Ejercicios Sesión 4
Memorias CUDA

Albert García García <agarcia@dtic.ua.es>

Sergio Orts Escolano <sorts@dtic.ua.es>

José García Rodríguez <jgarcia@dtic.ua.es>

Universidad de Alicante

Departamento de tecnología informática y computación

Ejercicio 1

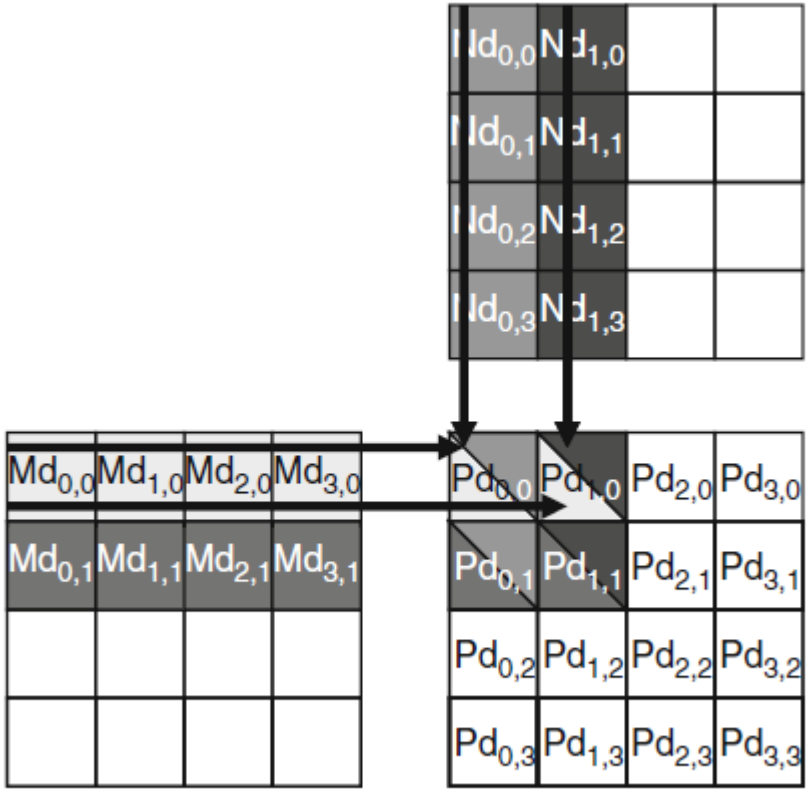
- Considera el cálculo suma de matrices donde cada elemento de la matriz resultado es la suma de los elementos correspondientes de las matrices de entrada. ¿Es posible utilizar memoria compartida para reducir el acceso a memoria global?. Pista: Analiza los elementos accedidos por cada hilo. ¿Existe algún patrón de acceso entre los hilos?
- Considera el cálculo multiplicación de matrices. ¿Es posible utilizar memoria compartida para este caso, reduciendo el acceso a memoria global?.

Ejercicio 1. Solución

- En la suma de matrices cada hilo únicamente accede a sus elementos correspondientes y no existe patrón de acceso compartido a elementos vecinos entre hilos de un mismo bloque. Por lo tanto, la utilización de memoria compartida no reduce el tiempo de acceso a memoria global.

- En el caso de la multiplicación de matrices existe un patrón de acceso a datos, donde varios elementos de las matrices de entrada son utilizados de forma compartida por varios hilos de cómputo. Con el almacenamiento de estos en memoria compartida el tiempo de acceso a memoria global se reduce de forma considerable. Para llevar a cabo el uso de memoria global se hace uso de la técnica de tiling vista en clase. El tiempo de acceso se reduce de forma lineal respecto al tamaño del TILE utilizado.

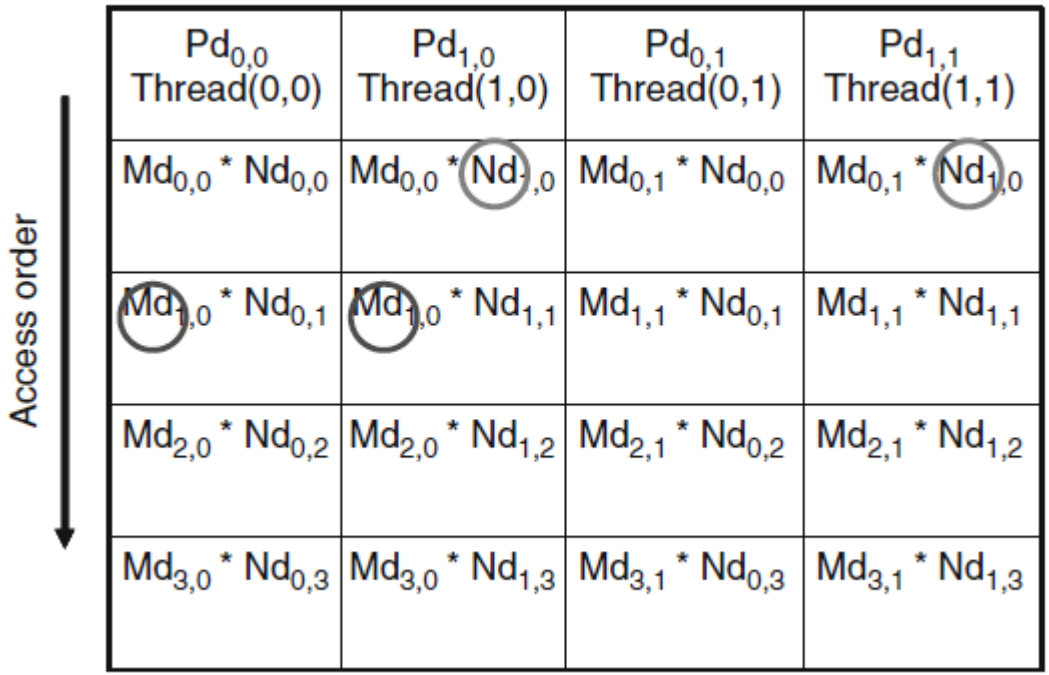
Ejercicio 1. Solución



Patrón de acceso a datos en memoria local.

Se observa como distintos hilos acceden a elementos repetidos.

Ejercicio 1. Solución



Orden de acceso de cada hilo a las matrices a sumar.

Ejercicio 2.

- Qué tipo de comportamiento incorrecto puede ocurrir si olvidamos utilizar la instrucción `__syncthreads()` en el kernel que se muestra a continuación. Existen dos llamadas a la función `__syncthreads()` justifica cada una de ellas.

- Pista: El kernel proporcionado multiplica dos matrices haciendo uso de la memoria compartida y la técnica de TILING vista en el ejercicio anterior.

```
__global__ void MatrixMulKernel(float* Md, float* Nd, float* Pd, int Width)
{
1.   __shared__float Mds[TILE_WIDTH][TILE_WIDTH];
2.   __shared__float Nds[TILE_WIDTH][TILE_WIDTH];

3.   int bx = blockIdx.x;  int by = blockIdx.y;
4.   int tx = threadIdx.x; int ty = threadIdx.y;

// Identify the row and column of the Pd element to work on
5.   int Row = by * TILE_WIDTH + ty;
6.   int Col = bx * TILE_WIDTH + tx;

7.   float Pvalue = 0;
// Loop over the Md and Nd tiles required to compute the Pd element
8.  for (int m = 0; m < Width/TILE_WIDTH; ++m) {

// Collaborative loading of Md and Nd tiles into shared memory
9.    Mds[ty][tx] = Md[Row*Width  + (m*TILE_WIDTH +  tx)];
10.   Nds[ty][tx] = Nd[(m*TILE_WIDTH  + ty)*Width + Col];
11.   __syncthreads();

12.   for (int k = 0; k < TILE_WIDTH; ++k)
13.     Pvalue += Mds[ty][k] * Nds[k][tx];
14.   __syncthreads();
    }
15. Pd[Row*Width + Col] = Pvalue;
}
```

Ejercicio 2. Solución

- La primera llamada a `__syncthreads` del código se utiliza para asegurar que todos los hilos de un mismo bloque han copiado el elemento correspondiente de las matrices de entrada antes de empezar a computar el producto matricial.

- La segunda llamada a `__syncthreads` asegura que todos los hilos han terminado de cálcular el producto parcial del TILE en el que se encuentran y antes de pasar al siguiente TILE todos deben haber alcanzado ese punto de ejecución.

Ejercicio 2. Solución

	Phase 1			Phase 2		
$T_{0,0}$	$\mathbf{Md_{0,0}}$ ↓ $Mds_{0,0}$	$\mathbf{Nd_{0,0}}$ ↓ $Nds_{0,0}$	$PValue_{0,0}$ += $Mds_{0,0}$*$Nds_{0,0}$ + $Mds_{1,0}$*$Nds_{0,1}$	$\mathbf{Md_{2,0}}$ ↓ $Mds_{0,0}$	$\mathbf{Nd_{0,2}}$ ↓ $Nds_{0,0}$	$PValue_{0,0}$ += $Mds_{0,0}$*$Nds_{0,0}$ + $Mds_{1,0}$*$Nds_{0,1}$
$T_{1,0}$	$\mathbf{Md_{1,0}}$ ↓ $Mds_{1,0}$	$\mathbf{Nd_{1,0}}$ ↓ $Nds_{1,0}$	$PValue_{1,0}$ += $Mds_{0,0}$*$Nds_{1,0}$ + $Mds_{1,0}$*$Nds_{1,1}$	$\mathbf{Md_{3,0}}$ ↓ $Mds_{1,0}$	$\mathbf{Nd_{1,2}}$ ↓ $Nds_{1,0}$	$PValue_{1,0}$ += $Mds_{0,0}$*$Nds_{1,0}$ + $Mds_{1,0}$*$Nds_{1,1}$
$T_{0,1}$	$\mathbf{Md_{0,1}}$ ↓ $Mds_{0,1}$	$\mathbf{Nd_{0,1}}$ ↓ $Nds_{0,1}$	$PdValue_{0,1}$ += $Mds_{0,1}$*$Nds_{0,0}$ + Mds_{11}*$Nds_{0,1}$	$\mathbf{Md_{2,1}}$ ↓ $Mds_{0,1}$	$\mathbf{Nd_{0,3}}$ ↓ $Nds_{0,1}$	$PdValue_{0,1}$ += $Mds_{0,1}$*$Nds_{0,0}$ + $Mds_{1,1}$*$Nds_{0,1}$
$T_{1,1}$	$\mathbf{Md_{1,1}}$ ↓ $Mds_{1,1}$	$\mathbf{Nd_{1,1}}$ ↓ $Nds_{1,1}$	$PdValue_{1,1}$ += $Mds_{0,1}$*$Nds_{1,0}$ + $Mds_{1,1}$*$Nds_{1,1}$	$\mathbf{Md_{3,1}}$ ↓ $Mds_{1,1}$	$\mathbf{Nd_{1,3}}$ ↓ $Nds_{1,1}$	$PdValue_{1,1}$ += $Mds_{0,1}$*$Nds_{1,0}$ + $Mds_{1,1}$*$Nds_{1,1}$

time →

¿Preguntas?