TALLER DE PROGRAMACIÓN SOBRE GPUS

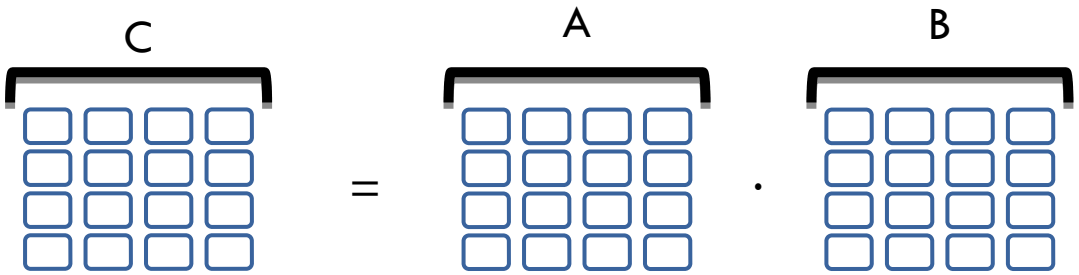
Facultad de Informática – Universidad Nacional de La Plata

Dr. Adrián Pousa

Ejemplo: Multiplicación de matrices

Multiplicación de matrices NxN

- Dadas dos matrices A y B de NxN elementos.



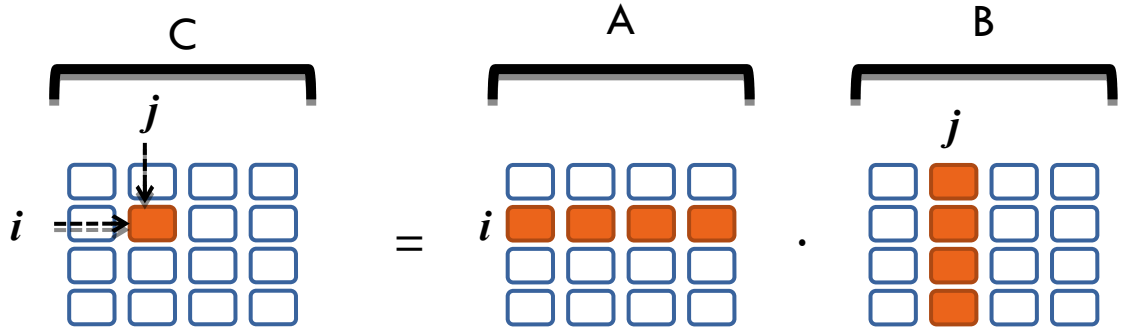
- La multiplicación A.B retorna una matriz resultado C donde cada elemento se calcula como:

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

Multiplicación de matrices NxN

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

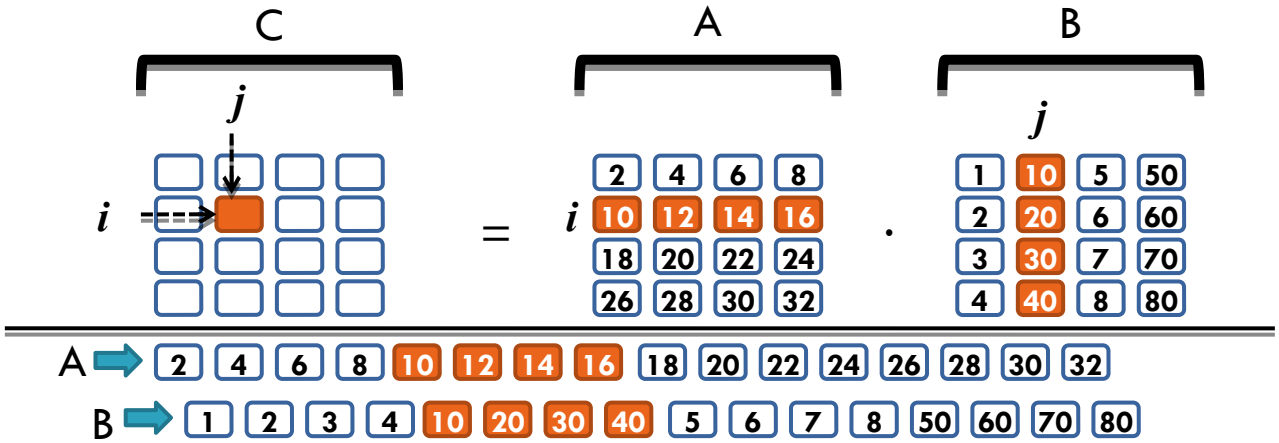
- Para calcular la posición (i,j) de la matriz resultante C es necesario procesar la fila i de A y la columna j de B.



Multiplicación de matrices NxN

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

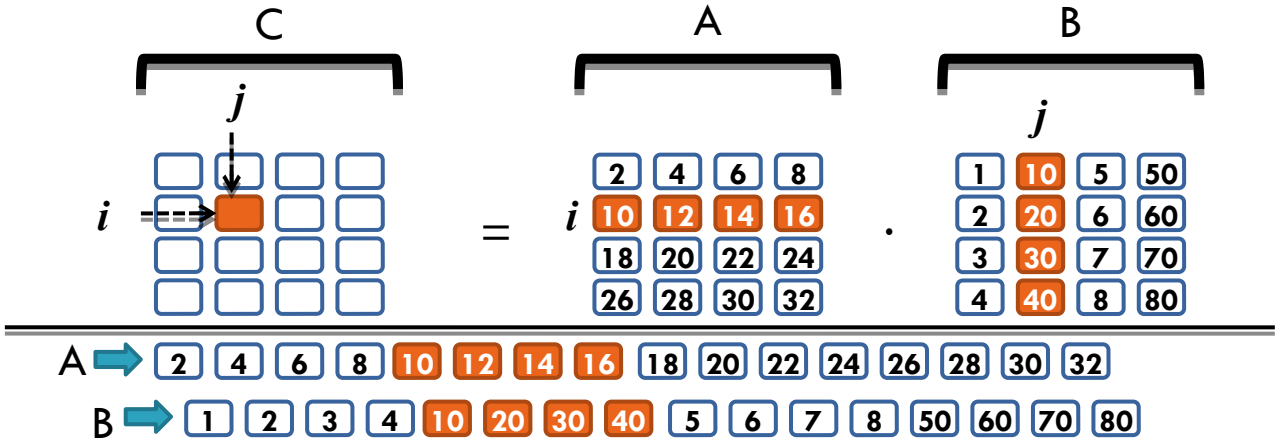
- Para mejorar la localidad de caché, las matrices se trabajan como arreglos. Luego, A y C se almacenan en memoria por filas, y B por columnas.



Multiplicación de matrices NxN

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

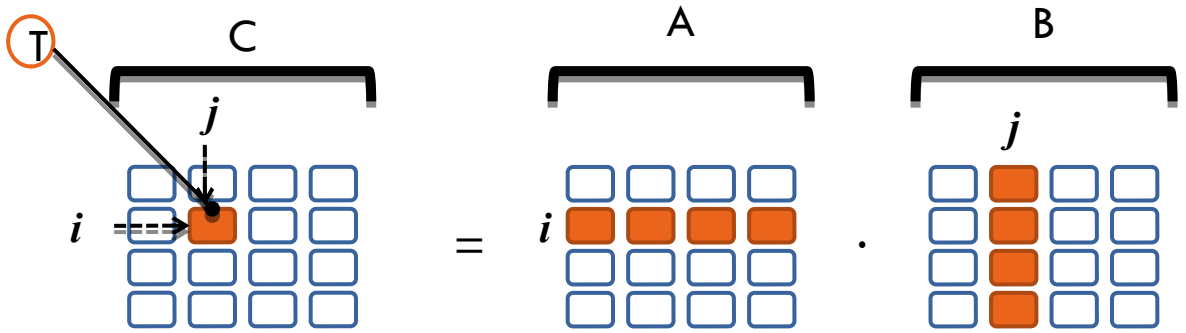
- Para acceder a la posición (i,j) de una matriz ordenada por filas: `M[i*N+j]`
- Para acceder a la posición (i,j) de una matriz ordenada por columnas: `M[i+j*N]`



Multiplicación de matrices NxN

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

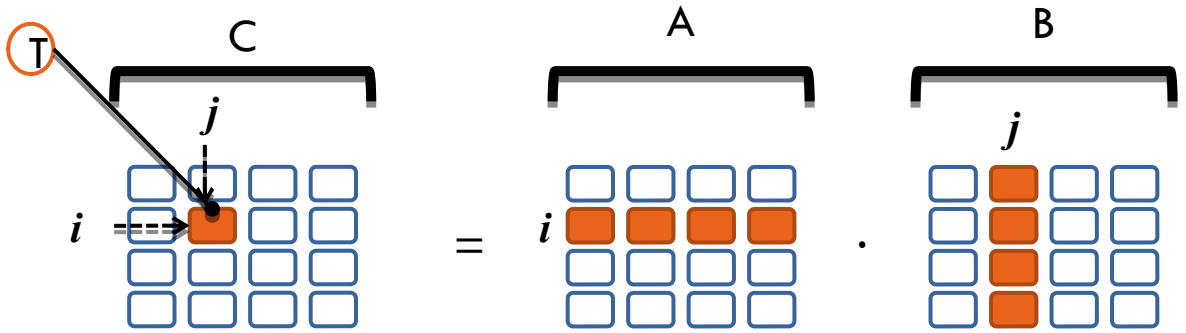
- La solución en GPU asigna a cada hilo el cálculo de una celda de la matriz resultado.
- Cada hilo CUDA necesita acceder a la fila i de A y la columna j de B.



Multiplicación de matrices NxN

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

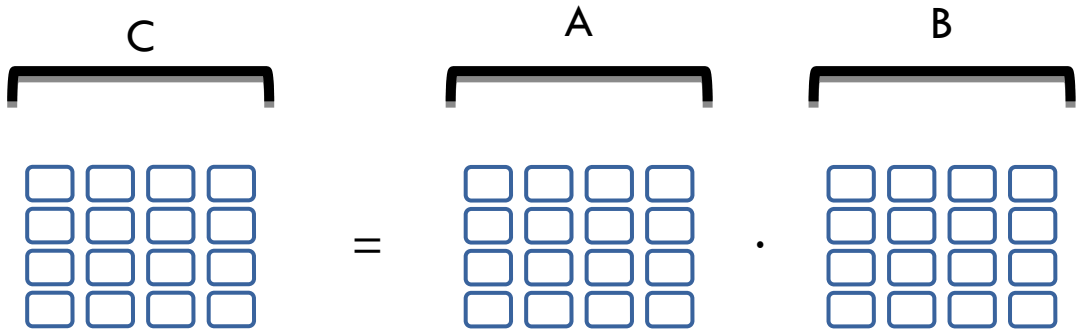
- Se utilizan además los beneficios que brinda CUDA al permitir crear Grids y Bloques bi-dimensionales de hilos.
- El grid se mapea a la estructura bi-dimensional de las matrices.



Multiplicación de matrices NxN

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

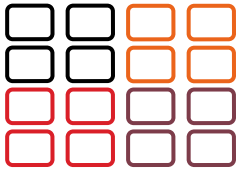
- Por ejemplo: se crean bloques de 2x2 hilos: 



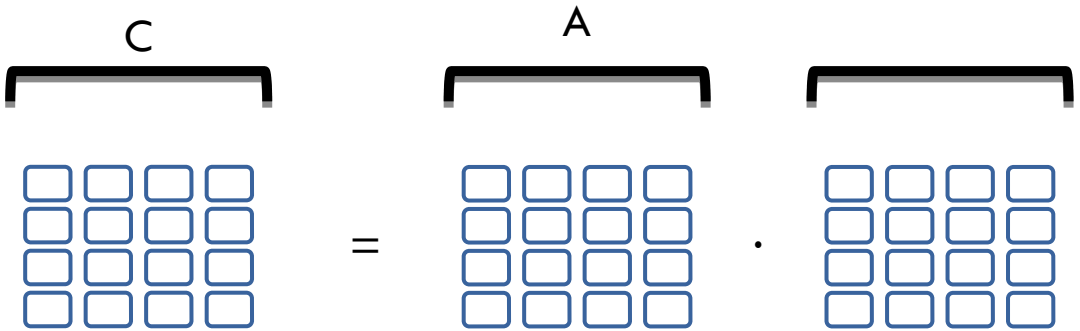
Multiplicación de matrices NxN

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

- Luego se crea un grid de 2x2 bloques:



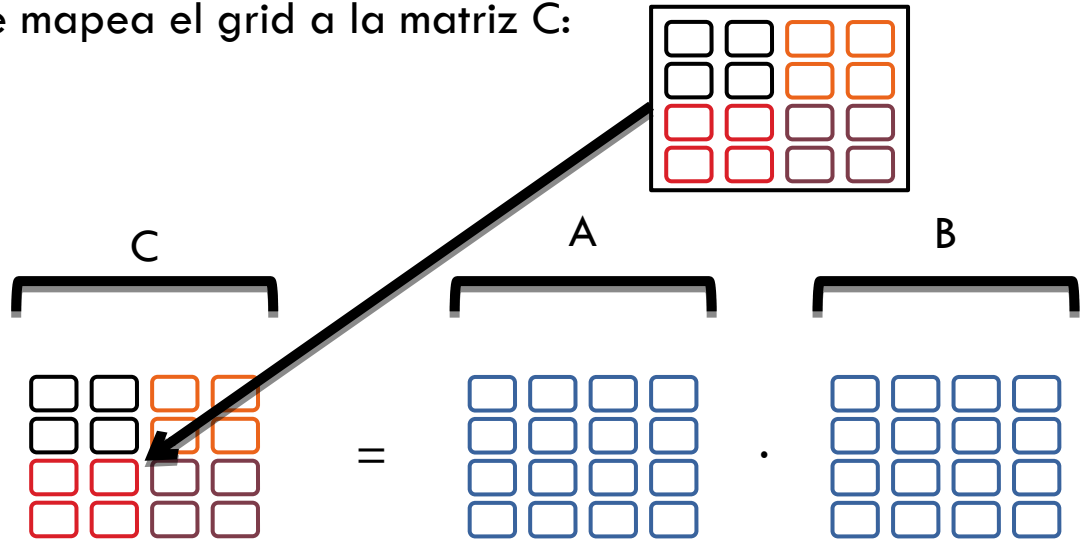
B



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:

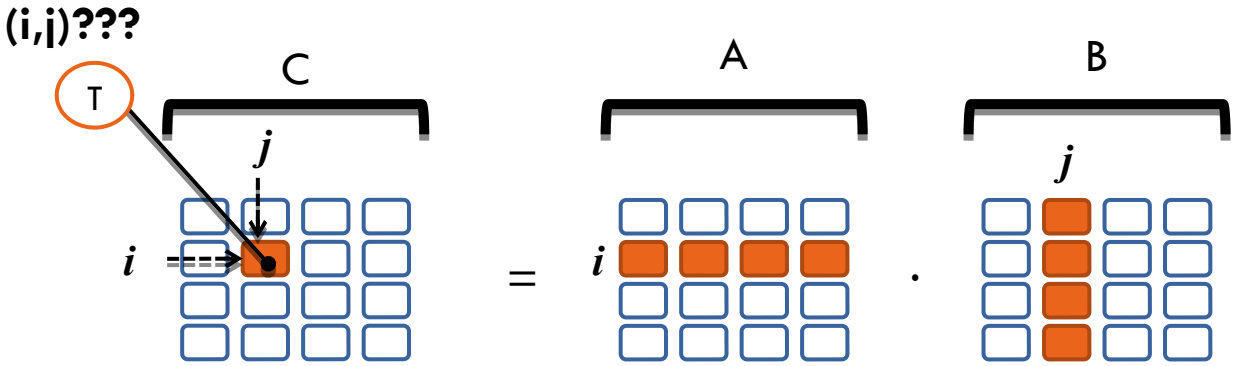


Se dice que cada bloque de threads de tamaño 2x2 calcula un "TILE" de tamaño 2x2 de la matriz resultado.

Multiplicación de matrices NxN

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

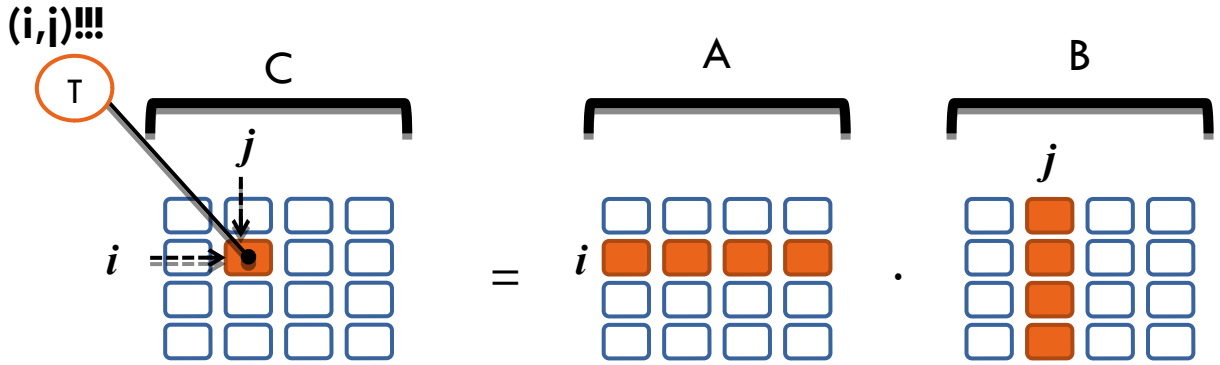
- **¿Cómo conoce cada hilo la posición a calcular?**



Multiplicación de matrices NxN

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

- **Utilizando las variables built-in provistas por CUDA.**

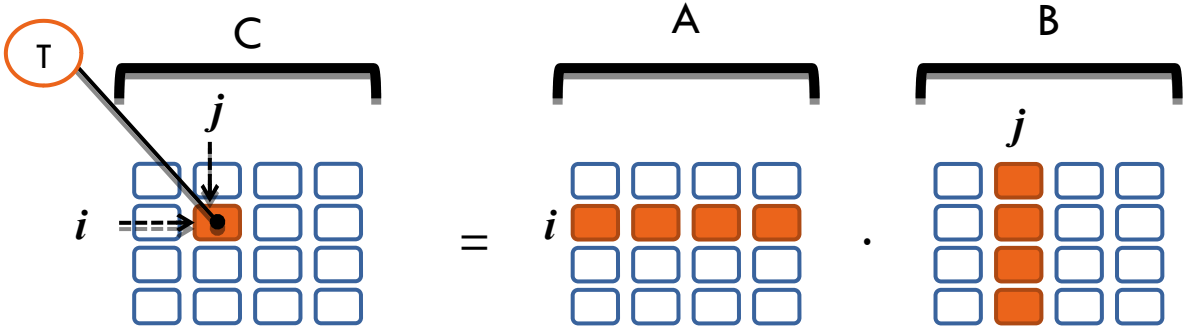


Multiplicación de matrices NxN

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

- **Utilizando las variables built-in provistas por CUDA.**

```
i = blockIdx.y*blockDim.y + threadIdx.y;
j = blockIdx.x*blockDim.x + threadIdx.x;
```

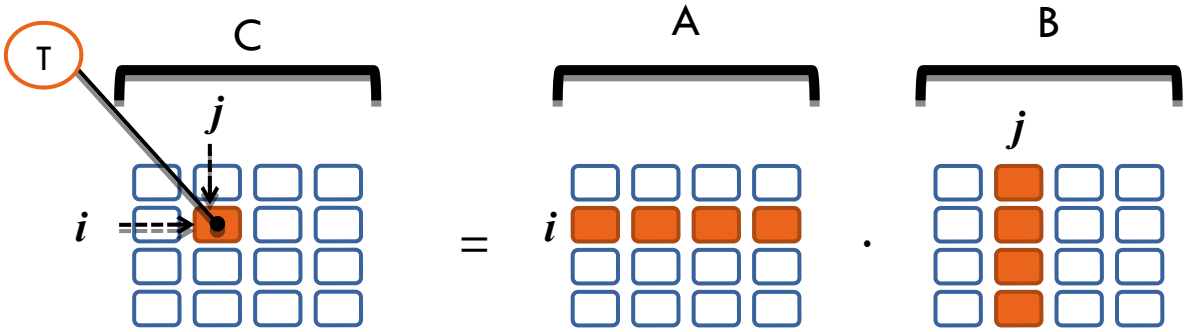


Multiplicación de matrices NxN

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

- El código que calcula la posición (i,j) de C:

```
__global__ mm(int *C, int *A, int *B,int N){
int i = blockIdx.y*blockDim.y + threadIdx.y;
int j = blockIdx.x*blockDim.x + threadIdx.x;
int k;
    for( k=0 ; k<N ; k++ )
      C[i*N+j] +=  A[i*N+k] * B[k+j*N];
}
```



Multiplicación de matrices NxN

Si la dimensión del grid no es proporcional al tamaño de la matriz (más hilos que posiciones a calcular de la matriz C) existen hilos que NO deberían trabajar.
Se agrega el siguiente if al código:

```
…
if ( i<N && j<N )
      for( k=0 ; k<N ; k++ )
            C[i*N+j] +=  A[i*N+k] * B[k+j*N];
…
```