TALLER DE PROGRAMACIÓN SOBRE GPUS

Facultad de Informática – Universidad Nacional de La Plata

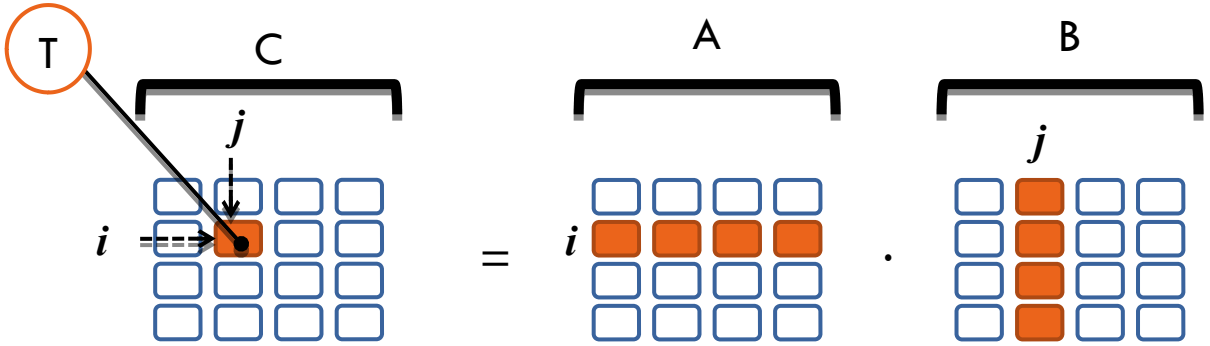
Dr. Adrián Pousa

Ejemplo: Optimización de multiplicación de matrices

Multiplicación de matrices NxN

$$c_{i,j} = \sum_{k=0}^{N-1} a_{i,k} \cdot b_{k,j}$$

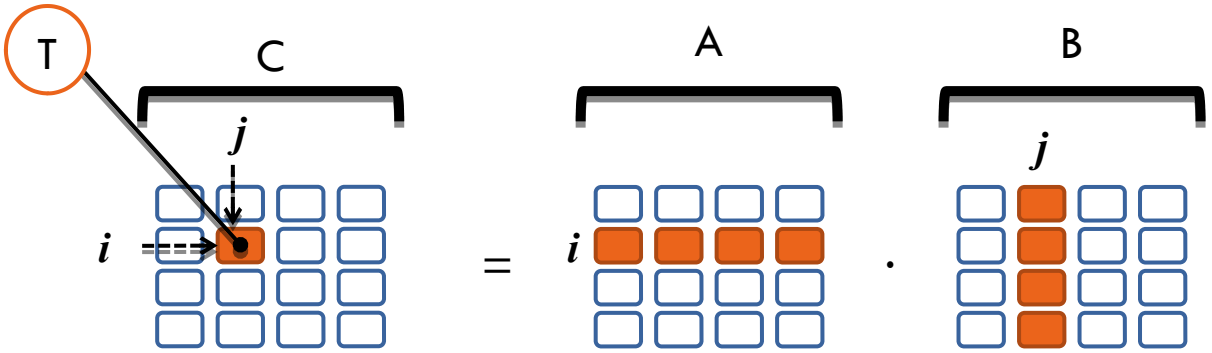
- La solución en GPU asigna a cada hilo el cálculo de una celda de la matriz resultado.
- Cada hilo CUDA necesita acceder a la fila i de A y la columna j de B.



Multiplicación de matrices NxN

$$c_{i,j} = \sum_{k=0}^{N-1} a_{i,k} \cdot b_{k,j}$$

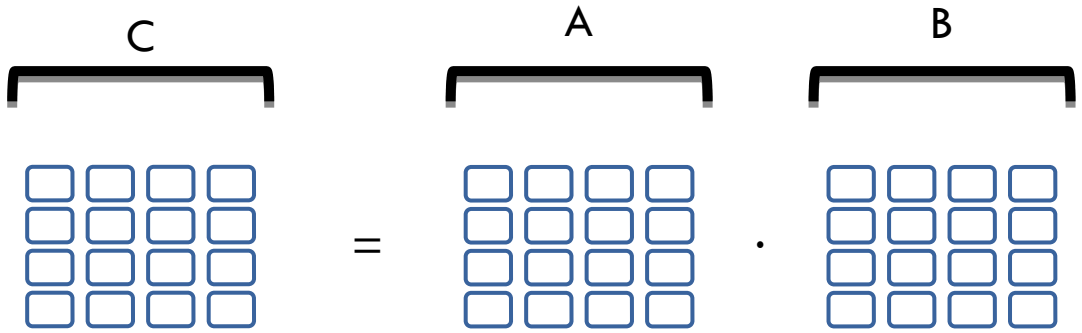
- Se utilizan además los beneficios que brinda CUDA al permitir crear Grids y Bloques bi-dimensionales de hilos.
- El grid se mapea a la estructura bi-dimensional de las matrices.



Multiplicación de matrices NxN

$$c_{i,j} = \sum_{k=0}^{N-1} a_{i,k} \cdot b_{k,j}$$

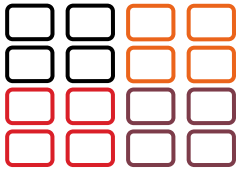
- Por ejemplo: se crean bloques de 2x2 hilos:



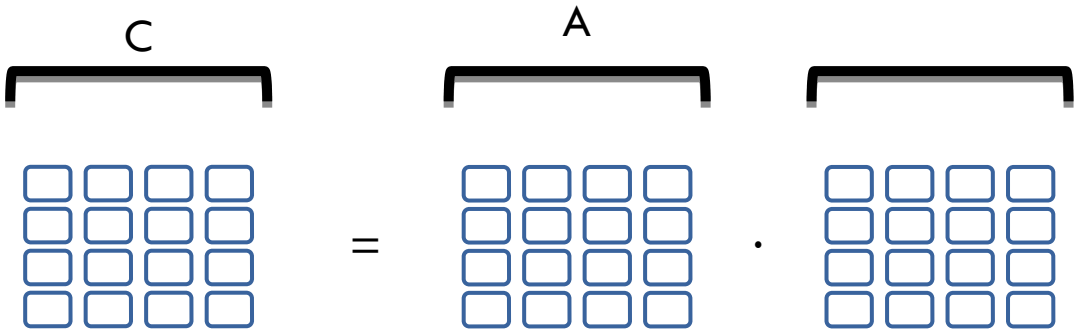
Multiplicación de matrices NxN

$$c_{i,j} = \sum_{k=0}^{N-1} a_{i,k} \cdot b_{k,j}$$

- Luego se crea un grid de 2x2 bloques:



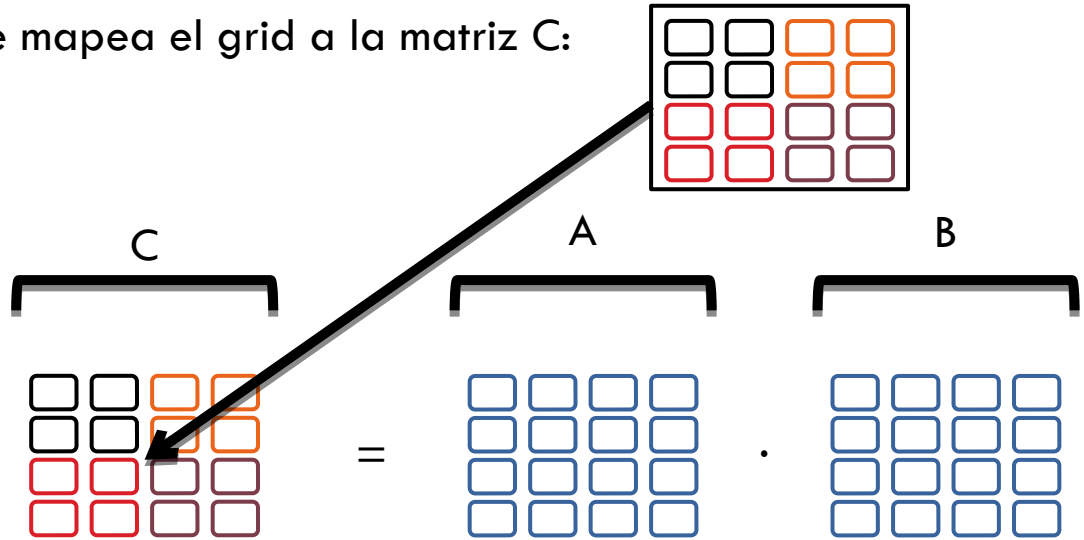
B



Multiplicación de matrices NxN

$$c_{i,j} = \sum_{k=0}^{N-1} a_{i,k} \cdot b_{k,j}$$

- Por último se mapea el grid a la matriz C:

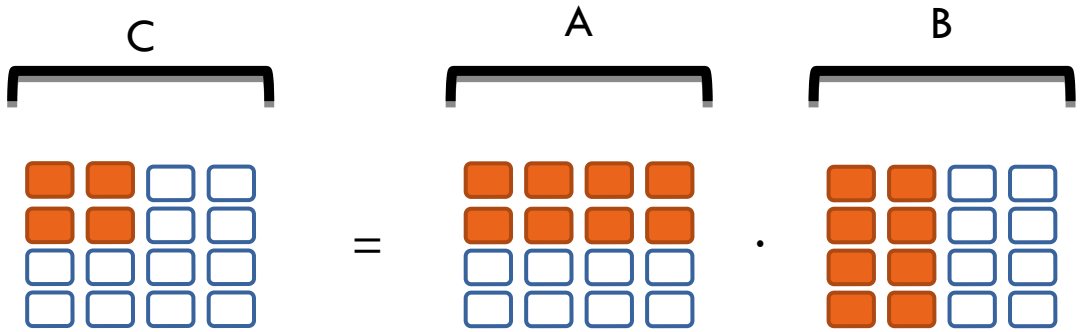


Se dice que cada bloque de threads de tamaño 2x2 calcula un "**TILE**" de tamaño 2x2 de la matriz resultado.

Multiplicación de matrices NxN

$$c_{i,j} = \sum_{k=0}^{N-1} a_{i,k} \cdot b_{k,j}$$

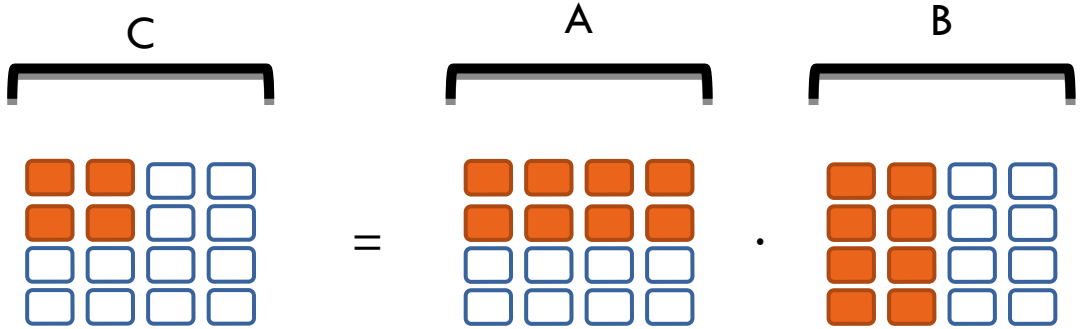
- Un bloque de hilos necesitará acceder a ciertas posiciones de las matrices A y B.



Multiplicación de matrices NxN

$$c_{i,j} = \sum_{k=0}^{N-1} a_{i,k} \cdot b_{k,j}$$

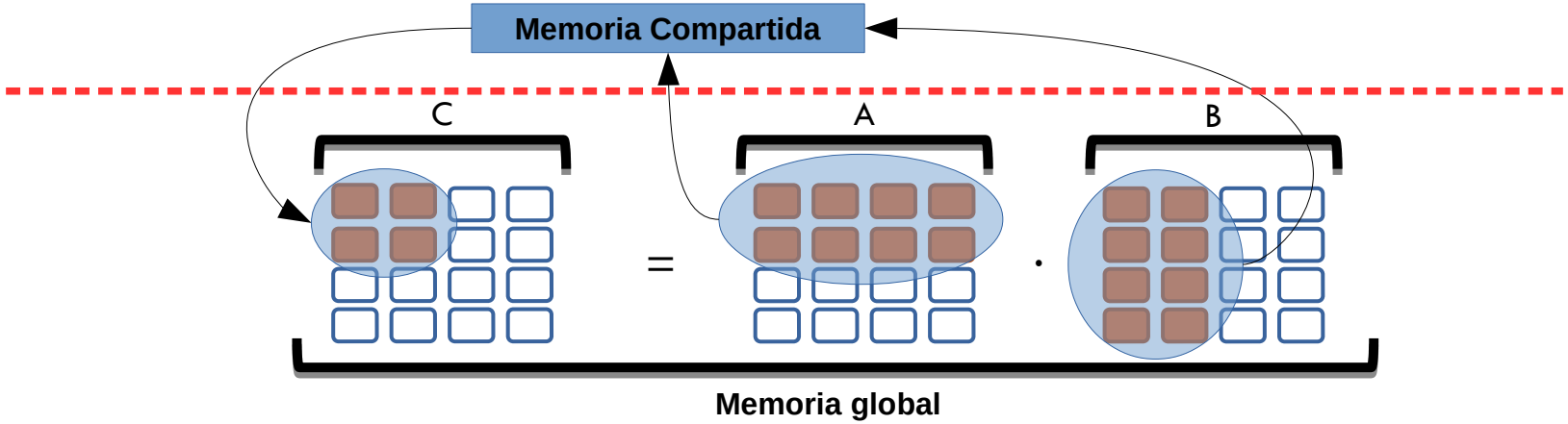
- Varios hilos de un mismo bloque acceden una y otra vez a las mismas posiciones.
- Estos accesos a memoria global resultan ser costosos.
- Es posible evitar que se acceda varias veces al mismo dato en memoria global utilizando una memoria más rápida como la memoria compartida.



Multiplicación de matrices NxN

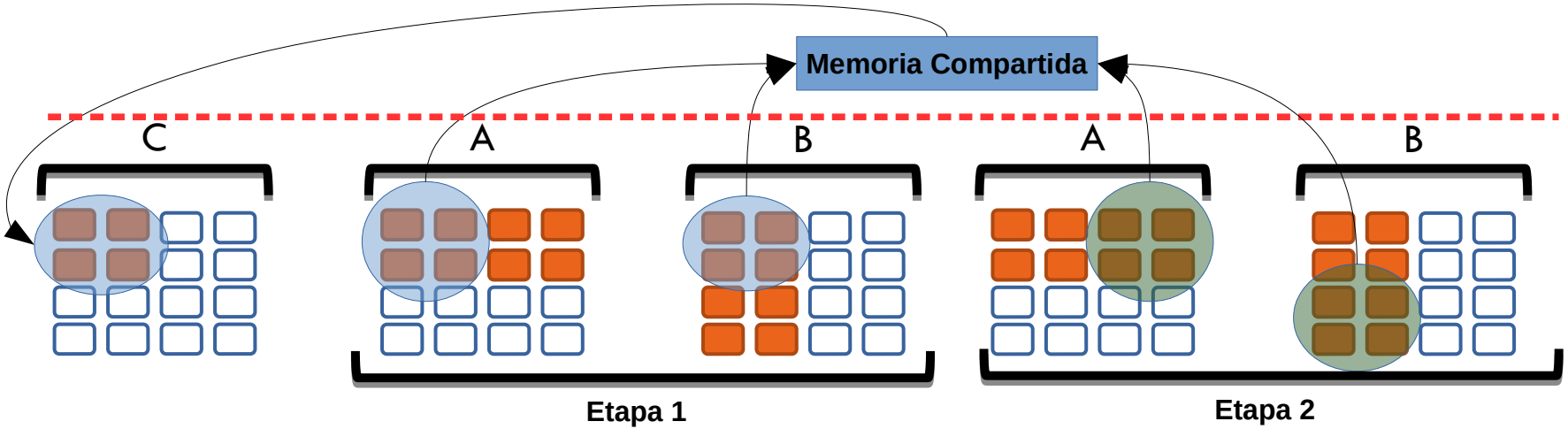
- La estrategia consiste en:

1) Traer, de forma coalescente, desde memoria global a la memoria compartida los datos necesarios correspondientes a las filas de A y columnas de B.
2) Procesarlos en memoria compartida.
3) Almacenar el resultado que reside en memoria compartida en la memoria global.



Multiplicación de matrices NxN

- Por las limitaciones de la memoria compartida y para maximizar el rendimiento las filas de A y las columnas de B se traen y calculan por etapas, de a TILES.
- Al completar el cálculo de todas las etapas se almacena el resultado que reside en memoria compartida en la memoria global.



Multiplicación de matrices NxN

Cada bloque bidimensional CUDA contiene BLOCK_SIZE * BLOCK_SIZE hilos

```
__global__ void mm_optimizado(float *C, float *A, float *B, int N){
 __shared As[BLOCK_SIZE][BLOCK_SIZE];
 __shared Bs[BLOCK_SIZE][BLOCK_SIZE];

 int fila = blockIdx.y * BLOCK_SIZE + threadIdx.y;
 int columna = blockIdx.x * BLOCK_SIZE + threadIdx.x;
 float c_temp=0.0;
   for(int etapa=0; etapa< N/BLOCK_SIZE; etapa++){
    As[threadIdx.y][threadIdx.x] = A[fila*N + etapa*BLOCK_SIZE + threadIdx.x];
    Bs[threadIdx.y][threadIdx.x] = B[(etapa*BLOCK_SIZE + threadIdx.y)*N + columna];
  __syncthreads();
   for( k=0 ; k<BLOCK_SIZE ; k++ )
    c_temp +=  As[fila,k] * B[k,columna];
  __syncthreads();
   }
       C[fila*N+columna]= c_temp;
   }
```

Espacio en memoria compartida para guardar cada TILE

Calcula la fila y columna sobre la que trabajará cada hilo

Lectura coalescente de memoria global a memoria compartida Cada hilo trae un dato por matriz.

Cálculo. En cada etapa deja resultados parciales en c_temp

Escritura coalescente
Cada hilo escribe su resultado almacenado en c_temp a la memoria global