Ejemplo de aplicación Multiplicación de matrices en CUDA

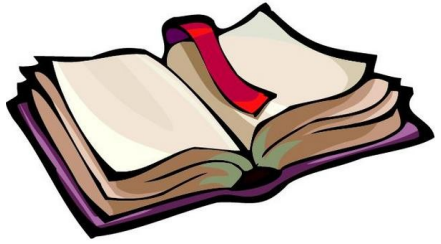
Universidad Nacional de La Plata

Facultad de Informática

Prof. Dr. Adrian Pousa

Agenda

I. Introducción a la multiplicación de matrices
II. Organización de matrices en memoria
III. Solución en GPU
IV. Optimización usando memoria compartida



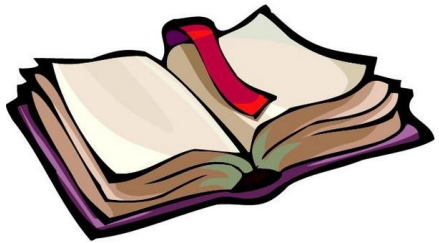
Agenda

I. Introducción a la multiplicación de matrices

II. Organización de matrices en memoria

III. Solución en GPU

IV. Optimización usando memoria compartida

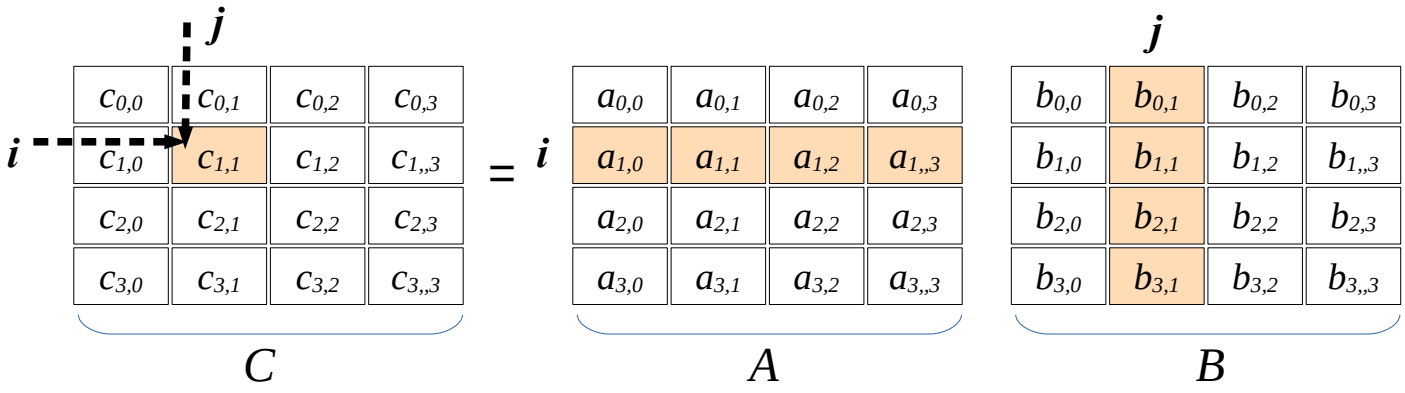


Introducción a la Multiplicación de matrices

- Dadas dos matrices A y B de NxN elementos. La multiplicación AB retorna una matriz resultado C donde cada elemento se calcula como:

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

- 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.



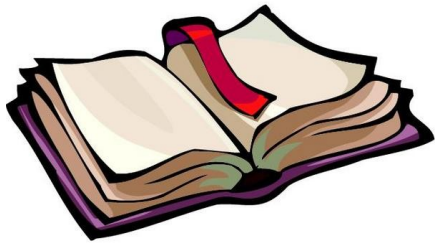
Agenda

I. Introducción a la multiplicación de matrices

II. Organización de matrices en memoria

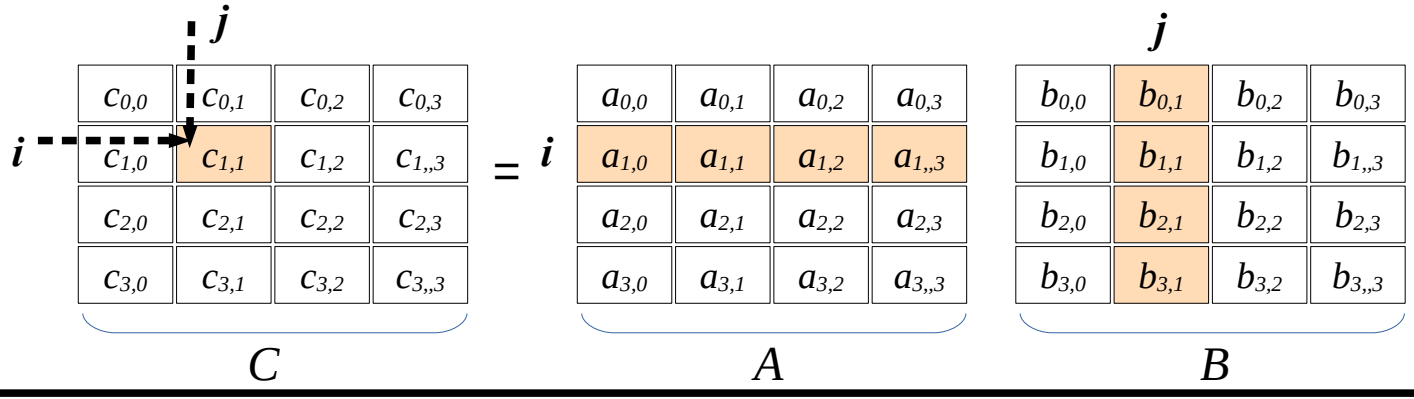
III. Solución en GPU

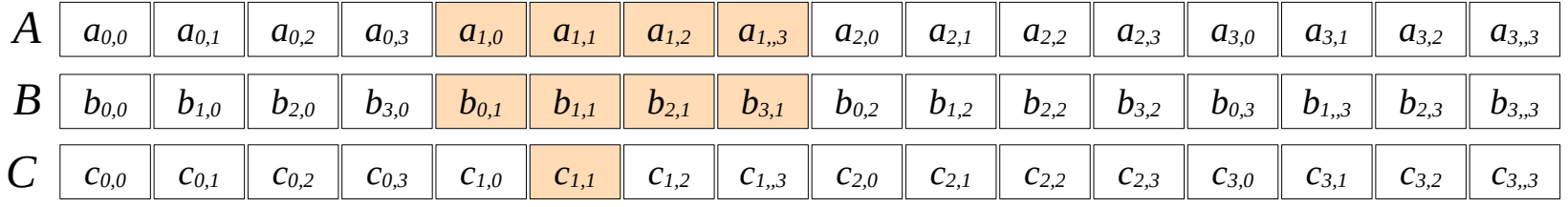
IV. Optimización usando memoria compartida



Multiplicación de matrices
Organización de matrices en memoria - Localidad

- Para mejorar la localidad de caché, las matrices se trabajan como arreglos ordenados en memoria de acuerdo al patrón en el cual se acceden:
  - Las matrices A y C se almacenan en memoria por filas, y B por columnas.





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

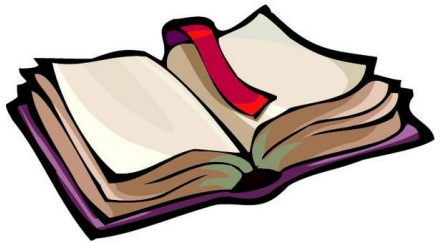
Agenda

I. Introducción a la multiplicación de matrices

II. Organización de matrices en memoria

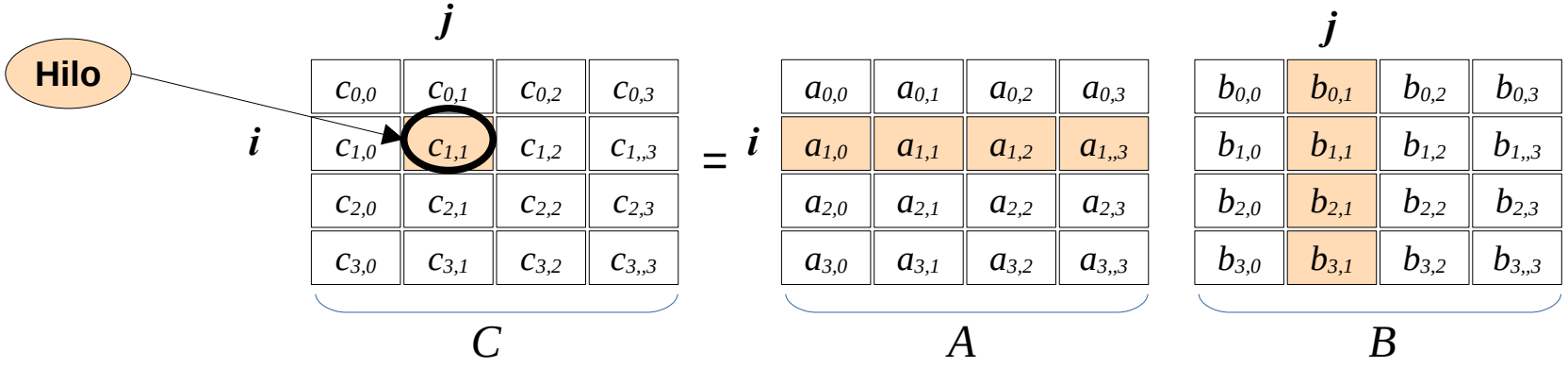
III. Solución en GPU

IV. Optimización usando memoria compartida

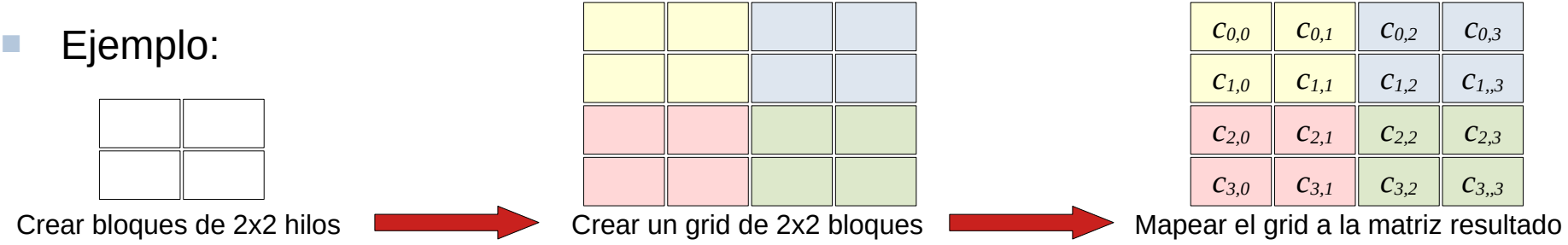


Multiplicación de matrices
Solución en GPU

- 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.

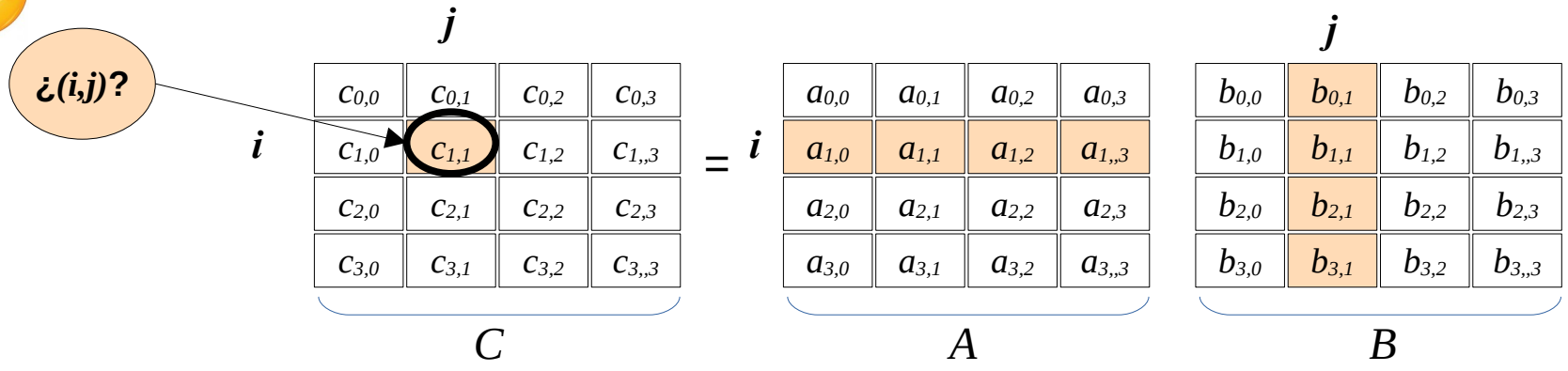


- CUDA permite crear Grids y Bloques bi-dimensionales de hilos que pueden mapearse a la estructura bi-dimensional de las matrices:
  - Ejemplo:



Multiplicación de matrices
Solución en GPU

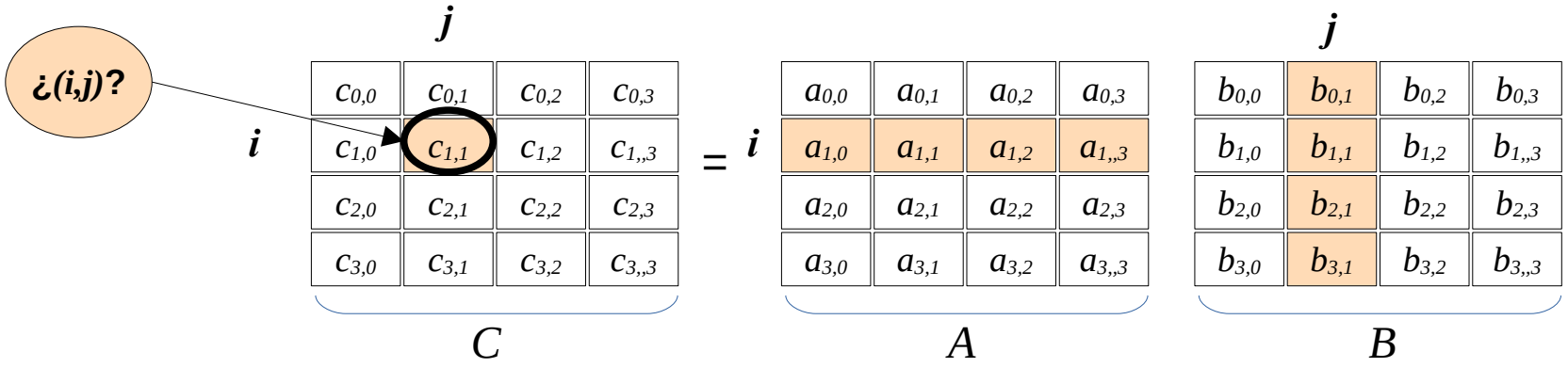
¿Cómo conoce cada hilo CUDA la posición que debe calcular?



Multiplicación de matrices
Solución en GPU

¿Cómo conoce cada hilo CUDA la posición que debe calcular?

Utilizando las variables built-in!!!



- **En el Kernel**: código que calcula la posición *(i,j)*:

```
__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
Solución en GPU

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:

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

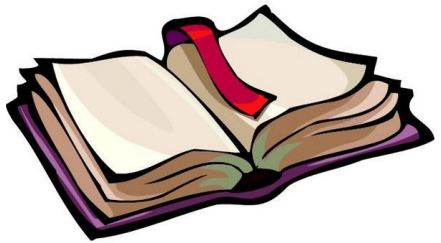
Agenda

I. Introducción a la multiplicación de matrices

II. Organización de matrices en memoria

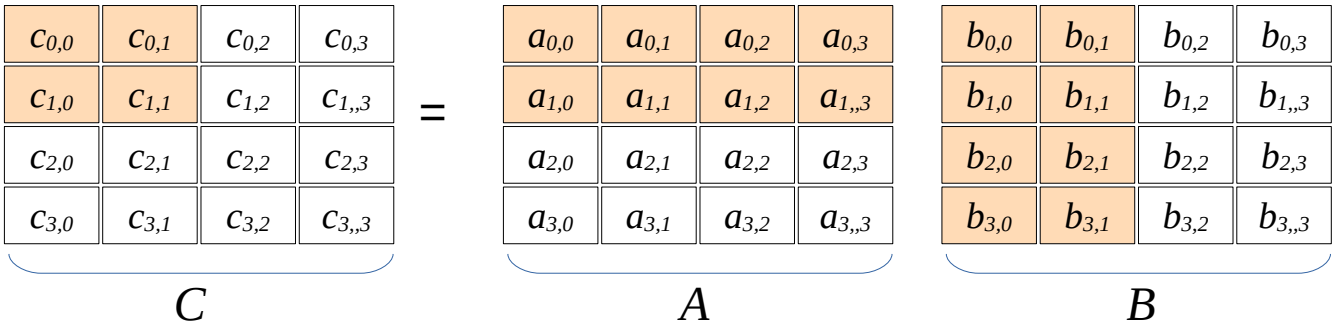
III. Solución en GPU

IV. Optimización usando memoria compartida



Multiplicación de matrices
Solución GPU - Problemas

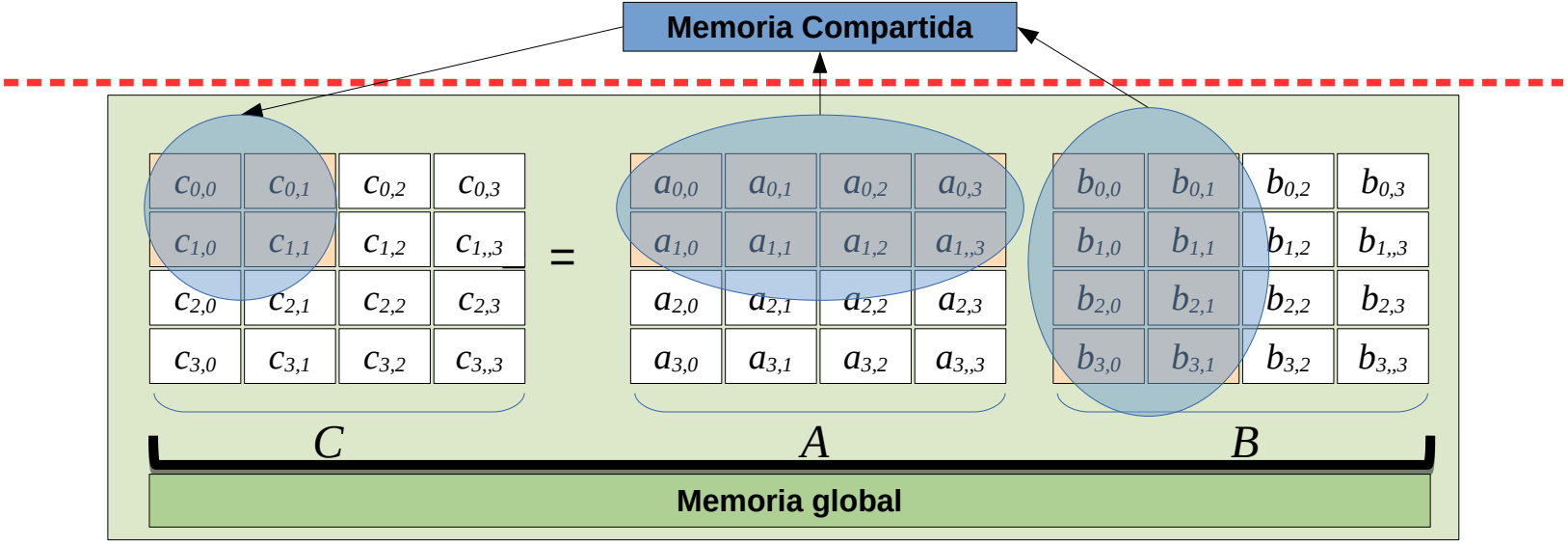
- Un bloque de hilos necesitará acceder a ciertas posiciones de las matrices A y B.



  - Varios hilos de un mismo bloque acceden varias veces a las mismas posiciones en memoria global.
  - Estos accesos a memoria global resultan ser costosos.
  - Podemos evitar la repetición de accesos al mismo dato en memoria global utilizando una memoria más rápida como la memoria compartida.

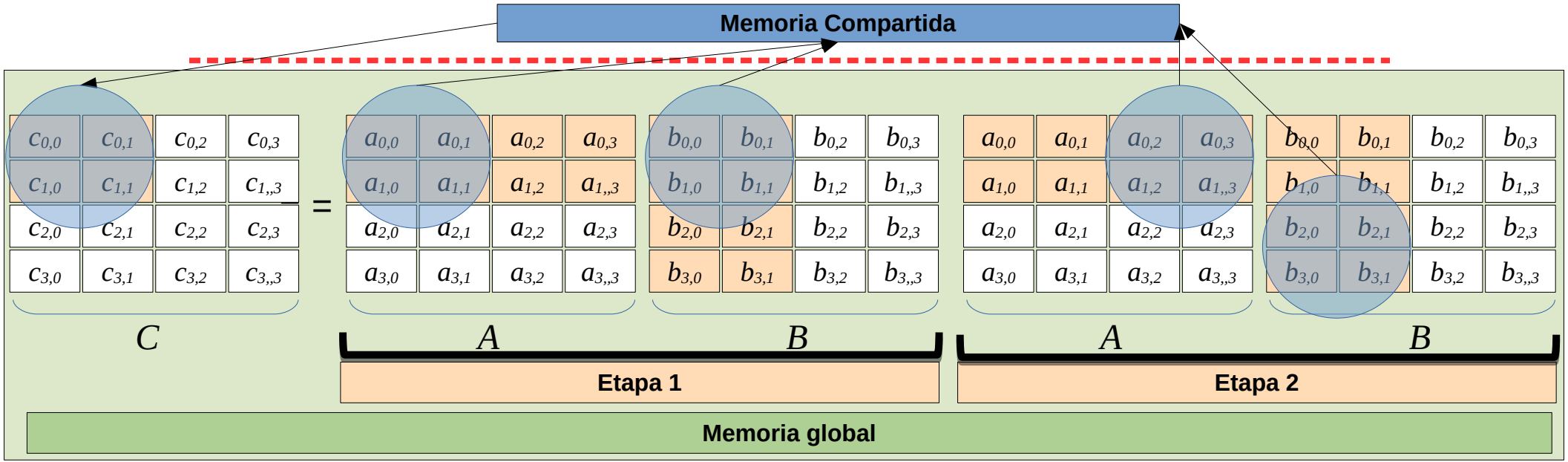
Multiplicación de matrices
Mejoras por el uso de memoria compartida

- La estrategia consiste en:
  - Traer, de forma coalescente, desde memoria global a memoria compartida, los datos necesarios correspondientes a las filas de A y columnas de B.
  - Procesarlos en memoria compartida.
  - El resultado en memoria compartida almacenarlo en la memoria global.



Multiplicación de matrices
Mejoras por el uso de memoria compartida

- 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
Mejoras por el uso de memoria compartida - Implementación

- Suponemos que 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