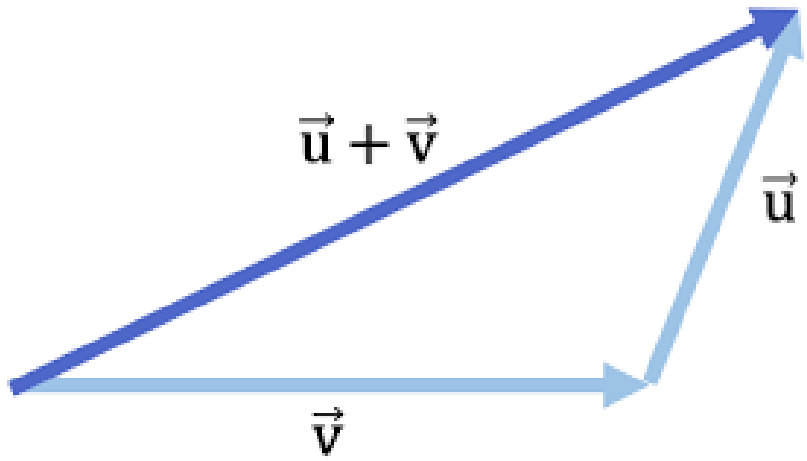
Práctica 03 Suma de vectores



Materia: Tópico II. (Procesamiento Paralelo con CUDA)

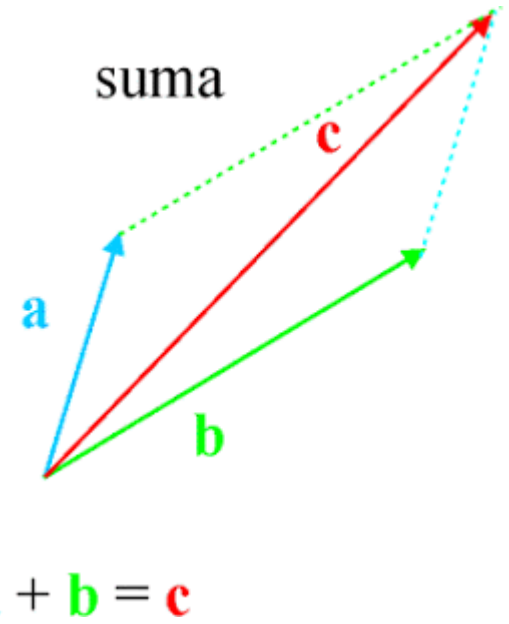
Dra. Sandra Luz Canchola Magdaleno
U.A.Q. Fac. de Informática
Correo: sandra.canchola@uaq.mx

Suma de vectores 2D y 3D

Sea $\vec{u}$ y $\vec{v}$ vectores en el mismo espacio , entonces:

2D: $\vec{u} + \vec{v} = (u_1 + v_1, u_2 + v_2)$

3D: $\vec{u} + \vec{v} = (u_1 + v_1, u_2 + v_2, u_3 + v_3)$



Suma 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
$a_0 + b_0$	$a_1 + b_1$	$a_2 + b_2$		$a_{n-1} + b_{n-1}$

Operaciones de memoria (CPU)

- **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)

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 dese el device.

  Ejemplo:

  cudaMemcpy(destino,origen, tamaño_de_memoria, indicador_flujo_de_inf)

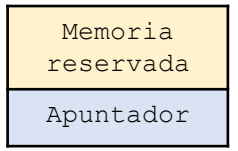
  Donde Indicador= cudaMemcpyHostToDevice, cudaMemcpyDeviceToHost, cudaMemcpyDeviceToDevice

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

  Ejemplo:

  cudaFree (apuntador)

Memoria



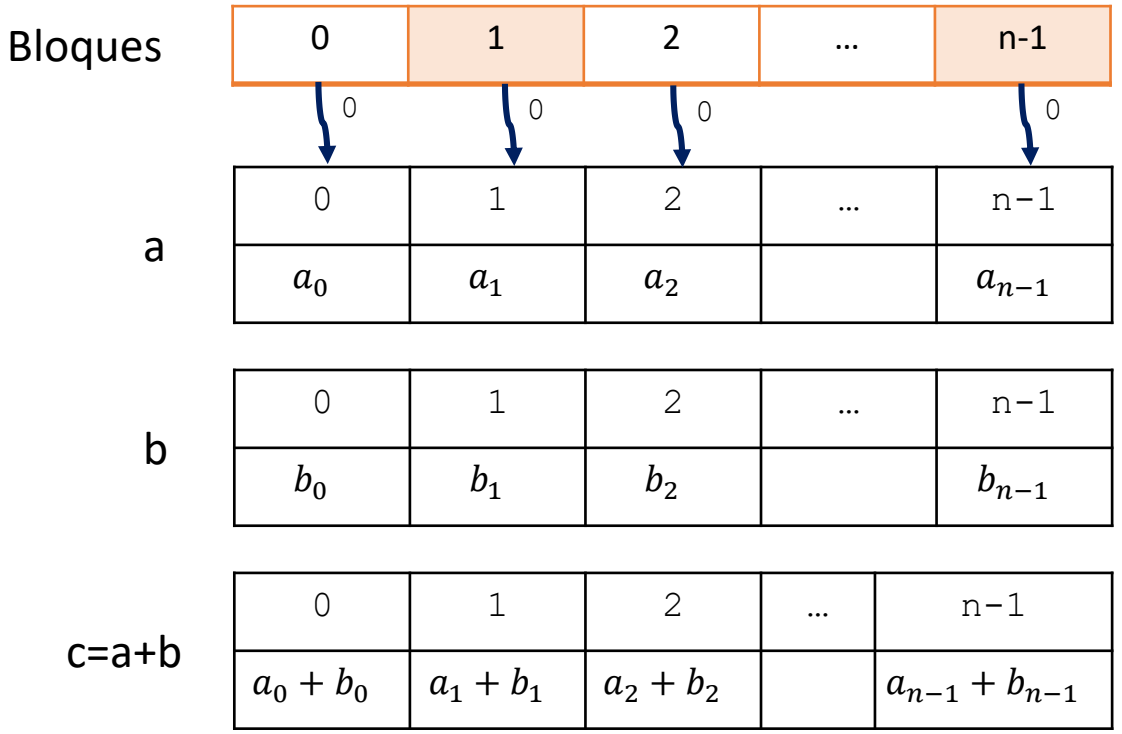
CPU (Host)

A01	length	50									
A05	maxHilos	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_c	α + χ	ϕ + γ	η + φ	λ + θ	τ + ι	κ + ϖ	π + υ	ε + β	…	ω + δ
B01	cpu_c	α + χ	ϕ + γ	η + φ	λ + θ	τ + ι	κ + ϖ	π + υ	ε + β	…	ω + δ
B05											
B10	dev_a	D10									
B15	dev_b	D45									
B20	dev_c	D90									
C30											
E07											
E10											

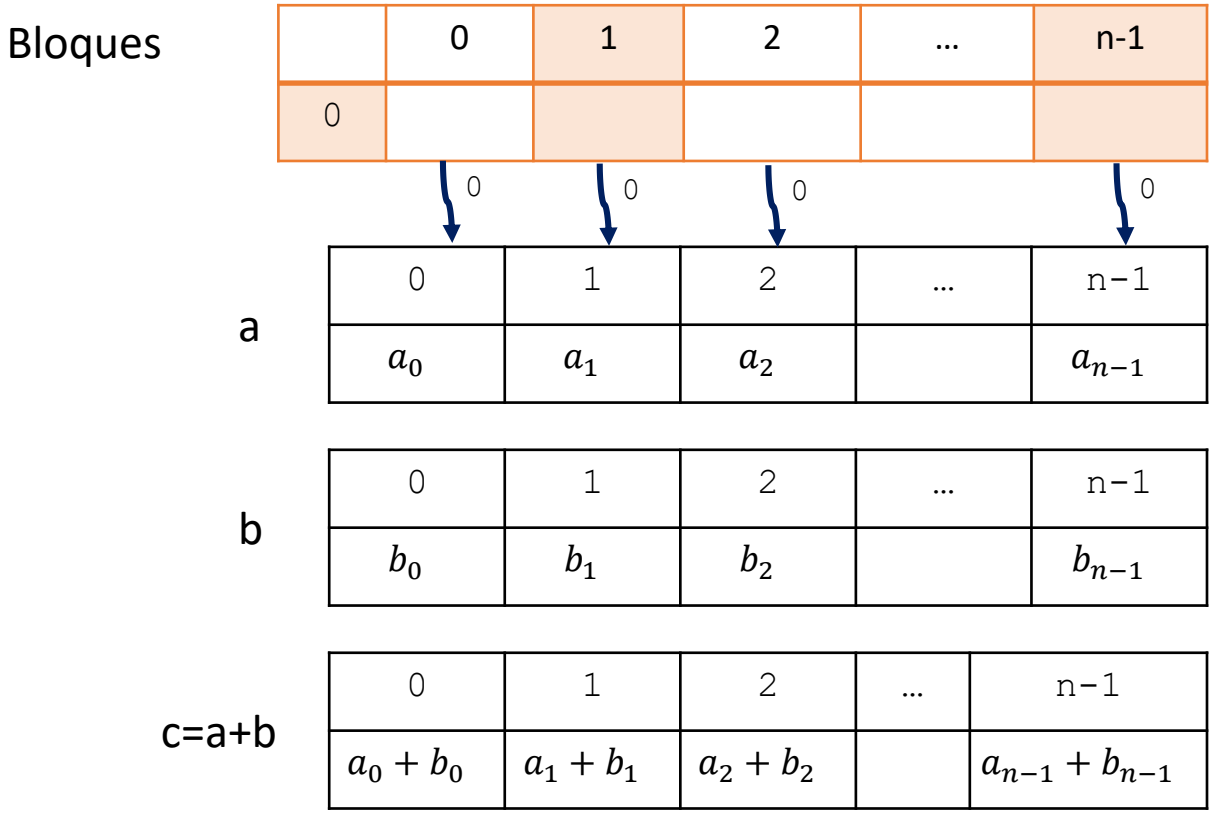
GPU (Device)

D01										
D05										
D10	α	ϕ	η	λ	τ	κ	π	ε	…	ω
D45	χ	γ	φ	θ	ι	ϖ	υ	β	…	δ
D90	α + χ	ϕ + γ	η + φ	λ + θ	τ + ι	κ + ϖ	π + υ	ε + β	…	ω + δ
F01										
F05										
F10										
F15										
F20										
G30										
H07										
I10										

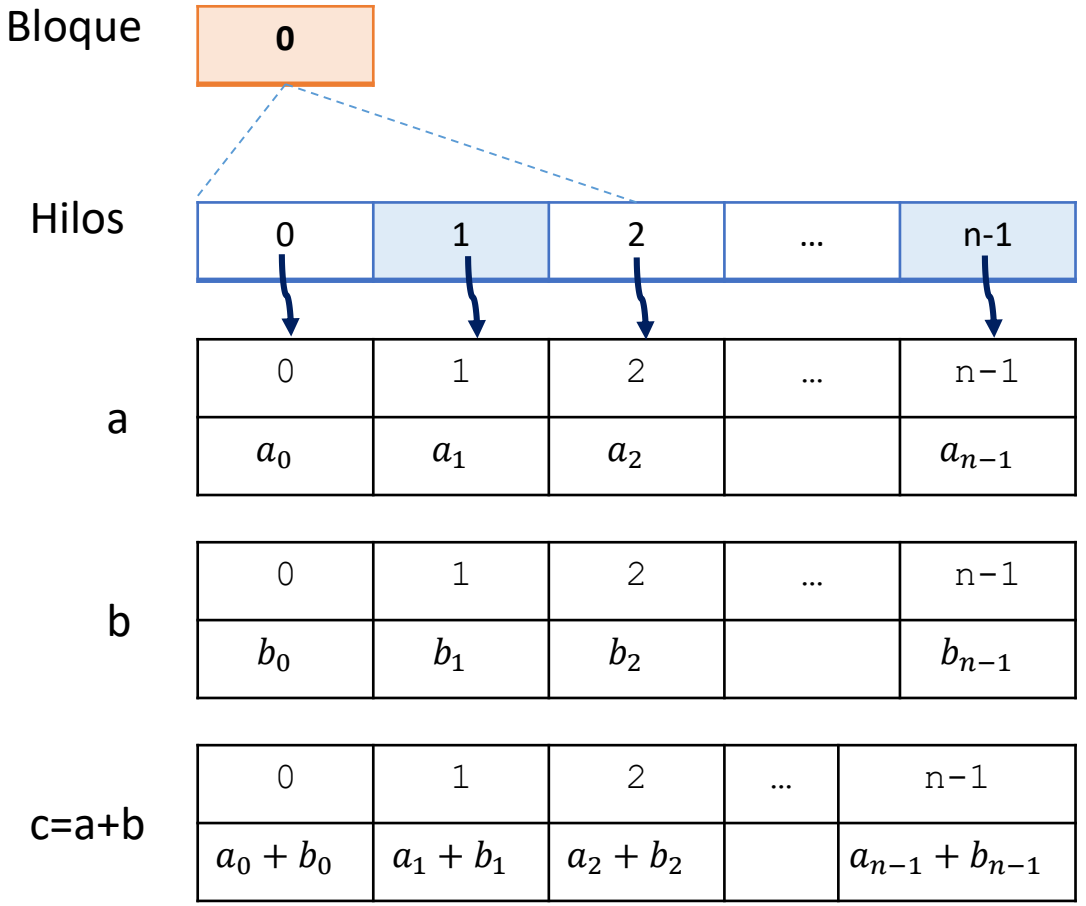
Caso 1. N Bloques con un hilo único



Caso 2. 1xN Bloques con un hilo único



Caso 3. Un bloque con N hilos

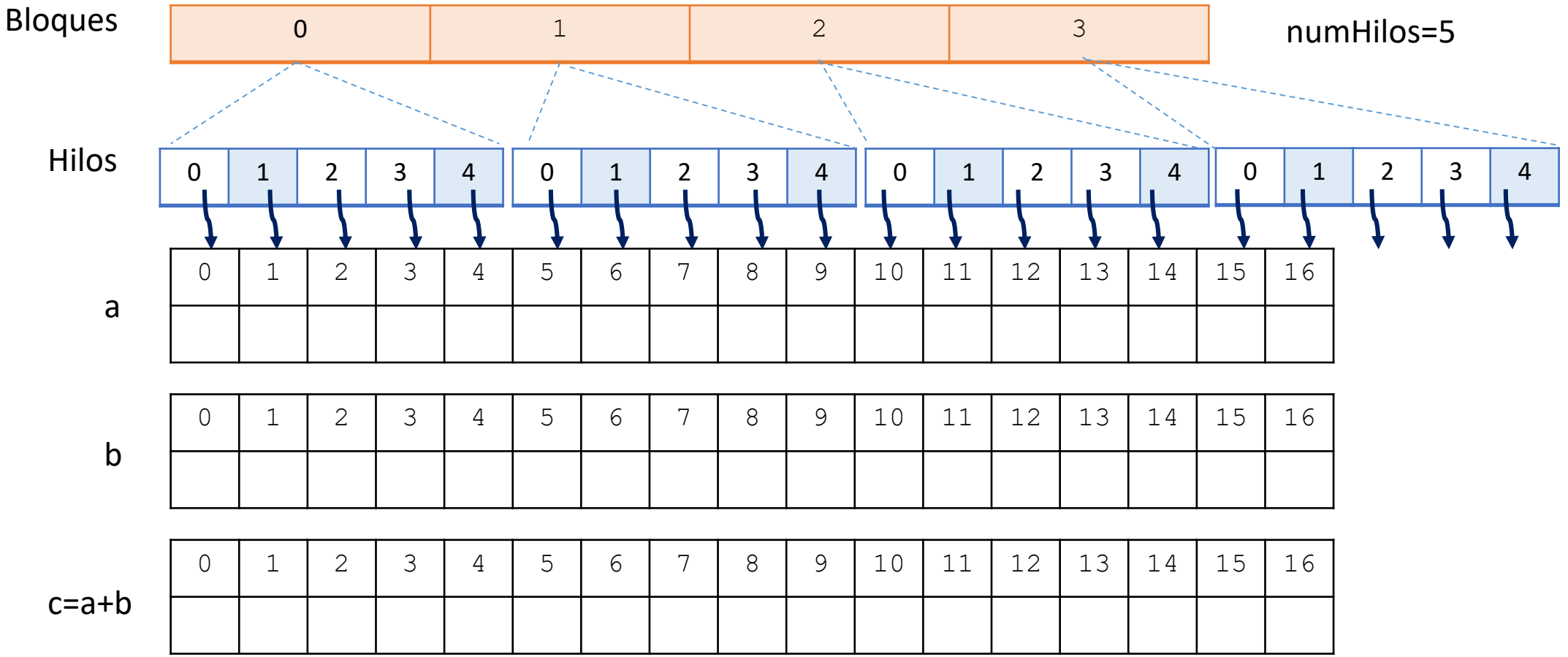


Caso 4. X bloques con numHilos c/u

Ejemplo:

Si length es 5000 y numHilos es 1024, entonces se generan [(5000/1024)≈5] 5 bloques, por lo tanto tenemos 5120 hilos. Si se requieren 5000 hilos, tenemos 120 hilos extras sin hacer trabajo (ociosos).

Caso 4. 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 5. X bloques con numHilos c/u tratando de generar los menos hilos ociosos

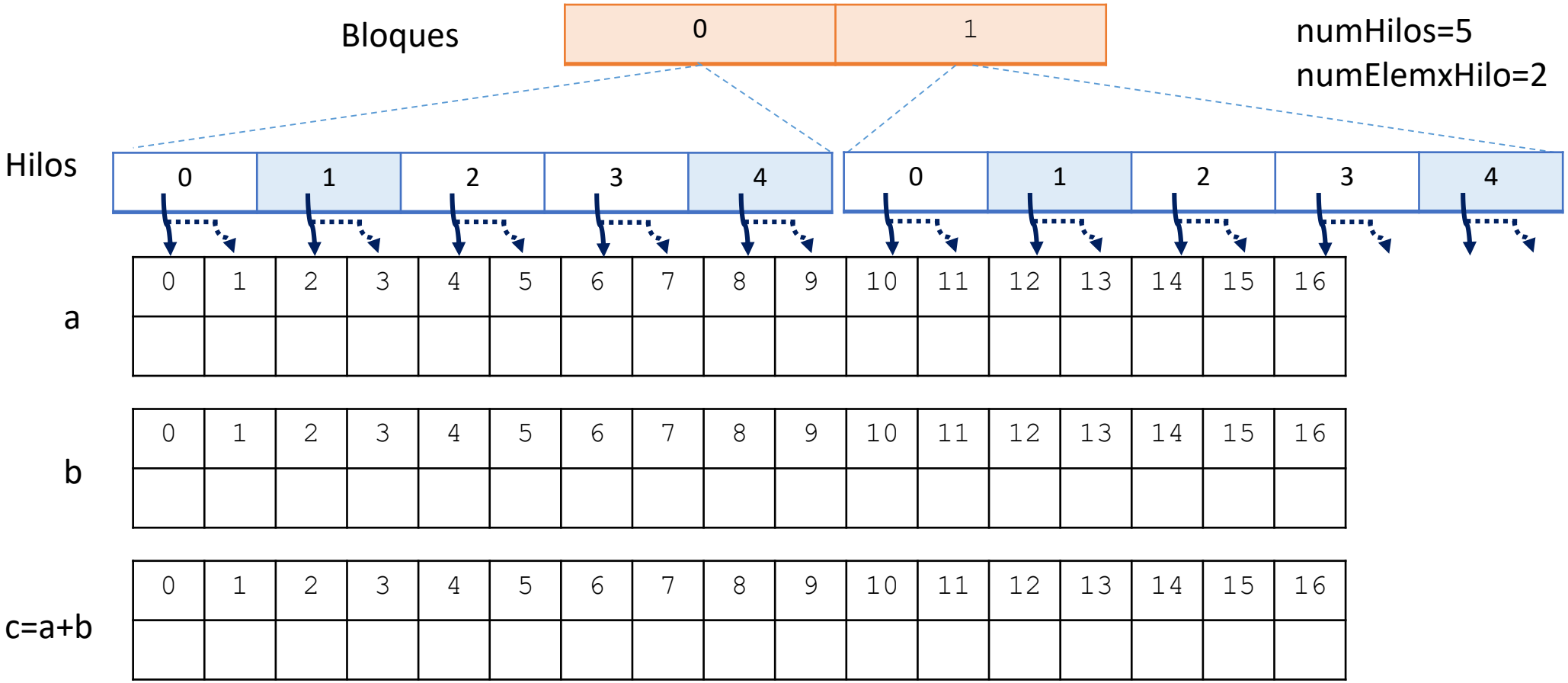
Ejemplo:

Si length es 5000 y maxHilos es 1024, entonces:
NumBloques= 5000/1024 = 4.8828 ≈ 5
NumHilos= 5000/5 = 1000 (Hilos totales 5000)

Si length es 4833 y maxHilos es 1024, entonces:
NumBloques= 4833/1024 = 4.7197 ≈ 5
NumHilos= 4833/5 = 966.6 ≈ 967 (Hilos totales 4835)

Si length es 2512 y maxHilos es 1024, entonces:
NumBloques= 2512/1024 = 2.453 ≈ 3
NumHilos= 2512/3 = 837.333 ≈ 838 (Hilos totales 2514)

Caso 6. X bloques con numHilos que atienden cada uno a numElemxHilo c/u



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

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