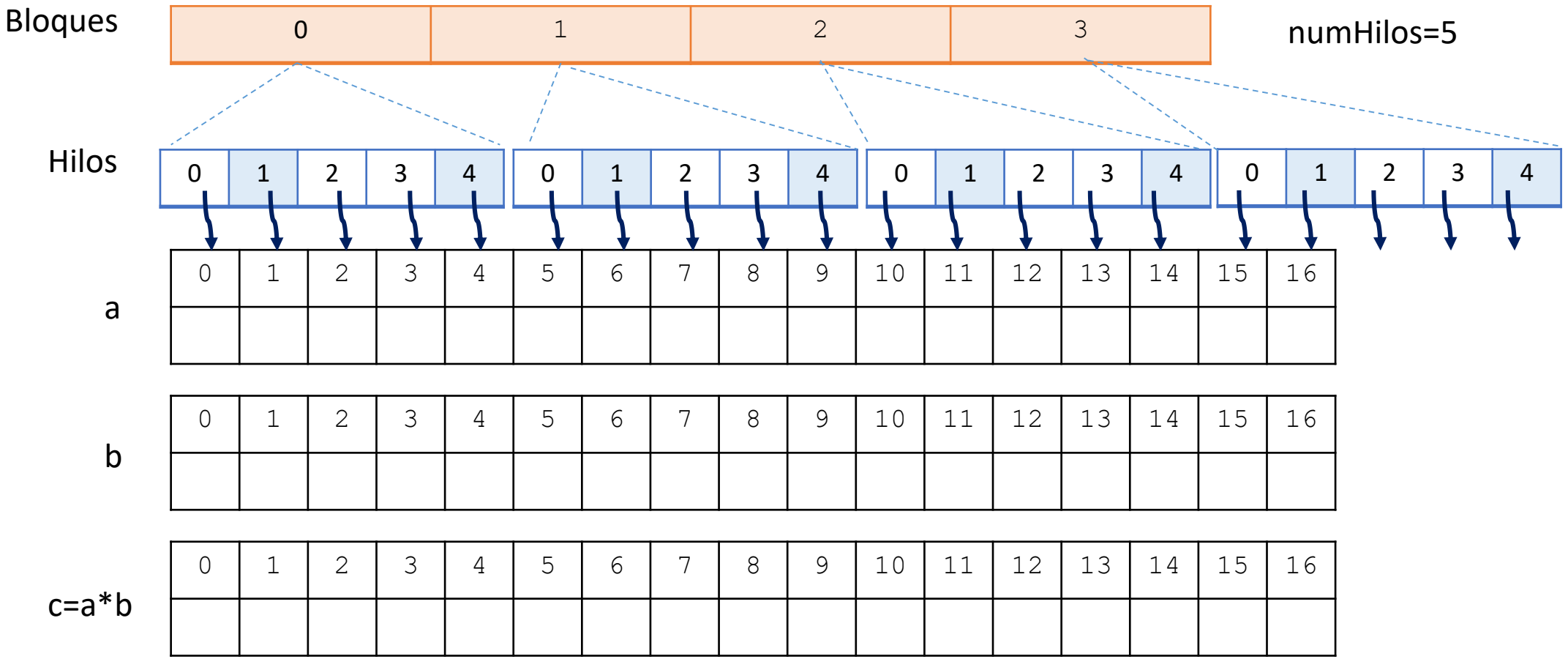
suma_parcial

0	1	2	3
9	42	13	-1

suma

63

Caso 1a. X bloques con numHilos c/u



blockIdx.x	threadIdx.x	tid
0	0	0
0	1	1
0	2	2
0	3	3
0	4	4
1	0	5
1	1	6
1	2	7
1	3	8
1	4	9
2	0	10
2	1	11
2	2	12
2	3	13
2	4	14
3	0	15
3	1	16
3	2	17
3	3	18
3	4	19

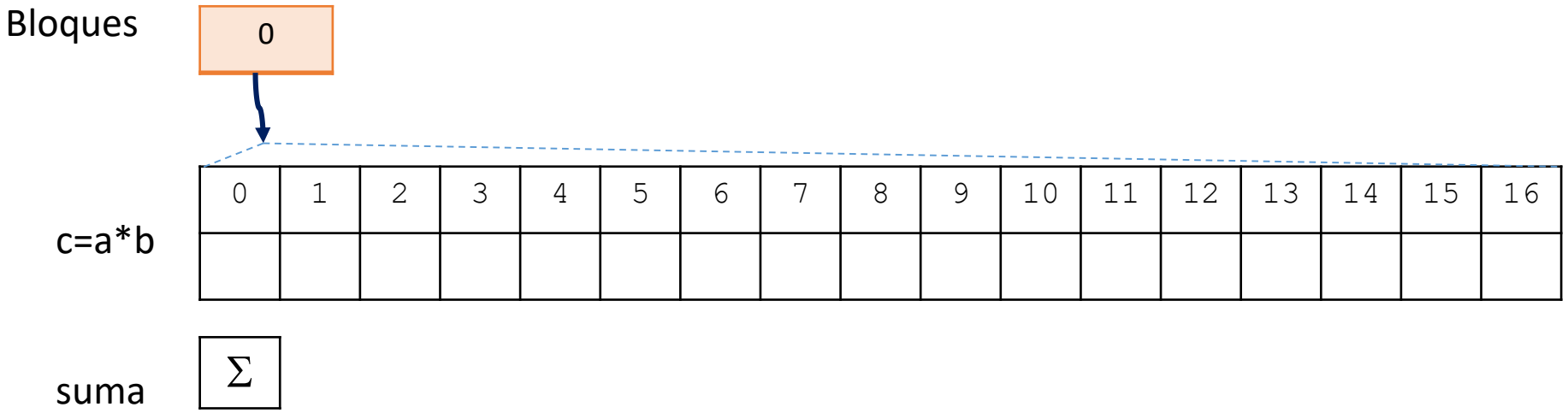
tid= (blockIdx.x*blockDim.x)+threadIdx.x

Caso 1a. X bloques con numHilos c/u

```
dim3 dimGrid(numBloques);
dim3 dimBlock(hilosxBloque);
…
__global__ void dot1(float* a, float* b, float* c) {
        int tid = (blockIdx.x * blockDim.x) + threadIdx.x;
        if (tid < length)
                c[tid] = a[tid] * b[tid];
}
__global__ void sumarDot1(float* c, float* suma) {
        float temp = 0;
        for (int i = 0; i < length; i++) {
                temp = temp + c[i];
        }
        *suma = temp;
}
```

```
dot1 << <dimGrid, dimBlock >> > (dev_a, dev_b, dev_axb);
sumarDot1 << <1, 1 >> > (dev_axb, dev_suma1);
```

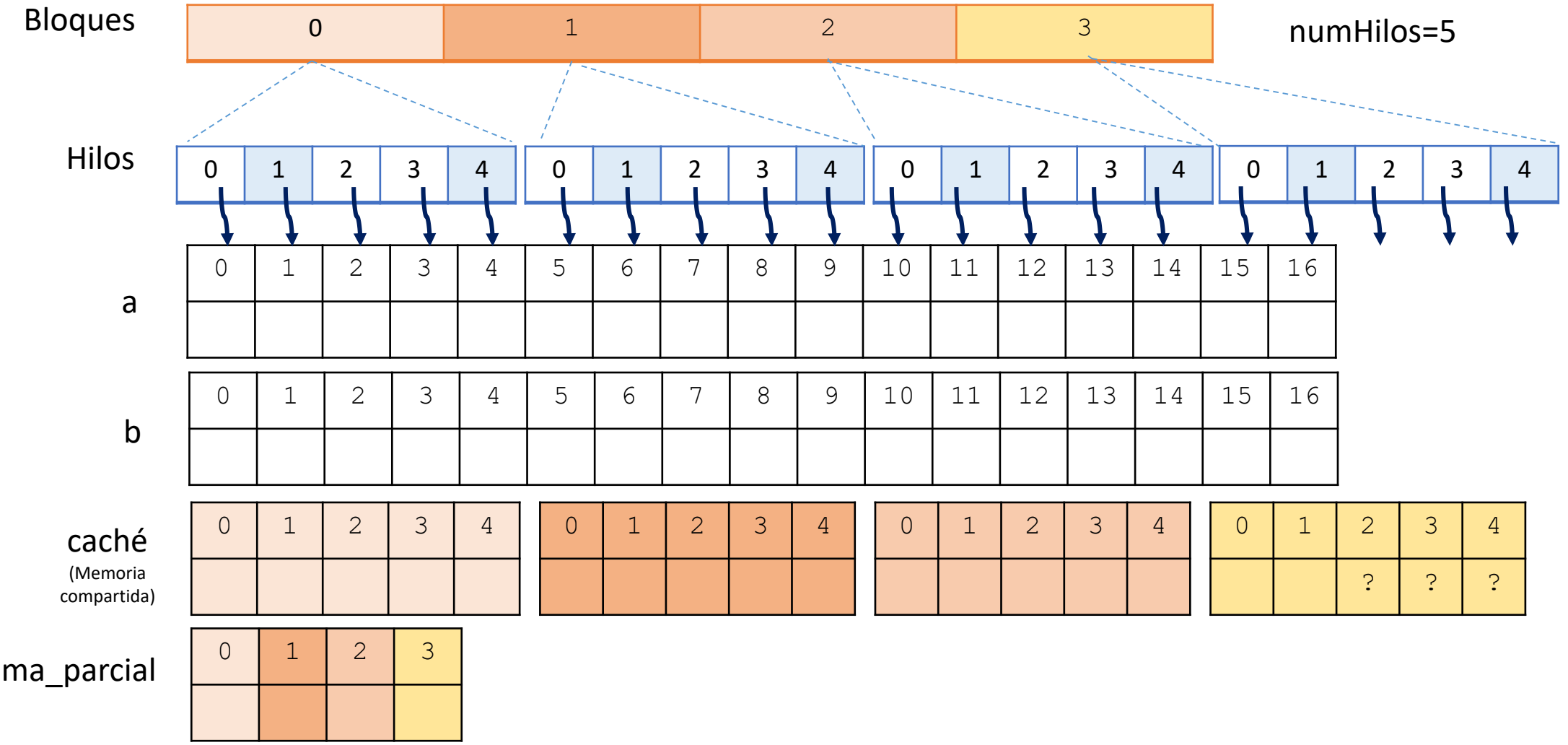
Caso 1b. Hilo único



Caso 1b. Hilo único

```
dim3 dimGrid(numBloques);
dim3 dimBlock(hilosxBloque);
dot1 << <dimGrid, dimBlock >> > (dev_a, dev_b, dev_axb);
sumarDot1 << <1, 1 >> > (dev_axb, dev_suma1);
…
__global__ void dot1(float* a, float* b, float* c) {
        int tid = (blockIdx.x * blockDim.x) + threadIdx.x;
        if (tid < length)
                c[tid] = a[tid] * b[tid];
}
__global__ void sumarDot1(float* c, float* suma) {
        float temp = 0;
        for (int i = 0; i < length; i++) {
                temp = temp + c[i];
        }
        *suma = temp;
}
```

Caso 2a. X bloques con numHilos c/u con memoria compartida



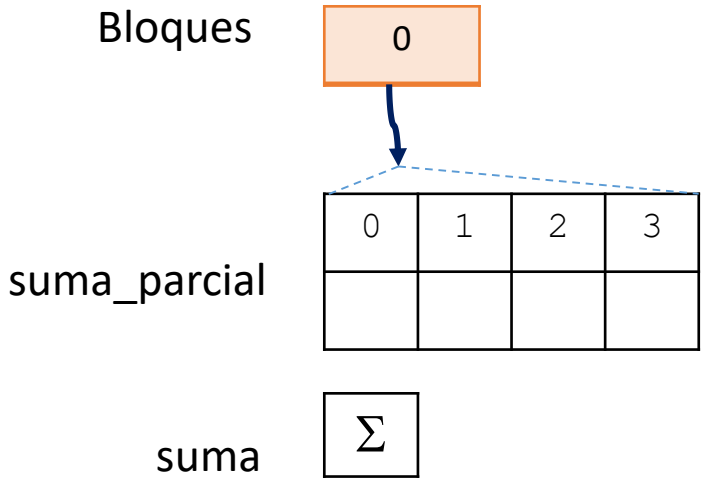
Caso 2a. X bloques con numHilos c/u con memoria compartida

```
dim3 dimGrid(numBloques);
dim3 dimBlock(hilosxBloque);
…
__global__ void dot2(float* a, float* b, float* c) {
        __shared__ float cache[hilosxBloque];
        int tid = (blockIdx.x * blockDim.x) + threadIdx.x;
        if (tid < length)
                cache[threadIdx.x] = a[tid] * b[tid];
        else
                cache[threadIdx.x] = 0;
        __syncthreads();
        // calculo de la suma parcial del bloque
        if (threadIdx.x == 0){  // soy el primer hilo
                float suma = 0;
                for (int i = 0; i < blockDim.x; i++) {
                        suma = suma + cache[i];
                }
                c[blockIdx.x] = suma;
        }
}
```

```
dot2 << <dimGrid, dimBlock >> > (dev_a, dev_b, dev_axb_parcial);
sumarDot2 << <1, 1 >> > (dev_axb_parcial, dev_suma2, numBloques);

 __global__ void sumarDot2(float* c, float* suma, int numBloques) {
        float temp = 0;
        for (int i = 0; i < numBloques; i++) {
                temp = temp + c[i];
        }
        *suma = temp;
 }
```

Caso 2b. Hilo único con memoria compartida

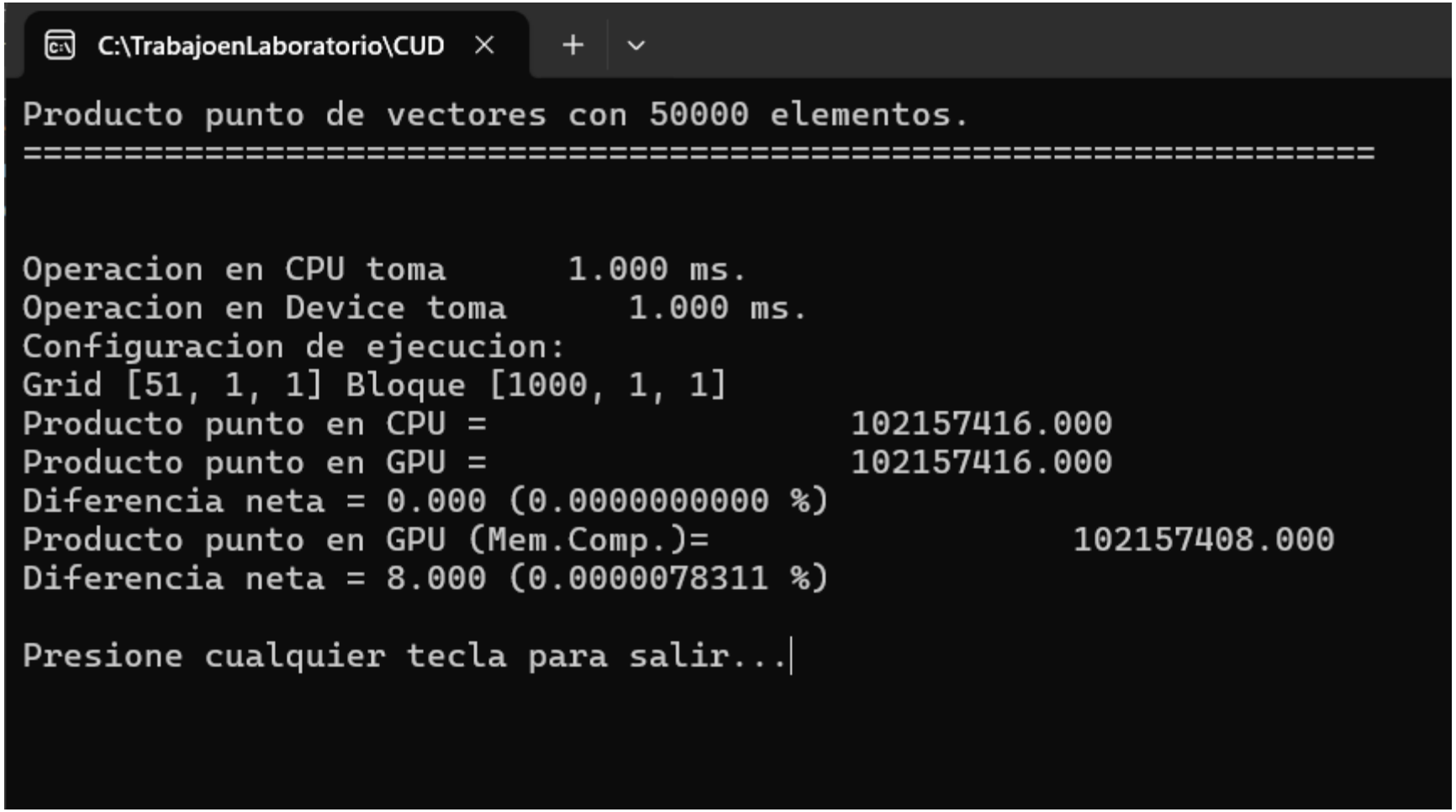


Caso 2b. Hilo único con memoria compartida

```
dim3 dimGrid(numBloques);
dim3 dimBlock(hilosxBloque);
…
__global__ void dot2(float* a, float* b, float* c) {
        __shared__ float cache[hilosxBloque];
        int tid = (blockIdx.x * blockDim.x) + threadIdx.x;
        if (tid < length)
                cache[threadIdx.x] = a[tid] * b[tid];
        else
                cache[threadIdx.x] = 0;
        __syncthreads();
        // calculo de la suma parcial del bloque
        if (threadIdx.x == 0){  // soy el primer hilo
                float suma = 0;
                for (int i = 0; i < blockDim.x; i++) {
                        suma = suma + cache[i];
                }
                c[blockIdx.x] = suma;
        }
}
```

```
dot2 << <dimGrid, dimBlock >> > (dev_a, dev_b, dev_axb_parcial);
sumarDot2 << <1, 1 >> > (dev_axb_parcial, dev_suma2, numBloques);

 __global__ void sumarDot2(float* c, float* suma, int numBloques) {
        float temp = 0;
        for (int i = 0; i < numBloques; i++) {
                temp = temp + c[i];
        }
        *suma = temp;
}
```


C:\TrabajoenLaboratorio\CUD
Producto punto de vectores con 50000 elementos.
==========
Operacion en CPU toma 1.000 ms.
Operacion en Device toma 1.000 ms.
Configuracion de ejecucion:
Grid [51, 1, 1] Bloque [1000, 1, 1]
Producto punto en CPU = 102157416.000
Producto punto en GPU = 102157416.000
Diferencia neta = 0.000 (0.0000000000 %)
Producto punto en GPU (Mem.Comp.)= 102157408.000
Diferencia neta = 8.000 (0.0000078311 %)
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**. 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