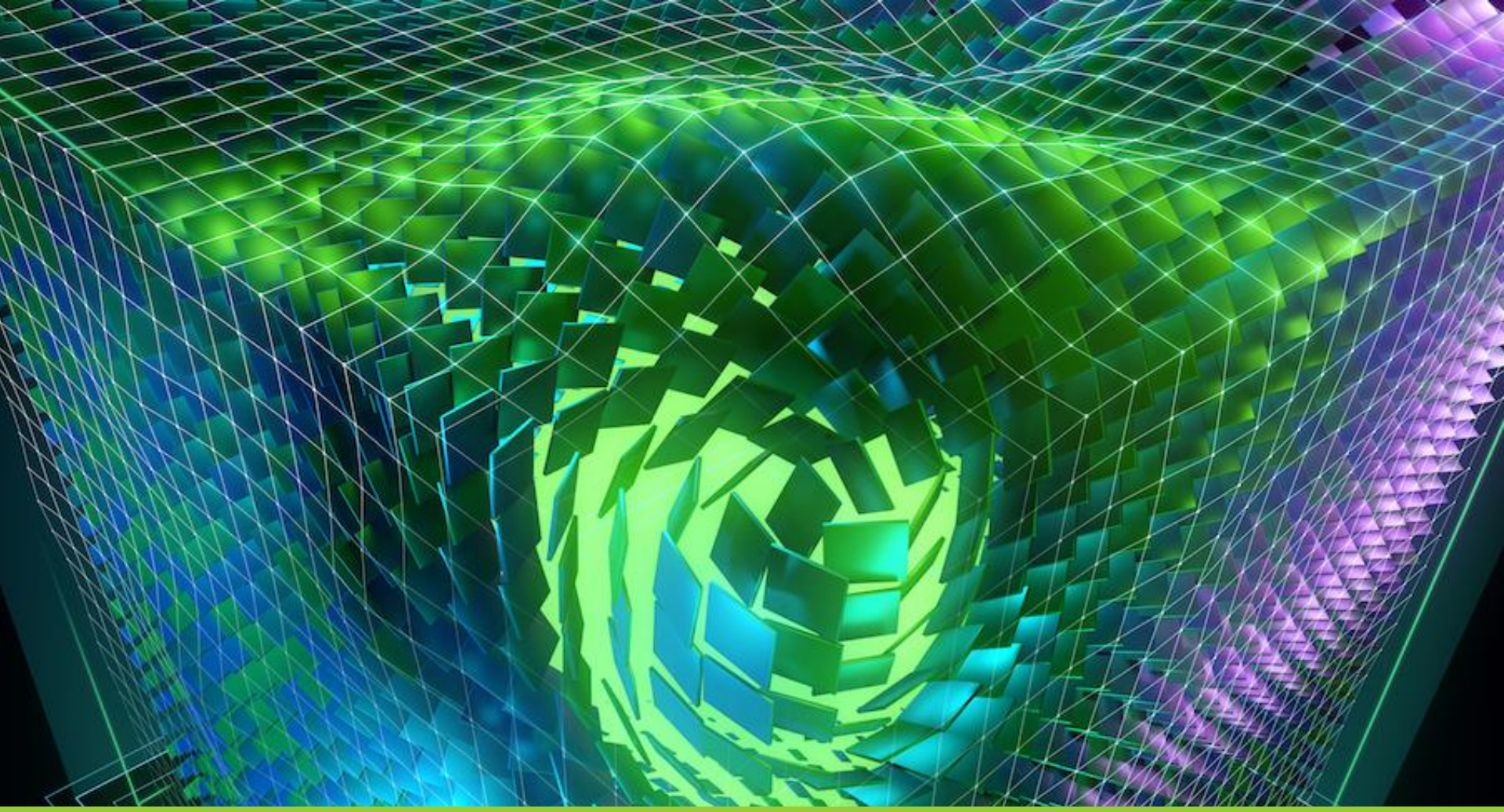


INF351 – Computación de Alto Desempeño

Distribución de Trabajo

PROF. ÁLVARO SALINAS

Distribución Lógica y Física

```
#include "../cpp/include/files.h"
#include "../include/structs.h"
#include <iostream>
#include <iomanip>
#include <math.h>
#include <string>
#include <cuda_runtime.h>
#include <fstream>
#include <time.h>

__global__ void wKernel(int Lx, int Ly, const prec* __restrict__ h,
  const prec* __restrict__ b, prec* w) {

  int i = threadIdx.x + blockIdx.x*blockDim.x;
  if (i < Lx*Ly) {
      w[i] = h[i] + b[i];
  }
```

Kernel

Funciones de C definidas de forma especial para su ejecución en paralelo.

Se declaran con el especificador __global__ y el nivel de paralelismo se define en las llamadas utilizando una nueva sintaxis de configuración de ejecución <<< … >>>.

```
// Kernel definition
__global__ void VecAdd(float* A, float* B, float* C) {
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}

int main() {
    ...
    // Kernel invocation
    VecAdd<<<1, N>>>(A, B, C);
    ...
}
```

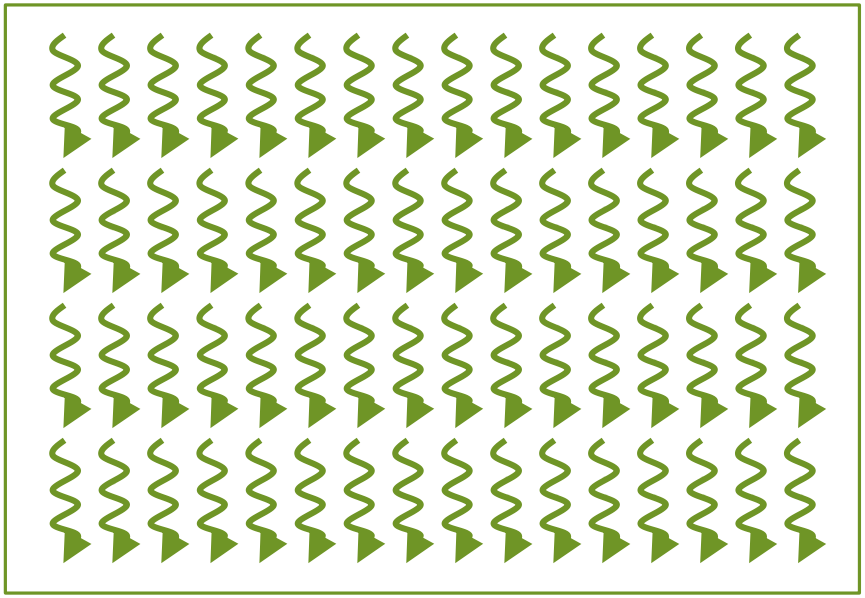
Thread

Un CUDA thread (hilo o hebra) equivale a un subproceso.

Cada thread corresponde a una ejecución de un kernel.

Se definen y diferencian entre sí por un identificador, el cual puede ser utilizado dentro de un kernel, generalmente como índice de un arreglo.

Block



Un thread block corresponde a un conjunto de threads que se asigna a un SM para ser procesado.

La cantidad de threads por bloque es definida por el programador.

Varios bloques pueden ejecutar un kernel en paralelo en uno o más SM.

La ejecución de un kernel finaliza cuando han sido procesados todos los bloques especificados en su invocación.

También poseen un identificador que los distingue del resto.

Grid

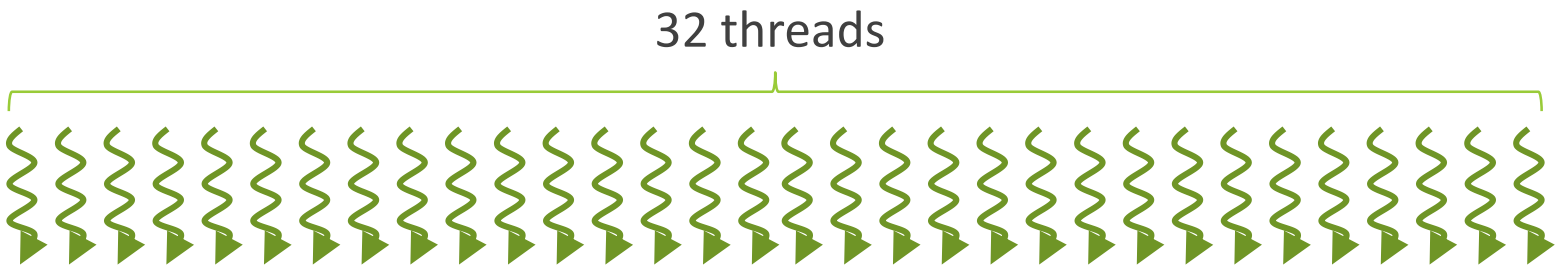
Corresponde al grupo de bloques con que fue invocado un kernel.

Permite una organización de los bloques con el fin de asignarles un identificador.

Warp

Los bloques se dividen automáticamente en warps, los cuales son grupos de a lo más 32 threads asignados a un scheduler dentro de un SM.

Para que un warp posea menos de 32 thread, un bloque debe contener un número de threads no divisible por 32. De todas formas, el comportamiento de un warp con menos de 32 threads es el mismo que uno completo.

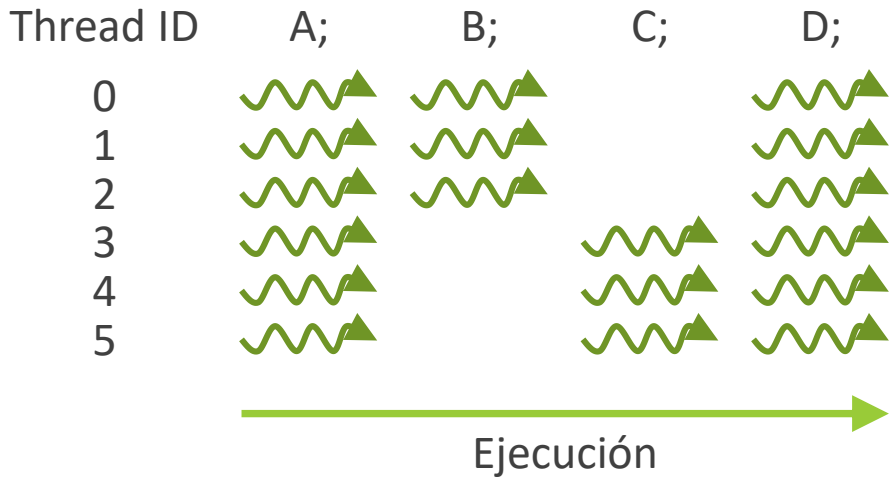


Warp

La hebras dentro de un warp pueden ejecutarse en paralelo siempre que ejecuten la misma instrucción.

Si existe divergencia, la ejecución se serializa de acuerdo a los grupos que ejecuten instrucciones distintas.

```
__global__ void kernel() {
    int i = threadIdx.x;
    A;
    if(i<3)
        B;
    else
        C;
    D;
}
```



Dimensiones – Block

El número máximo de threads en un bloque es de 1024.

Un bloque puede tener a lo más 3 dimensiones, cuyos valores máximos son:

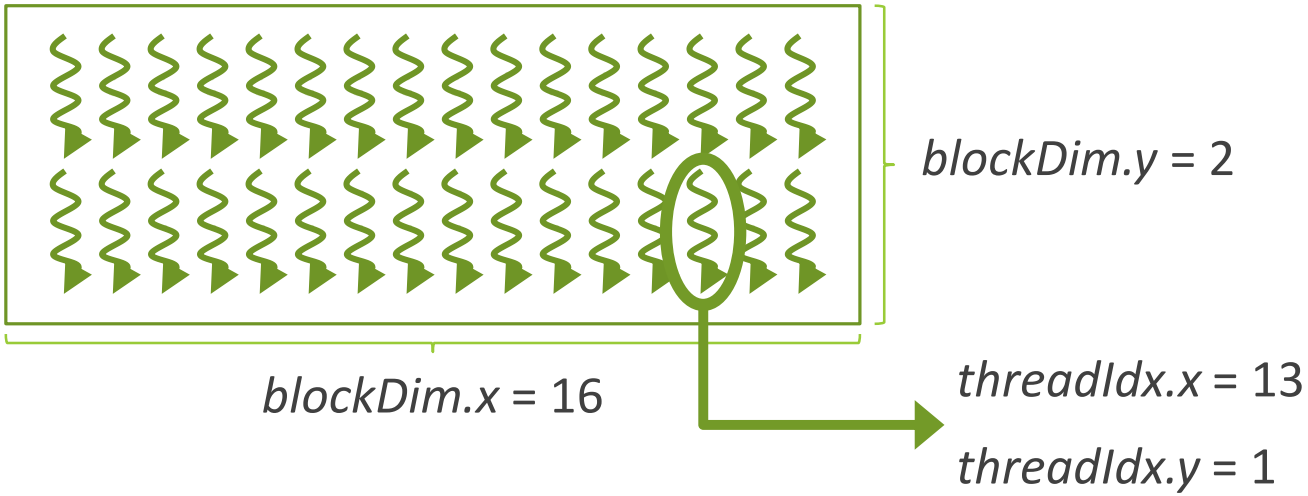
- Dimensión x: 1024
- Dimensión y: 1024
- Dimensión z: 64

Las dimensiones de un bloque pueden ser obtenidas con *blockDim.x*, *blockDim.y* y *blockDim.z*.

El identificador de un thread dentro de un bloque está determinado por *threadIdx.x*, *threadIdx.y* y *threadIdx.z*.

Dimensiones – Block

Ejemplo:



Un índice unidimensional para una hebra determinada, puede calcularse tanto como *id = threadIdx.x + blockDim.x * threadIdx.y* o como *id = threadIdx.y + blockDim.y * threadIdx.x*.

Dimensiones – Grid

El número máximo de bloques en la grid está limitado por los valores máximos de sus dimensiones.

La grid puede tener a lo más 3 dimensiones, cuyos valores máximos son:

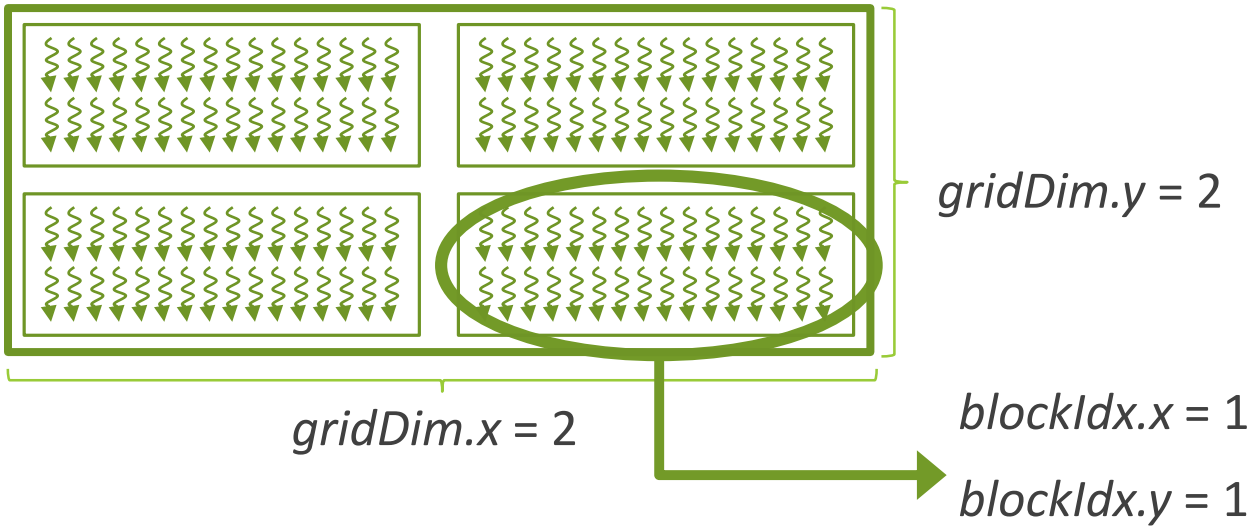
- Dimensión x: $2^{31} - 1$
- Dimensión y: 65535
- Dimensión z: 65535

Las dimensiones de la grid pueden ser obtenidas con *gridDim.x*, *gridDim.y* y *gridDim.z*.

El identificador de un bloque dentro de la grid está determinado por *blockIdx.x*, *blockIdx.y* y *blockIdx.z*.

Dimensiones – Grid

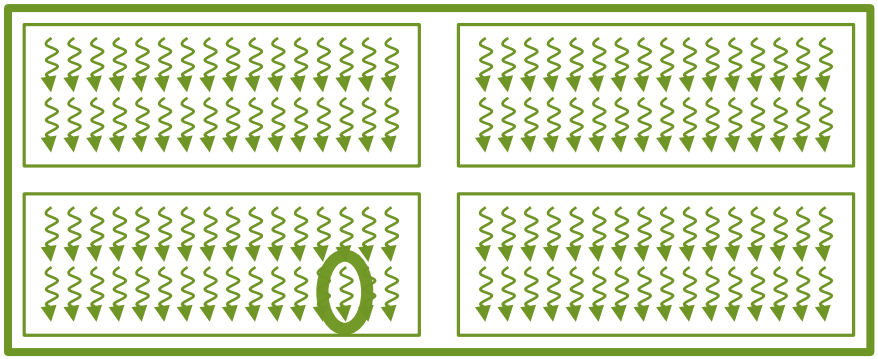
Ejemplo:



Un índice unidimensional para un bloque determinado, puede calcularse tanto como *id = blockIdx.x + gridDim.x * blockIdx.y* o como *id = blockIdx.y + gridDim.y * blockIdx.x*.

Dimensiones – Grid

Ejemplo:



Si bien *threadIdx.x* y *threadIdx.y* nos entregan los índices de una hebra dentro de un bloque, también podemos necesitar los índices de la hebra en la grid.

Para obtener el índice en la dimensión x basta con calcular *idx = threadIdx.x + blockIdx.x * blockDim.x*.

Para y tenemos *idy = threadIdx.y + blockIdx.y * blockDim.y*.

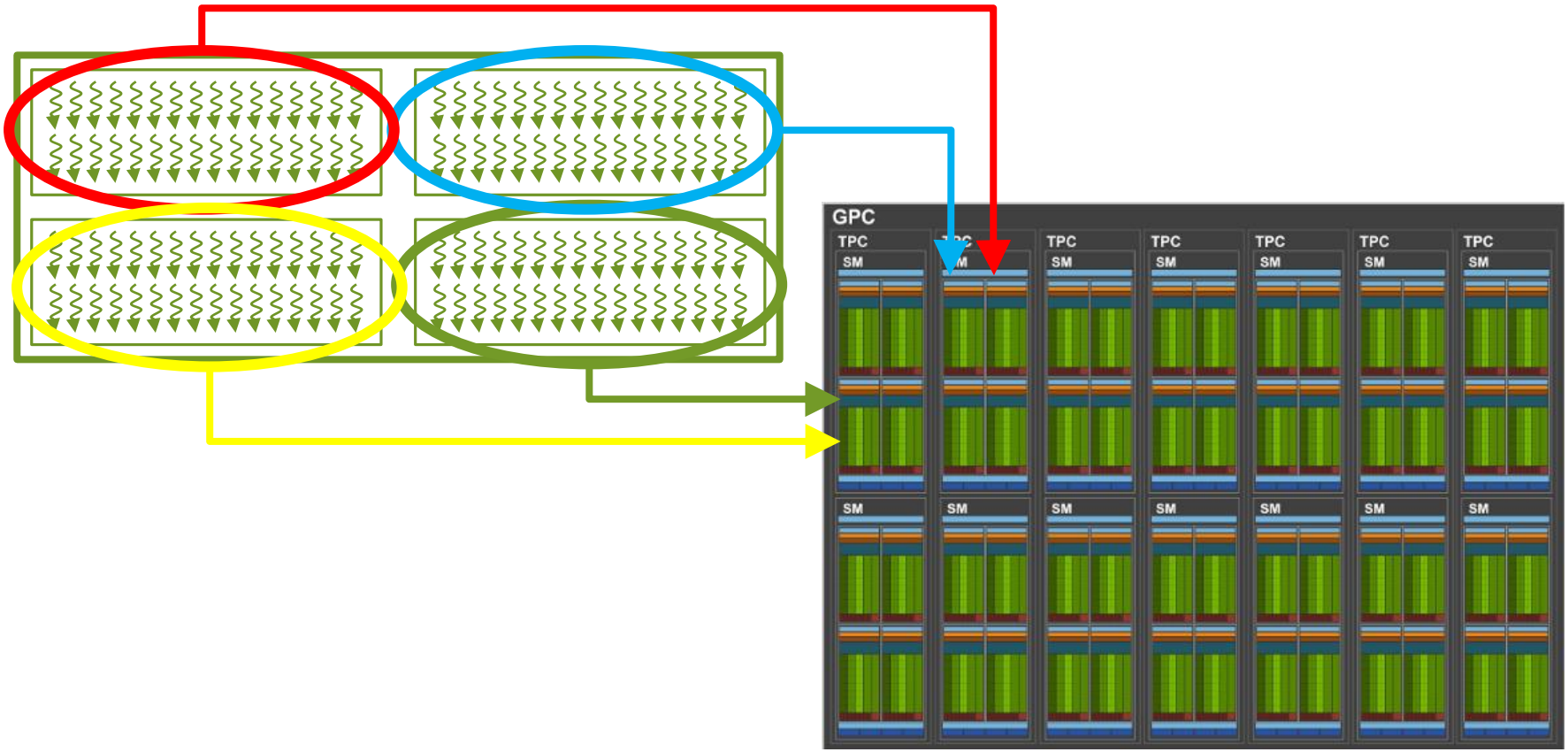
Finalmente, un índice unidimensional puede obtenerse mediante *id = idx + idy * blockDim.x * gridDim.x*.

Dimensiones – Recomendaciones

Si bien hemos aprendido a manejar bloques y grids de más de una dimension, se recomiendan las siguientes prácticas:

- Utilizar bloques y grids unidimensionales. De todas formas tenemos una cantidad máxima de threads igual a $1024 \times (2^{31} - 1) \approx 2.2 \times 10^{12}$, lo cual es más que suficiente. Por otra parte, obtener índices para arrays de más dimensiones es relativamente sencillo a partir de los identificadores unidimensionales de threads.
- Ya que solemos necesitar una cantidad de hebras mayor o igual a la cantidad de datos que queremos procesar, una de las dos dimensiones (bloques y grid) debe ser fijada mientras la otra se calcula en ejecución. Se recomienda fijar el número de hebras por bloque (múltiplo de 32) y calcular el número de bloques.
- Finalmente, los números mágicos para la cantidad de threads por bloque son 512 para Volta y Turing y 256 para microarquitecturas anteriores.

Mapping Lógico-Físico



Mapping Lógico-Físico

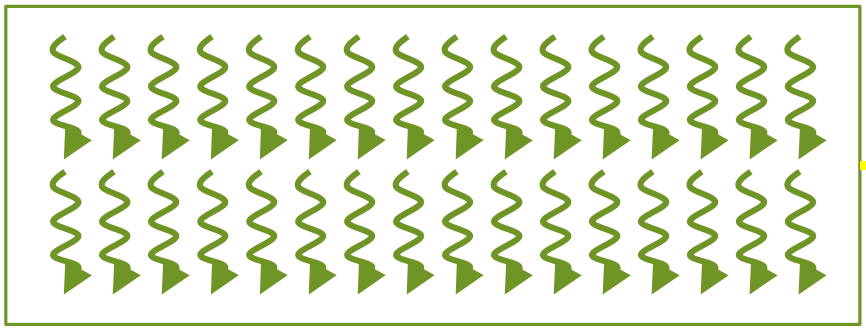
En primer lugar, los bloques se asignan a los SM.

La cantidad de bloques por SM depende de la cantidad de recursos disponibles.

Los limitantes son:

- 32 bloques por SM desde Maxwell en Adelante y 16 para microarquitecturas anteriores.
- 64 warps por SM (recordemos que un bloque puede contener a lo más 1024 threads, es decir, 32 warps).
- 65536 registros de 32 bits por SM.

Mapping Lógico-Físico



Una vez asignado un bloque a un SM, las hebras del bloque son agrupadas en warps por lo que se denomina Warp Scheduler.

Mapping Lógico-Físico

Finalmente, los warps residentes en los schedulers son preparados para que cada hebra sea asignada a un CUDA core.