
## SHARED MEM, SYNC, ATTOMIC FUNCTIONS

# CUDA PROGRAMMING

# TEMAS DE HOY

A. TIPOS DE MEMORIA EN EL GPU

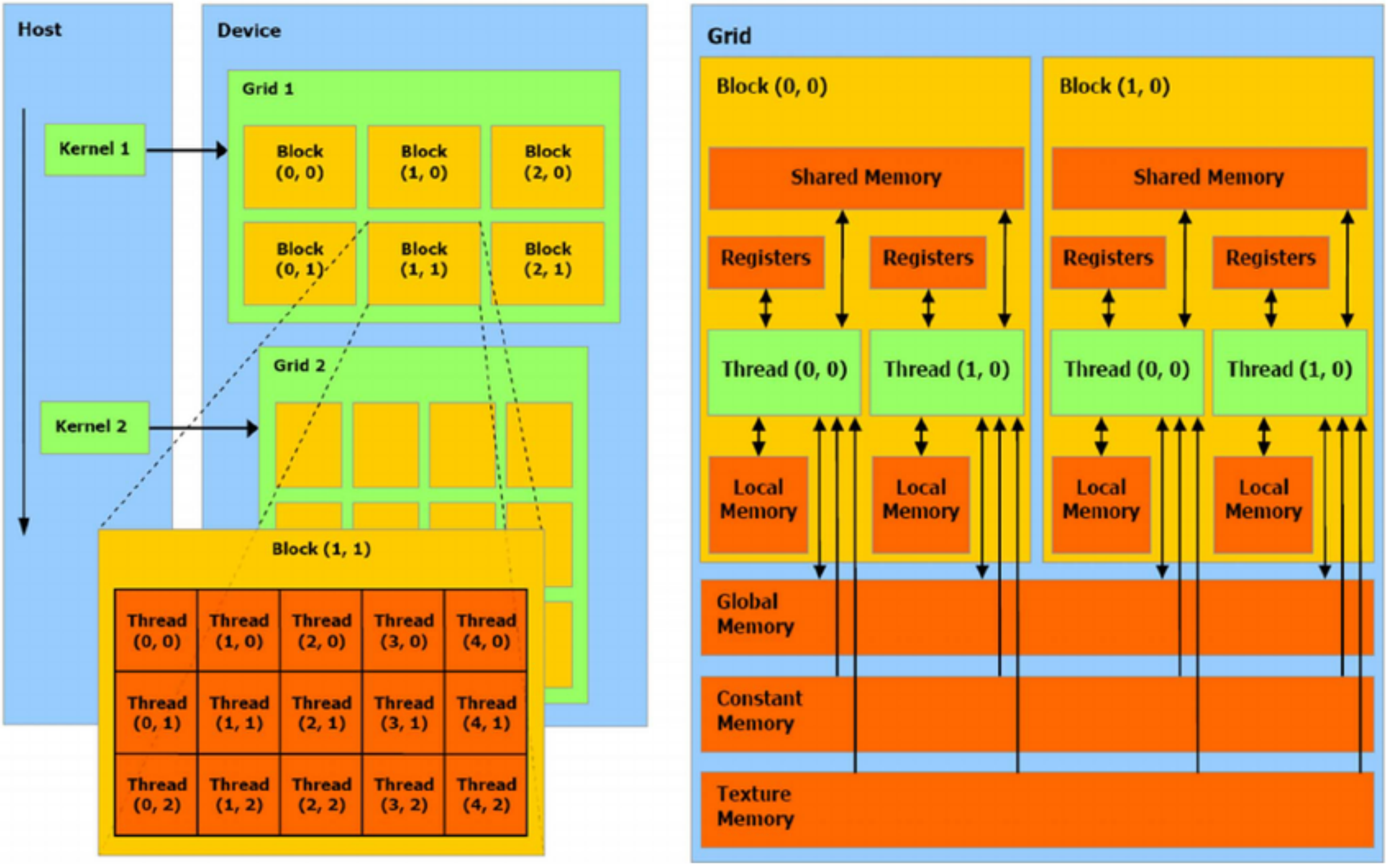
B. MEMORIA COMPARTIDA

C. CUDA SYNC

D. OPERACIONES ATOMICAS

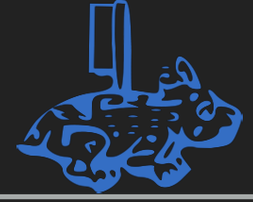
# TIPOS DE MEMORIA EN EL GPU

- **Memoria Global (__device__)**: Cualquier hilo en cualquier bloque tiene acceso a ella. Tiene la desventaja de ser lenta.
- **Memoria Compartida (__shared__)**: Se comparte entre los hilos de un bloque.
- **Memoria Constante (__constant__)**: Truculenta, puede ser rápida o lenta dependiendo de cómo se use.
- **Memoria de textura**: Optimizada para acceso en patrones 2D


Host
Device
Grid 1
Kernel 1
Block (0, 0)
Block (1, 0)
Block (2, 0)
Block (0, 1)
Block (1, 1)
Block (2, 1)
Grid 2
Kernel 2
Block (1, 1)
Thread (0, 0)
Thread (1, 0)
Thread (2, 0)
Thread (3, 0)
Thread (4, 0)
Thread (0, 1)
Thread (1, 1)
Thread (2, 1)
Thread (3, 1)
Thread (4, 1)
Thread (0, 2)
Thread (1, 2)
Thread (2, 2)
Thread (3, 2)
Thread (4, 2)
Grid
Block (0, 0)
Block (1, 0)
Shared Memory
Registers
Thread (0, 0)
Thread (1, 0)
Local Memory
Global Memory
Constant Memory
Texture Memory

# MEMORIA COMPARTIDA (SHARED MEMORY)

- La memoria compartida se comparte entre los hilos de un bloque
- Para compartir memoria entre bloques se utiliza la memoria global.
- La memoria compartida suele ser no muy grande (~64KB)



# MEMORIA COMPARTIDA ESTATICA VS DINAMICA

La declaración nos indica que el arreglo tiene un tamaño fijo, por lo tanto no puede aumentar la cantidad de memoria **reservada.**

```
__shared__ int  ehecatl[64];
```

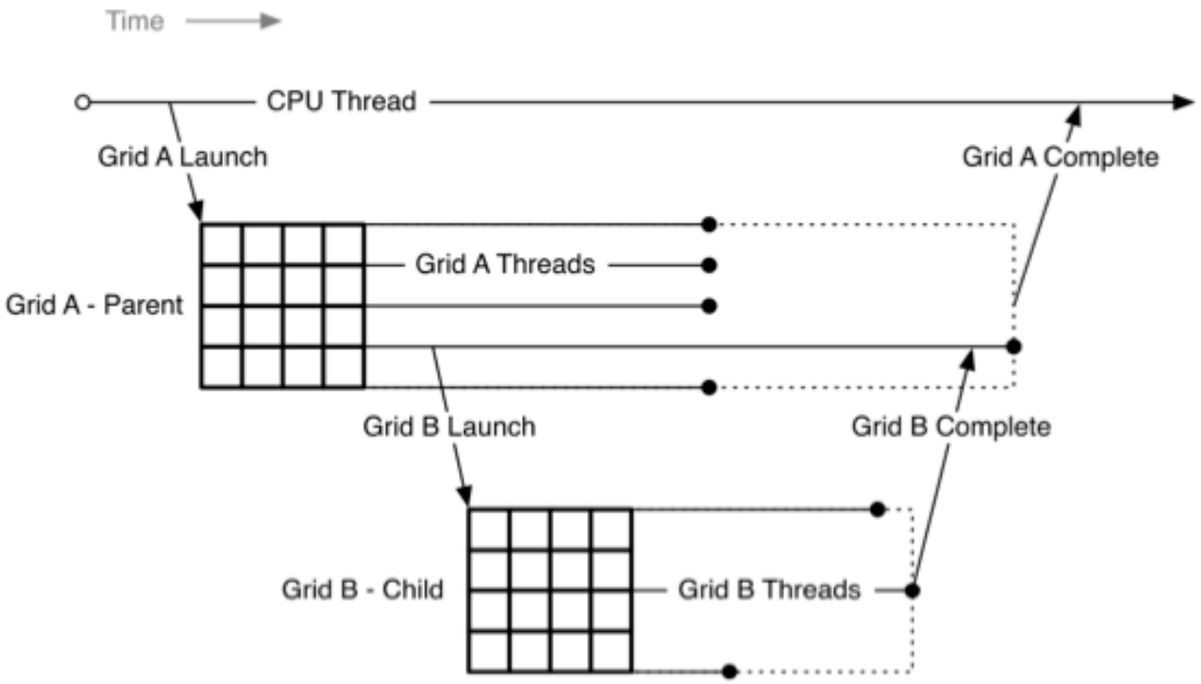
La palabra "extern" nos indica que el arreglo es dinámico, por lo tanto no es fija la cantidad de elementos que tiene el arreglo**.**

```
extern __shared__ int kopitl[];
```

extern->"**No reservar memoria**"

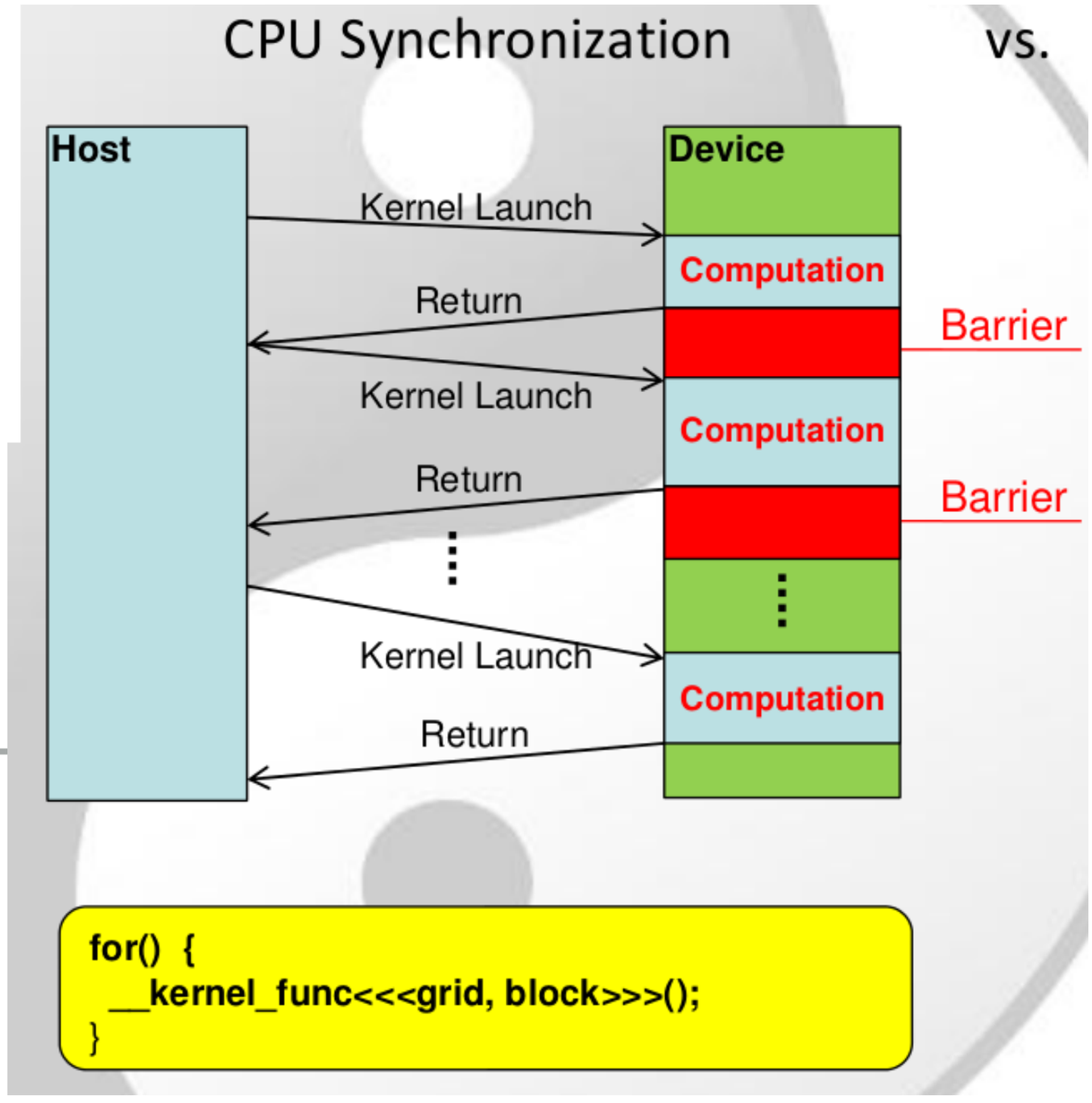
Y cuando se llama el KERNEL, se debe agregar un 3er parametro dentro de '<<< >>>', el cual indica el tamaño de la memoria compartida que se desea dar al kernel... Ejemplo:

```
miKernel<<<1,n,n*sizeof(int)>>>(dev_A, n);
```




# PARA ESPERAR QUE LOS HILOS TERMINEN SU TAREAS:

## __SYNCTHREADS();



# OPERACIONES ATOMICAS


Evita que 2 o más hilos quieran entrar y/o modificar un valor al mismo tiempo y en la misma dirección. Asigna turnos para que no haya perdida de datos.



# FUNCIONES ATOMICAS DISPONIBLES

```
atomic{Add, Sub, Exch, Min, Max, Inc, Dec, CAS,
       And, Or, Xor}
```

Syntax: `atomicAdd(float *address, float val)`

### B.12.1.2. atomicSub()

```
int atomicSub(int* address, int val);
unsigned int atomicSub(unsigned int* address,
                       unsigned int val);
```

reads the 32-bit word `old` located at the address `address` in global or shared memory, computes `(old - val)`, and stores the result back to memory at the same address. These three operations are performed in one atomic transaction. The function returns `old`.

# MAS INFORMACION EN: HTTP://DOCS.NVIDIA.COM/CUDA/CUDA-C-PROGRAMMING-GUIDE/INDEX.HTML#ATOMIC-FUNCTIONS