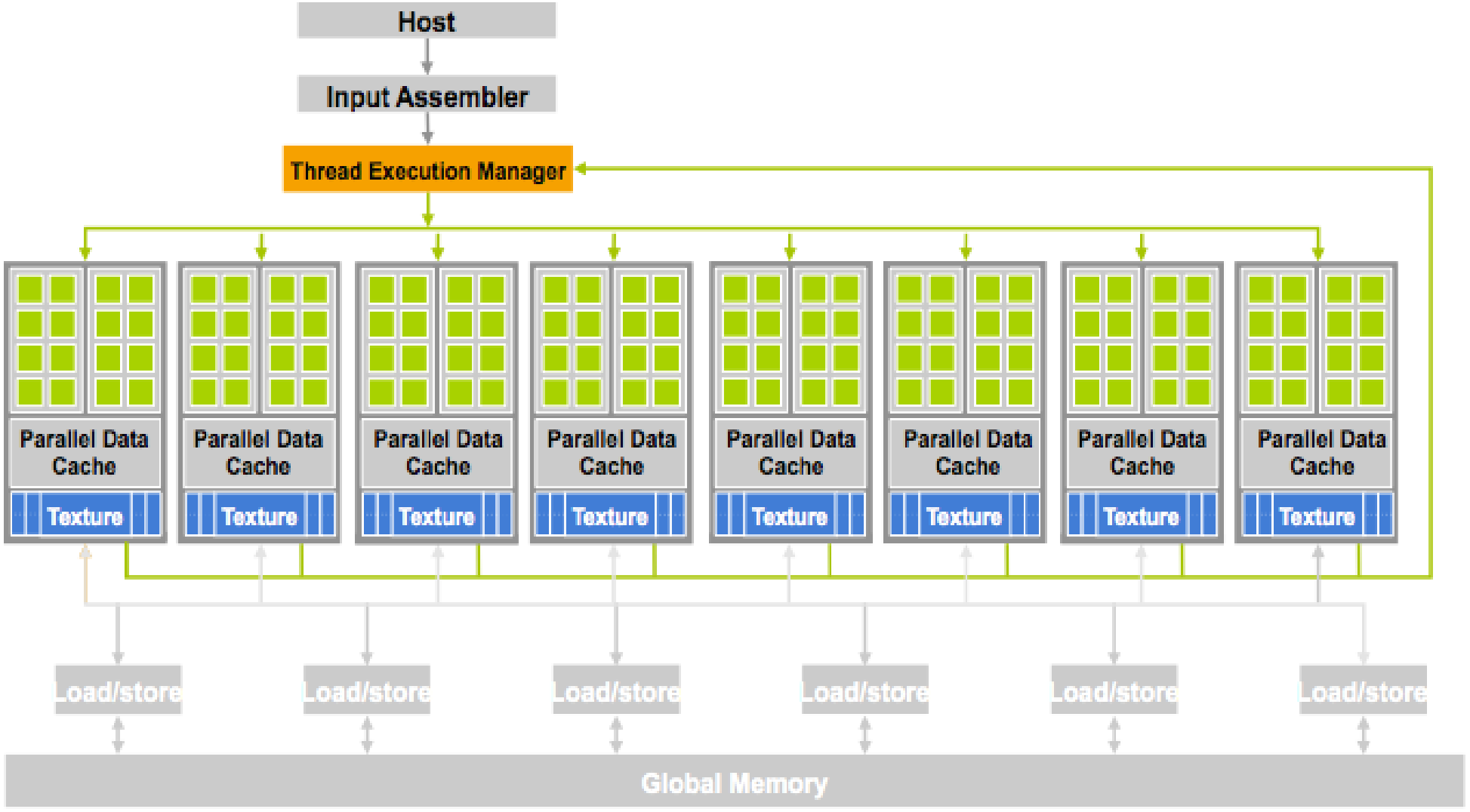
Programación Paralela

Modelos de Memoria de CUDA

José María Cecilia Canales

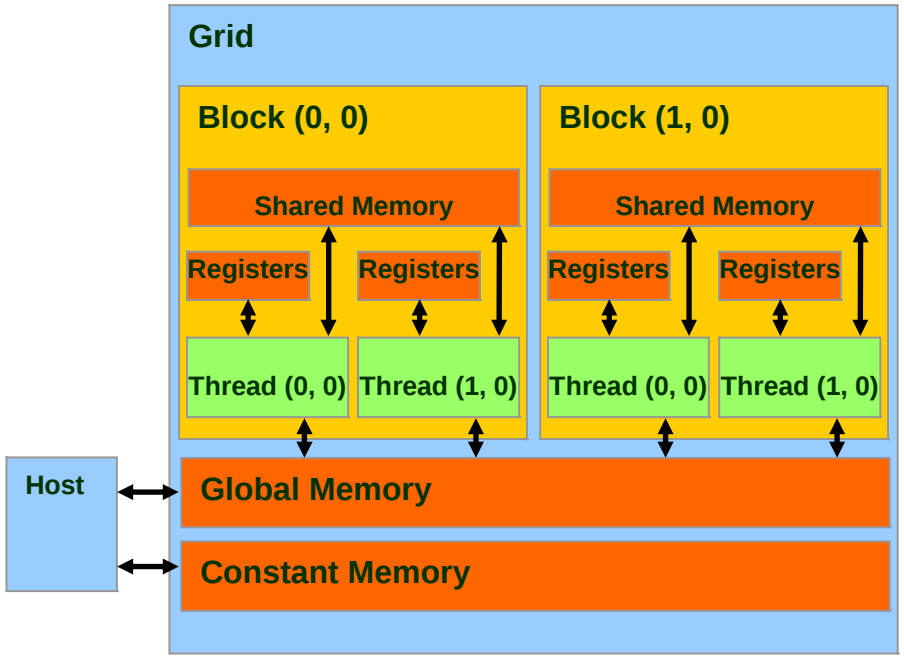
UCAM | Universidad Católica de Murcia


Host
Input Assembler
Thread Execution Manager
Parallel Data Cache
Texture
Load/store
Global Memory

Visión del programador de las memorias CUDA

Cada *thread* puede:

- Leer/escribir en registros (latencia ~1 ciclo). Visibilidad privada para cada hilo.
- Leer/escribir en *local memory.* Visibilidad privada para cada hilo.
- Leer/escribir en *shared memory* (latencia ~5 ciclos). Visibilidad privada para cada bloque.
- Leer/escribir en *global memory* (latencia ~500 ciclos). Visibilidad a todos los hilos del grid.
- Solo lectura *constant memory* (latencias ~5 ciclos si aciertos de caché). Visibilidad a todos los hilos del grid



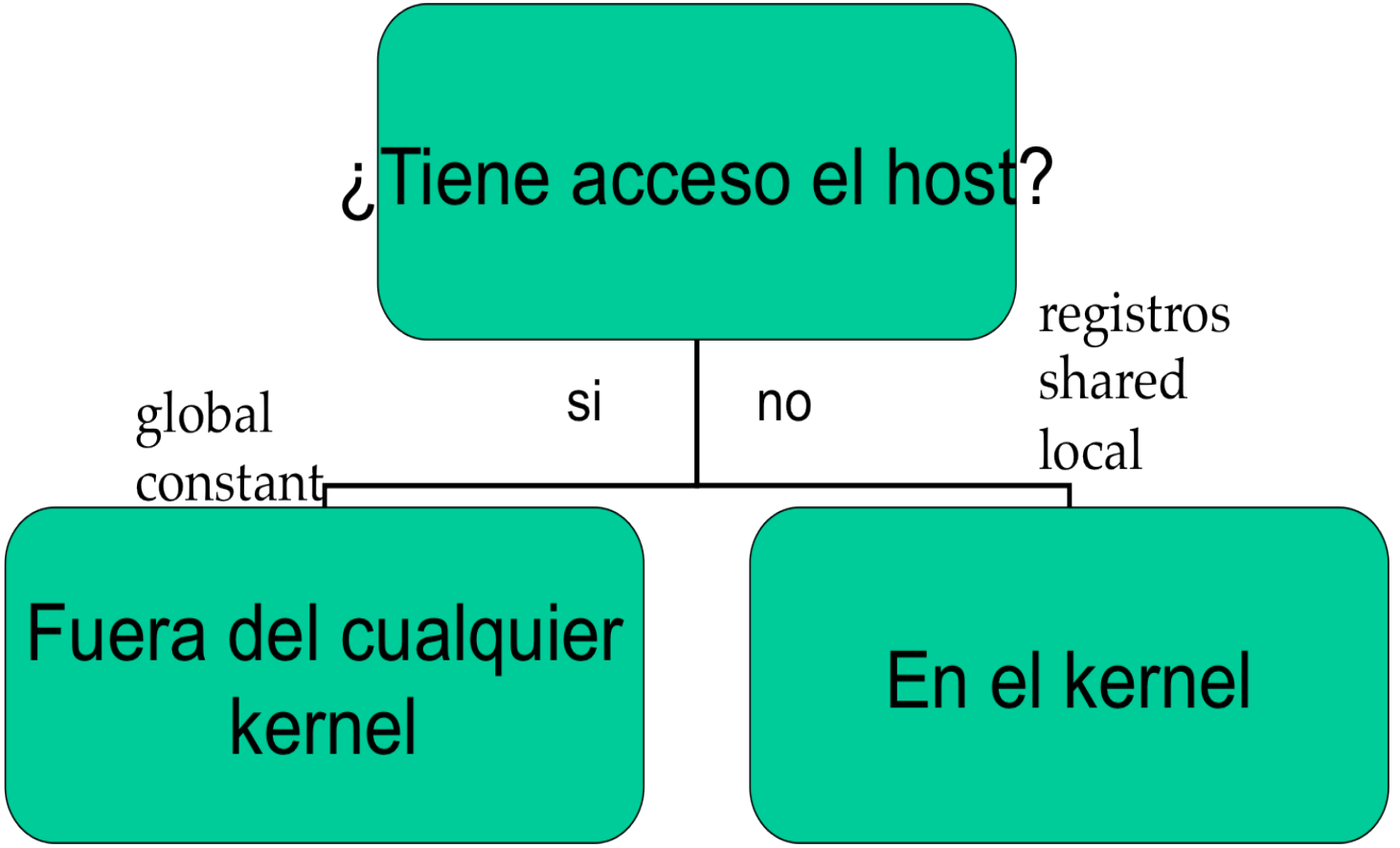
Tipos de variables en CUDA

Variable declaration	Memoria	Alcance	Vida
`int LocalVar;`	**Local**	**Thread**	**Thread**
`__device__ __shared__ int SharedVar;`	**Shared**	**Bloque**	**Bloque**
`__device__ int GlobalVar;`	**Global**	**Grid**	**Aplicación**
`__device__ __constant__ int ConstantVar;`	**Constant**	**Grid**	**Aplicación**

`__device__` es opcional cuando se usa con `__shared__`, o `__constant__`

Variables por defecto sin ningún prefijo se asignan a registros, salvo los arrays que residen en local memory

¿Dónde se declaran las variables?



Una estrategia común de programación

- La *global memory* es mucho mas lenta que la *shared memory*.
- Estrategia ***Tiling*** para aprovechar la *shared memory*:
  - Particionamos los datos en subconjuntos que quepan en *shared memory* (48 KB). Esto es lo que llamamos ***Tile***
  - Manejar cada subconjunto de datos con un bloque de hilos:
    - Cargamos un subconjunto de datos de *global memory* a *shared memory*, usando varios hilos para aprovechar el paralelismo de datos.
    - Realizamos las operaciones sobre cada subconjunto en *shared memory*.
    - Copiamos los resultado desde *shared memory* a *global memory.*

Una estrategia común de programación

- Memoria de constantes (*constant memory*) reside en *device memory* -> Acceso mucho mas lento que *shared memory*
  - Pero existe una cache de 64 Kbytes
  - Eficiente para sólo lectura
- Dividir los datos de acuerdo con los patrones de acceso y su naturaleza.
  - Si sólo lectura -> *constant memory* (muy rápido si está en cache)
  - Si los hilos de un mismo bloque comparten datos -> *shared memory* (muy rápido)
  - Si es información privada de cada hilo -> registros (lo más rápido)
  - Información de Entrada/Salida -> *global memory* (muy lenta)

También existe la memoria de texturas, pero no la vamos a tratar

Operaciones atómicas en la GPU

- Operaciones atómicas son operaciones que se ejecutan sin la interferencia de ningún otro hilo.
- Se utilizan para prevenir condiciones de carrera.
- Algunas operaciones atómicas (Ver programming guide):

  add, sub, min, max, ...

  and, or, xor

  Incremento, decremento

  Exchange, compare and swap

Requiere que el HW sea compute capability 1.1

Mas tópicos en el modelo de memoria

- Memoria caché para los accesos a memoria global (Arquitecturas Fermi y posterior)
- Constant memory
- Texture memory