Programación Paralela

Estrategias de paralelización: Tiling

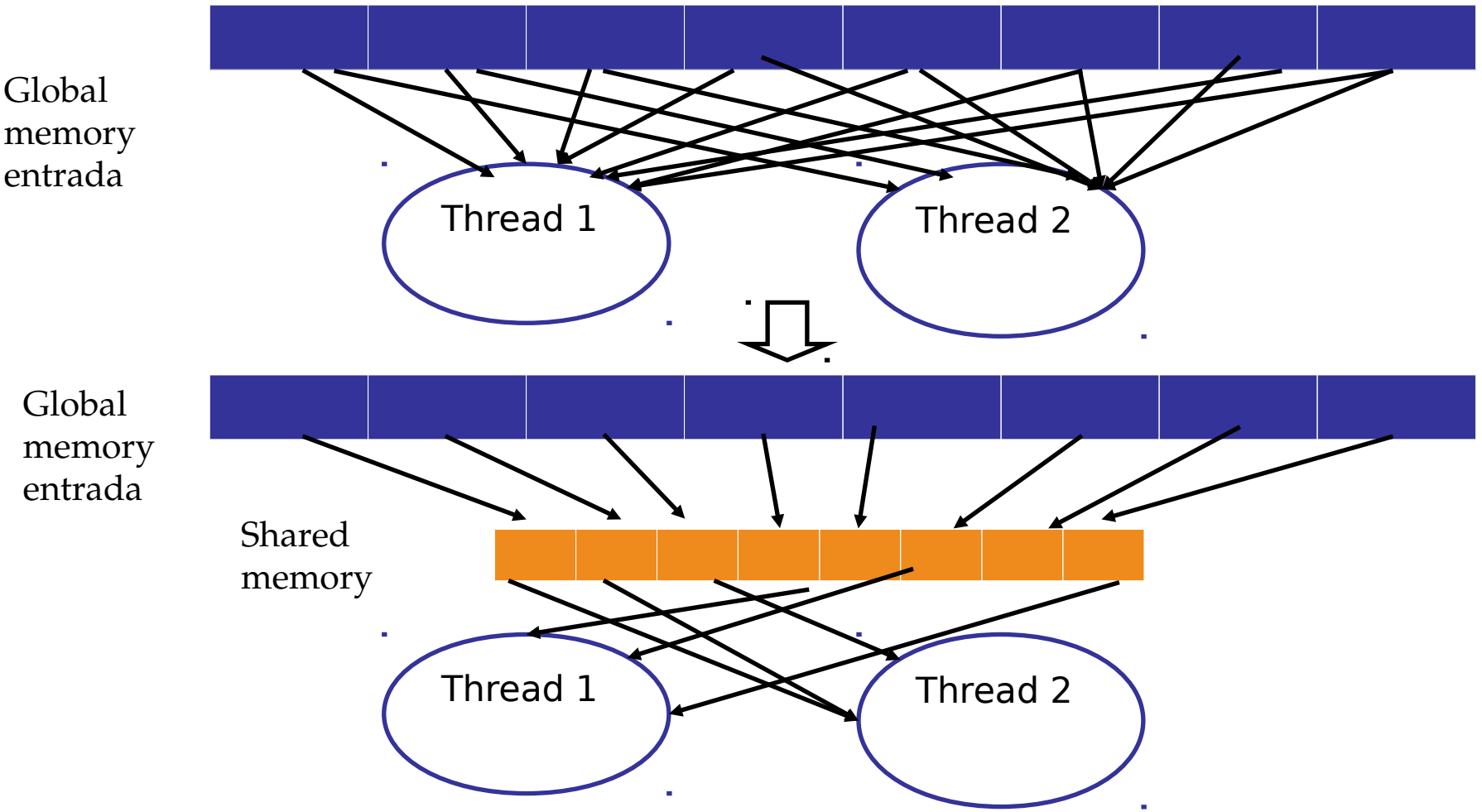
José María Cecilia Canales

UCAM | Universidad Católica de Murcia

Una estrategía común de programación

- La *global memory* es mucho mas lenta que la *shared memory*.
- Estrategia ***Tiling*** para aprovechar la *shared memory*:
  - Particionamos los datos en subconjuntos que quepan en *shared memory* (48 KB). Esto es lo que llamamos ***Tile***
  - Manejar cada subconjunto de datos con un bloque de hilos:
    - Cargamos un subconjunto de datos de *global memory* a *shared memory*, usando varios hilos para aprovechar el paralelismo de datos.
    - Realizamos las operaciones sobre cada subconjunto en *shared memory*.
    - Copiamos los resultado desde *shared memory* a *global memory.*

Usando bloques en la shared memory: Tiling



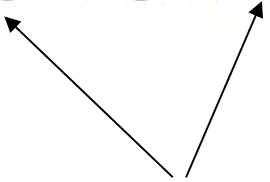
Resumen de la técnica

- Identificar un bloque (tile) de memoria global que se accede por múltiples hilos.
- Cargar el *tile* de memoria global en la *shared memory.*
- Todos los hilos acceden a los datos en shared memory.
- Traer el siguiente tile de memoria global a shared memory.

Multiplicación de matrices usando shared memory

Recordatorio de la M*M usando múltiple bloques

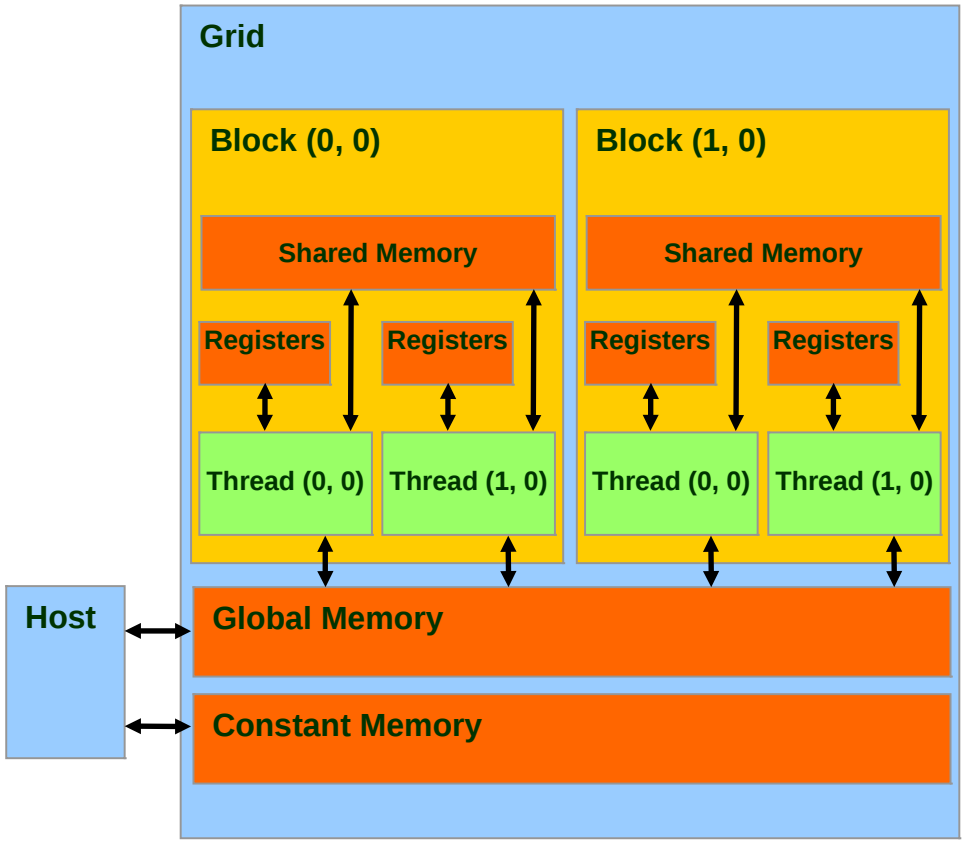
```
__global__ void MatrixMulKernel(float* Md, float* Nd, float* Pd, int Width) {

        // Calcula los indices de las filas de Pd y M
        int Row = blockIdx.y*TILE_WIDTH + threadIdx.y;
        // Calcula el indice de columnas de Pd y N
        int Col = blockIdx.x*TILE_WIDTH + threadIdx.x;
        float Pvalue = 0;
        // Cada hilo calcula un elemento de la submatriz asignada al bloque
        for (int k = 0; k < Width; ++k)
            Pvalue += Md[Row*Width+k] * Nd[k*Width+Col];

        Pd[Row*Width+Col] = Pvalue;
}
```



Accesos a device memoria (8 bytes) por cada 1 madd (2 FLOP) → 4 Bytes/FLOP

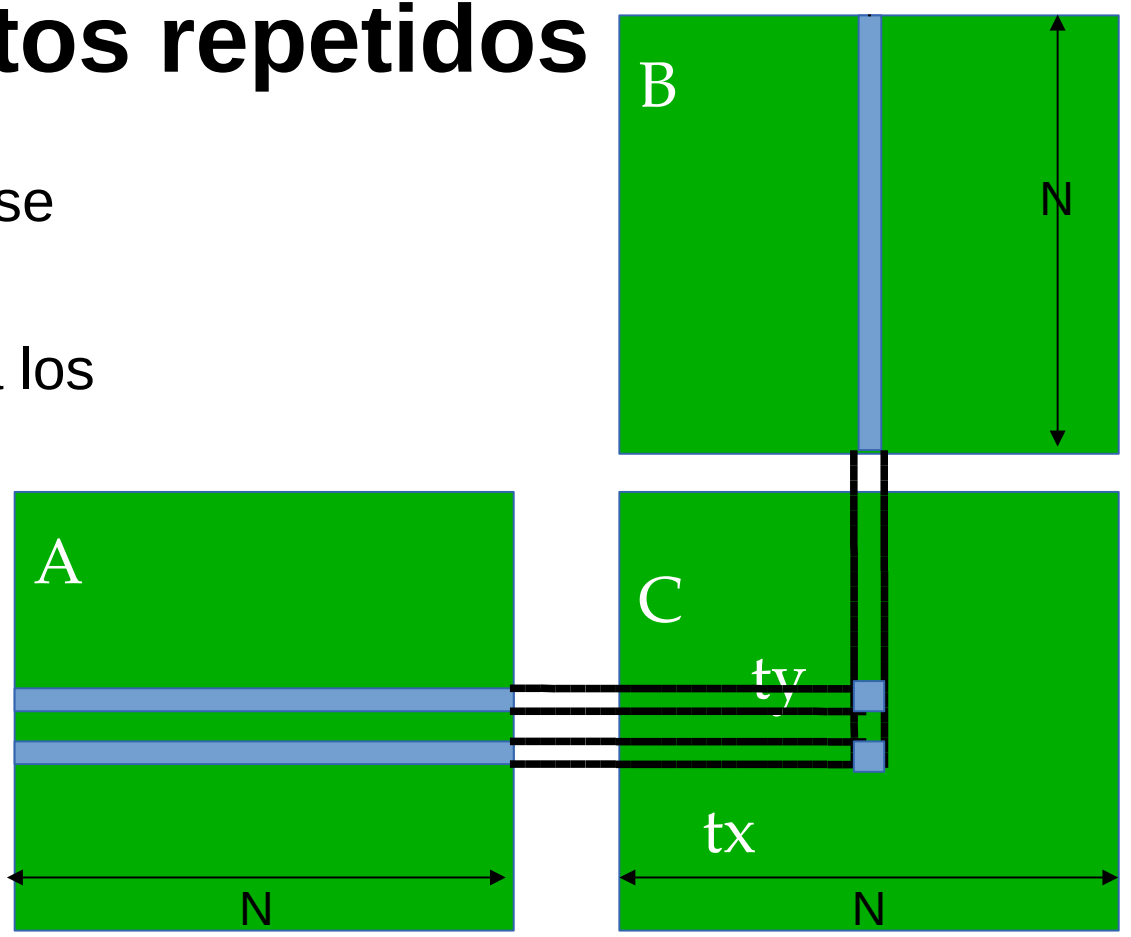
Análisis de Rendimento en G80.

- Todos los hilos acceden a global memory para obtener sus matrices de entrada
  - 2 accesos a memoria (8 bytes) por 2 FLOP
  - 4B/FLOP de ancho de banda bandwidth/FLOPS
  - 4B*346.5 GFLOPS (pico) = 1386 GB/s se necesita para obtener el pico máximo de FLOPS
  - 86.4 GB/s limita el código a 21.6 GFLOPS
- Esta versión del código consigue 15 GFLOPS
- Necesitamos rebajar los accesos a memoria drásticamente para conseguir el pico de 346.5 GFLOPS



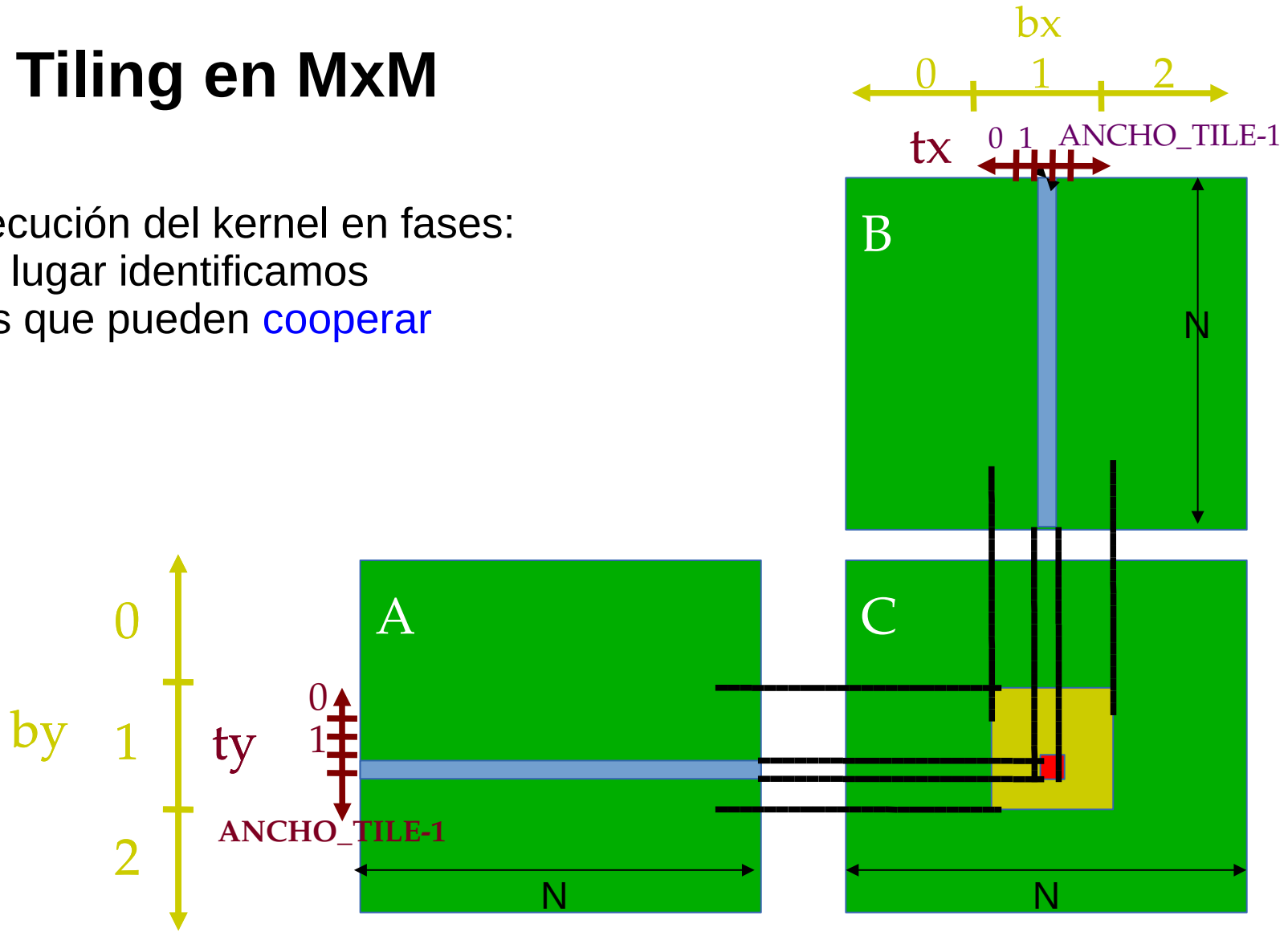
Elementos repetidos

- Cada elemento de entrada se lee por N hilos
- Hay hilos que se acceden a los mismos datos



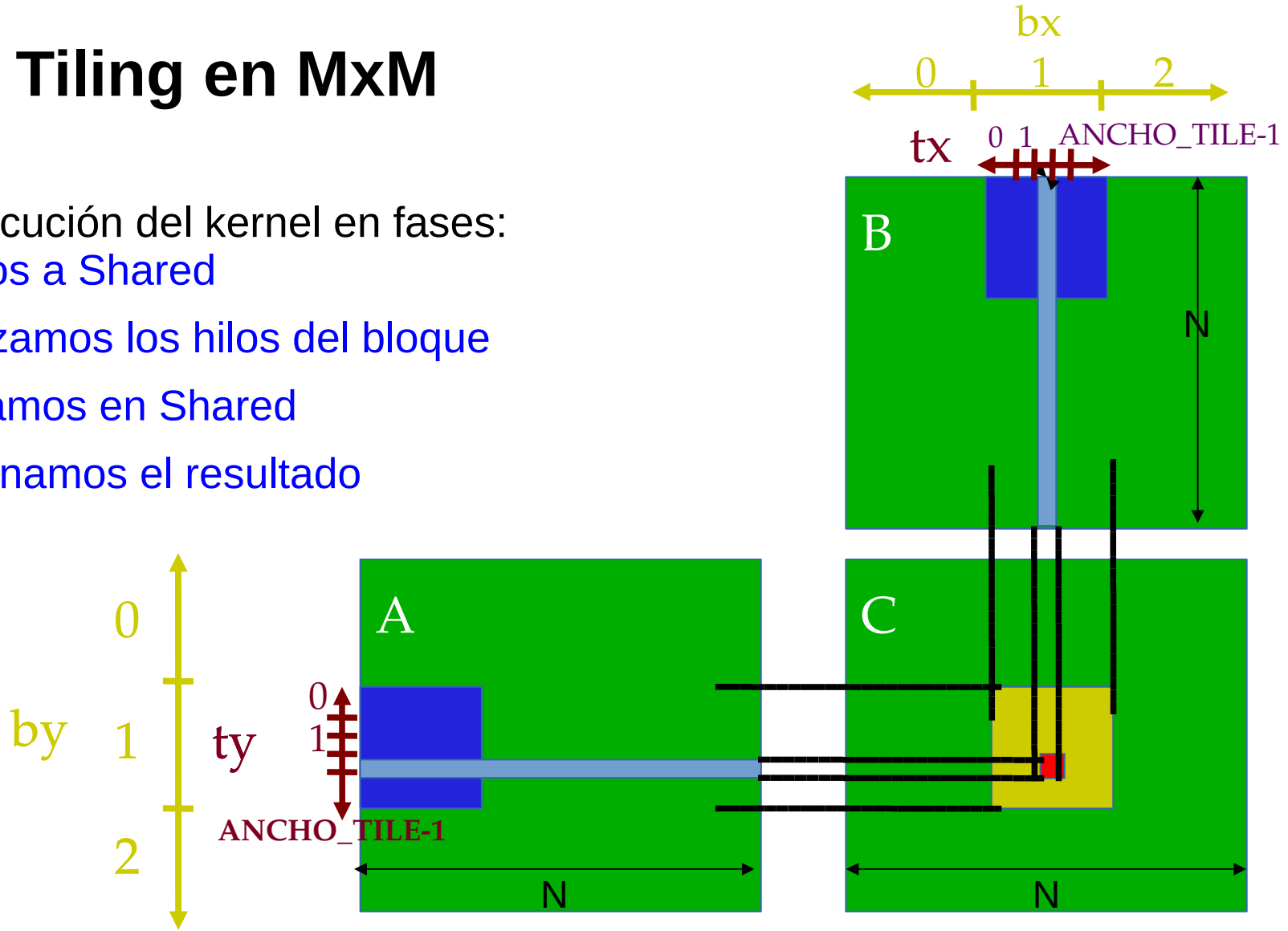
Tiling en MxM

- Dividimos la ejecución del kernel en fases:
  - En primer lugar identificamos elementos que pueden cooperar



Tiling en MxM

- Dividimos la ejecución del kernel en fases:
  1) Cargamos a Shared
  2) Sincronizamos los hilos del bloque
  3) Computamos en Shared
  4) Almancenamos el resultado



Sincronizar los hilos de un bloque

- Llamada a la función CUDA *__syncthreads()*

- Todos los hilos de un bloque deben llegar *__syncthreads()* antes de continuar.

- Se utiliza para coordinar algoritmos que utlizan tiling.
  - Tenemos que asegurar que todos los hilos han cargado su elemento.

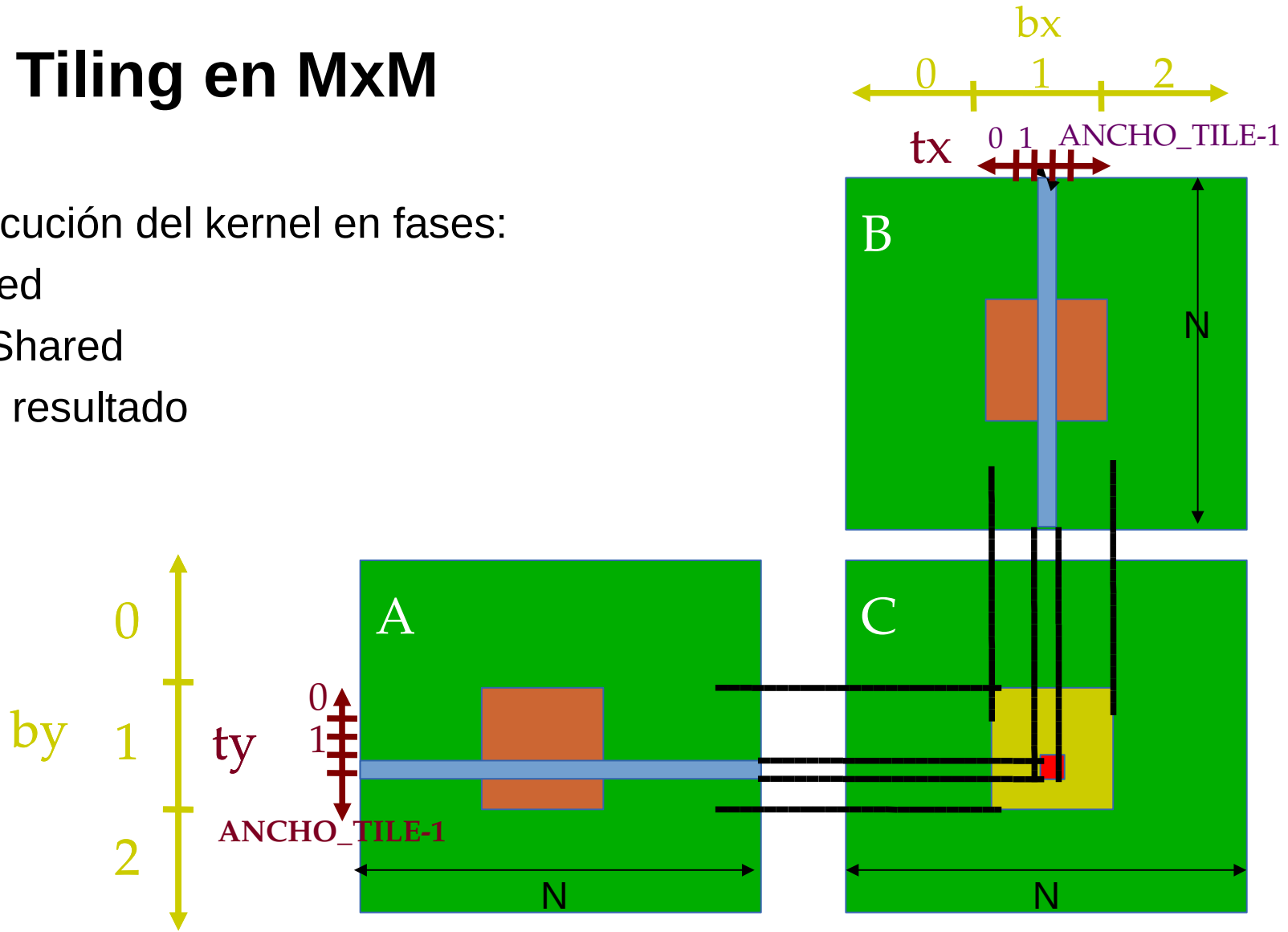
Tiling en MxM

- Dividimos la ejecución del kernel en fases:

  Cargamos a Shared

  Computamos en Shared

  Almancenamos el resultado



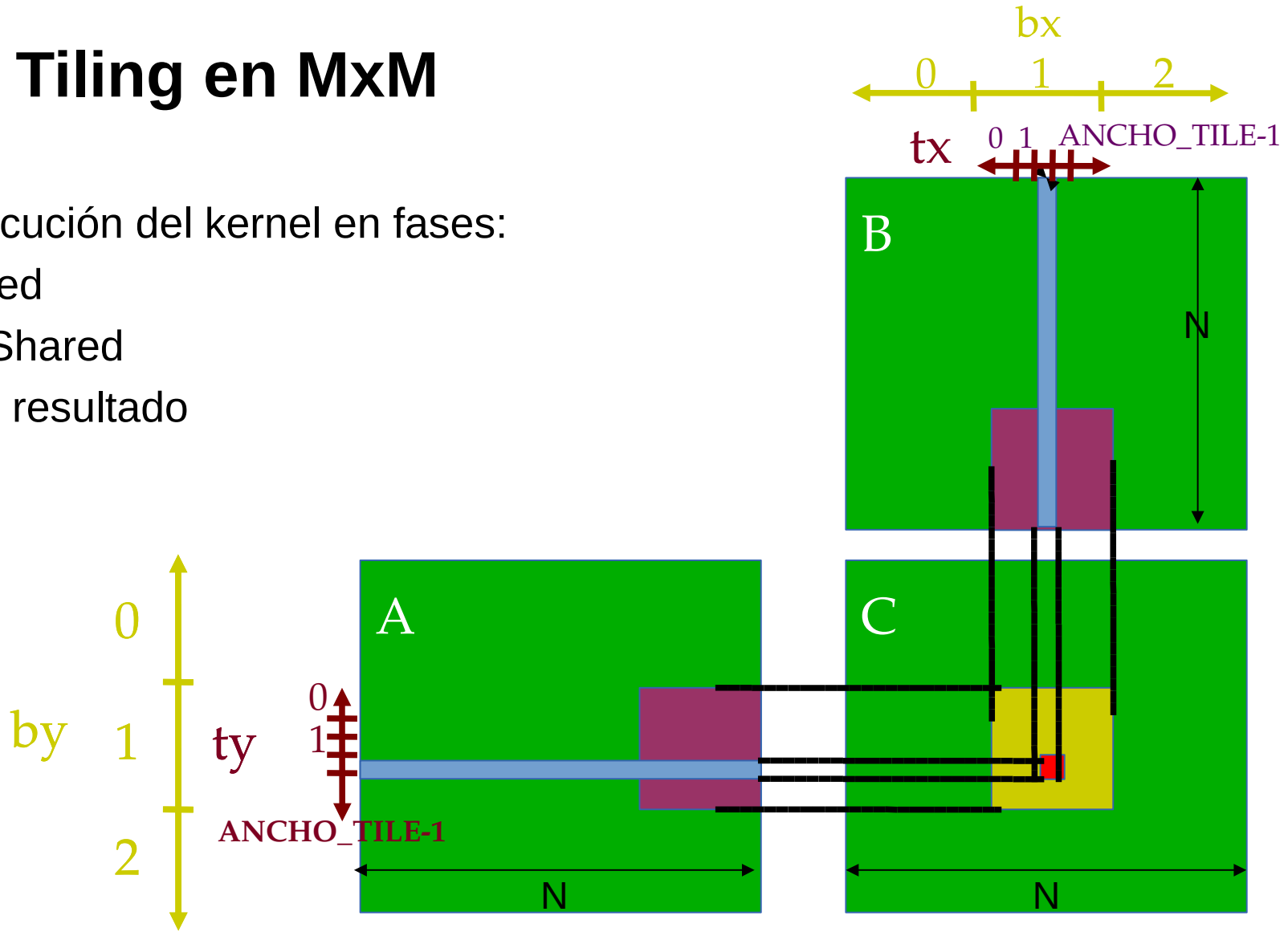
Tiling en MxM

- Dividimos la ejecución del kernel en fases:

  Cargamos a Shared

  Computamos en Shared

  Almancenamos el resultado



0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

A
8x8

X

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

B
8x8

=

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

C
8x8

B_y

0	0	1	2	3	4	5	6	7
	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23
	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39
	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55
	56	57	58	59	60	61	62	63

Fila= By*N*TAMBLOCK = By*8*2

B_y	t_y								
0	0	0	1	2	3	4	5	6	7
	1	8	9	10	11	12	13	14	15
1	0	16	17	18	19	20	21	22	23
	1	24	25	26	27	28	29	30	31
2	0	32	33	34	35	36	37	38	39
	1	40	41	42	43	44	45	46	47
3	0	48	49	50	51	52	53	54	55
	1	56	57	58	59	60	61	62	63

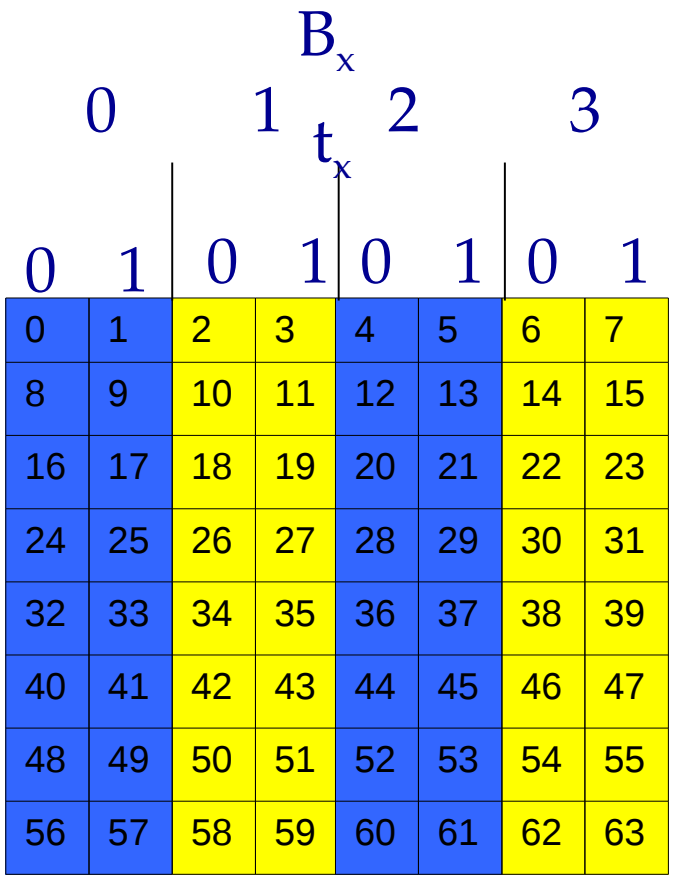
Fila += ty*N=
ty*8

B_x

0 1 2 3

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

Col= Bx*TAMBLOCK = Bx*2


B_x
0
1
2
3
t_x
0 1 0 1 0 1 0 1
0 1 2 3 4 5 6 7
8 9 10 11 12 13 14 15
16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31
32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47
48 49 50 51 52 53 54 55
56 57 58 59 60 61 62 63

Col+= tx

Las primeras consideraciones

- Cada thread block debe tener muchos hilos
  - Si el ANCHO_TILE es de 16 tenemos 16*16 = 256 threads
  - Si el ANCHO_TILE es de 32 tenemos 32*32 = 1024 threads
- Para un ancho de 16, cada bloque efectúa **2*256 = 512** lecturas de device memory.
  - Tendríamos **256 (hilos) * 2 madd * 16 tile** = 8192 madd operaciones
- Para un ancho de 32, cada bloque efectúa **2*1024 = 2048 float** lecturas de memoria global para realizar **1024* 2 * 32 = 65536** madd operaciones

Shared Memory y Threading

- Cada SM en G80 tiene 16KB o 48 KB de shared memory (Configurable)
  - Detalle de implementación
  - Para un ANCHO_TILE = 16, cada bloque usa 2*256*4B = 2KB de shared memory.
    - Cada SM podría tener hasta 8 bloques activos ejecutando.
    - Esto permite hasta 8*512 = 4,096 lecutas (2 por thread, 256 threads por bloque)
  - El siguiente ANCHO_TILE = 32 necesitaría 2*32*32*4B= 8KB de uso de shared memory por bloque, permitiendo 2 bloques activos al mismo tiempo.
- Usando tiles de 16x16, reducimos los accesos a memoria en un factor de 16.
  - El bandwidth de 86.4B/s ahora permite (86.4/4)*16 = 347.6 GFLOPS!

Efectos del Tiling (G80)