

# *Párhuzamos és Grid rendszerek*

## *(10. ea)*

*GPGPU*

Szeberényi Imre

BME IIT

<szebi@iit.bme.hu>

Az ábrák egy része az NVIDIA oktató anyagaiból és dokumentációiból származik.



MŰEGYETEM 1782

# Általános célú GPU

- A programozható vertex és fragment shaderek beépítésével általános célú eszközzé vált.
- Vektorprocesszor (SIMD), de pipeline egységek is vannak benne (MISD).
- Jellemzően SIMD
- Programozás: CUDA, OpenCL, Cg, ...

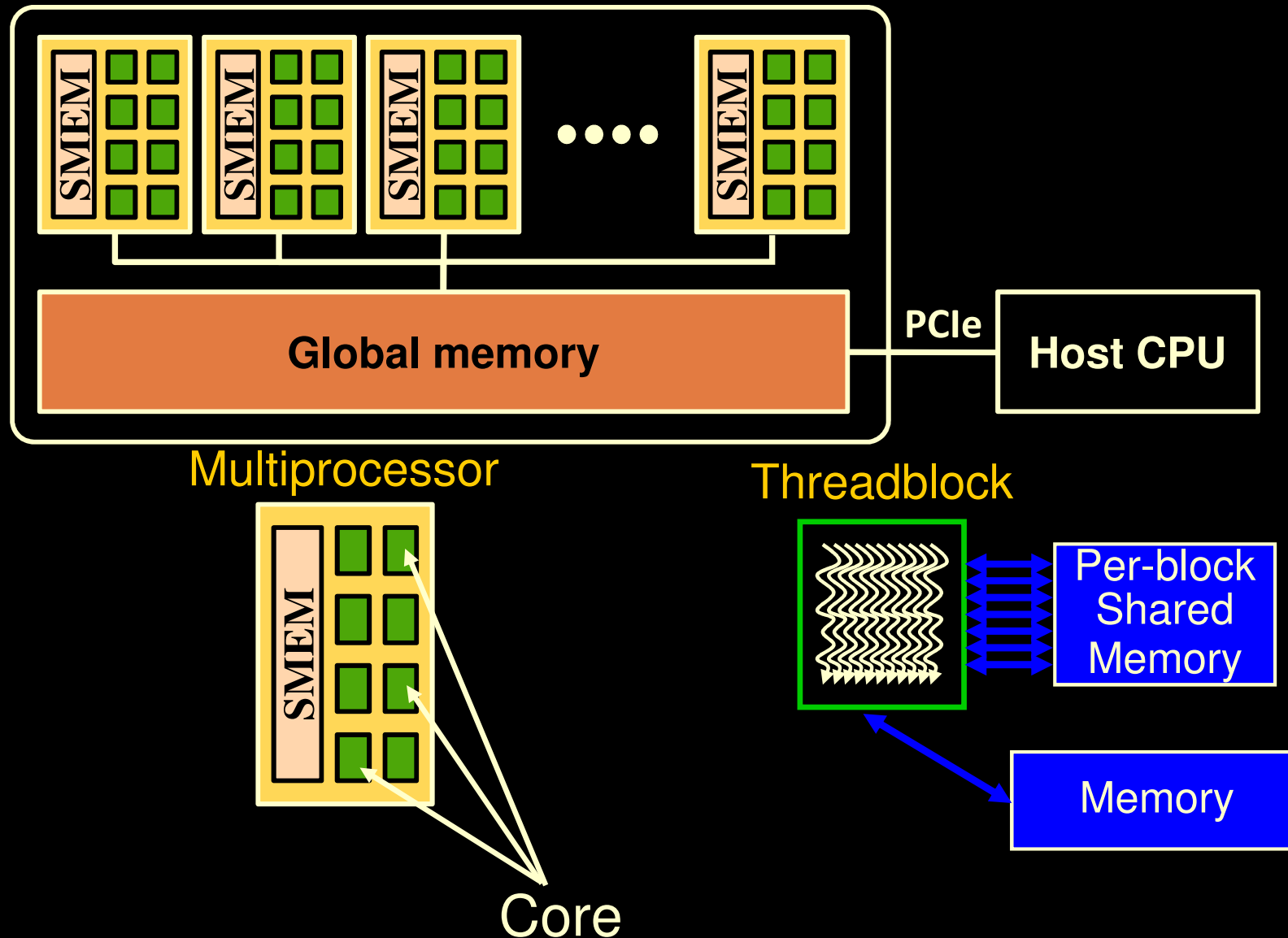
# *Egy példa*

## NVIDIA Quadro FX5800 grafikus kártya

- PCIe x16
- 240 CUDA mag
- 4 GB DDR3
- 78 GFlops double
- 933 Gflops single
- 189 W
- 300Millió háromszög / sec
- NVIDIA CUDA



# Compute Unified Device Arch.



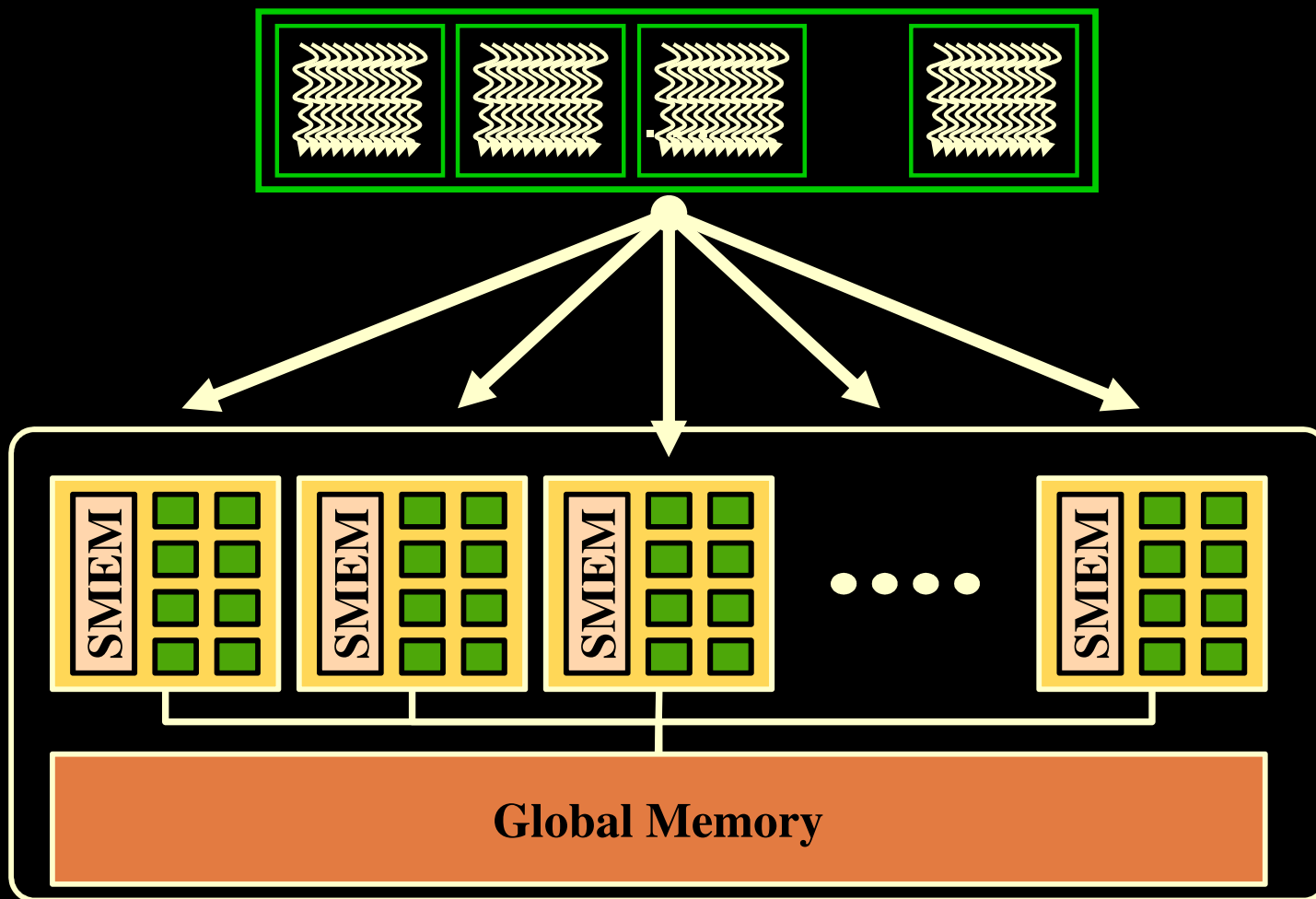
# *CUDA modell*

---

- Ún. thread modell (SIMT)
- A szálak ütemezésével nem kell a programozónak foglalkozni.
- Elrejt a konkrét architektúrát
- Támogatja a heterogén feldolgozás (CPU+GPU)

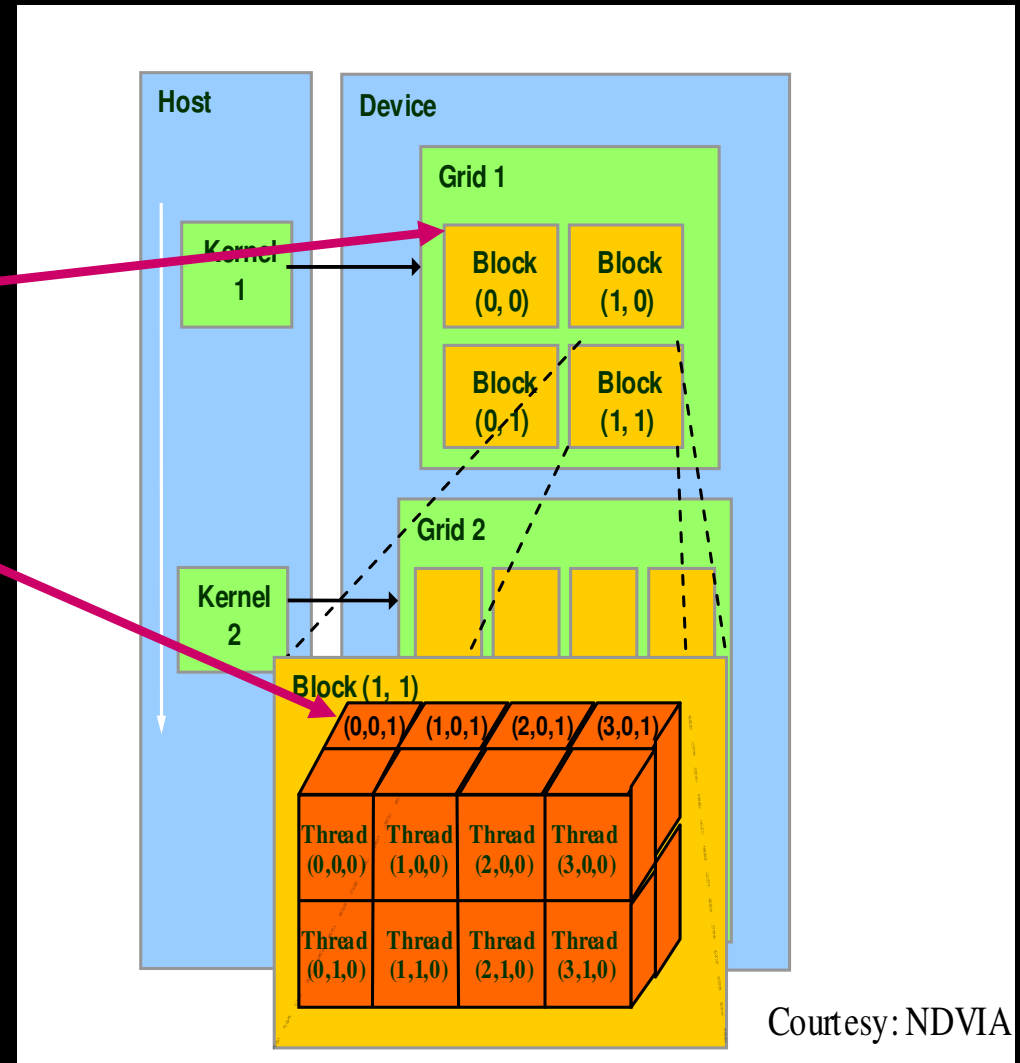
# Grid-be szervezett szálak

Many blocks of threads



# Block ID & Thread ID

- Minden szálnak és blokknak van egyedi azonosítója.
  - Block ID: 1D v 2D
  - Thread ID: 1D, 2D, v. 3D
  - gridDim, blockIdx
  - blockDim, threadIdx

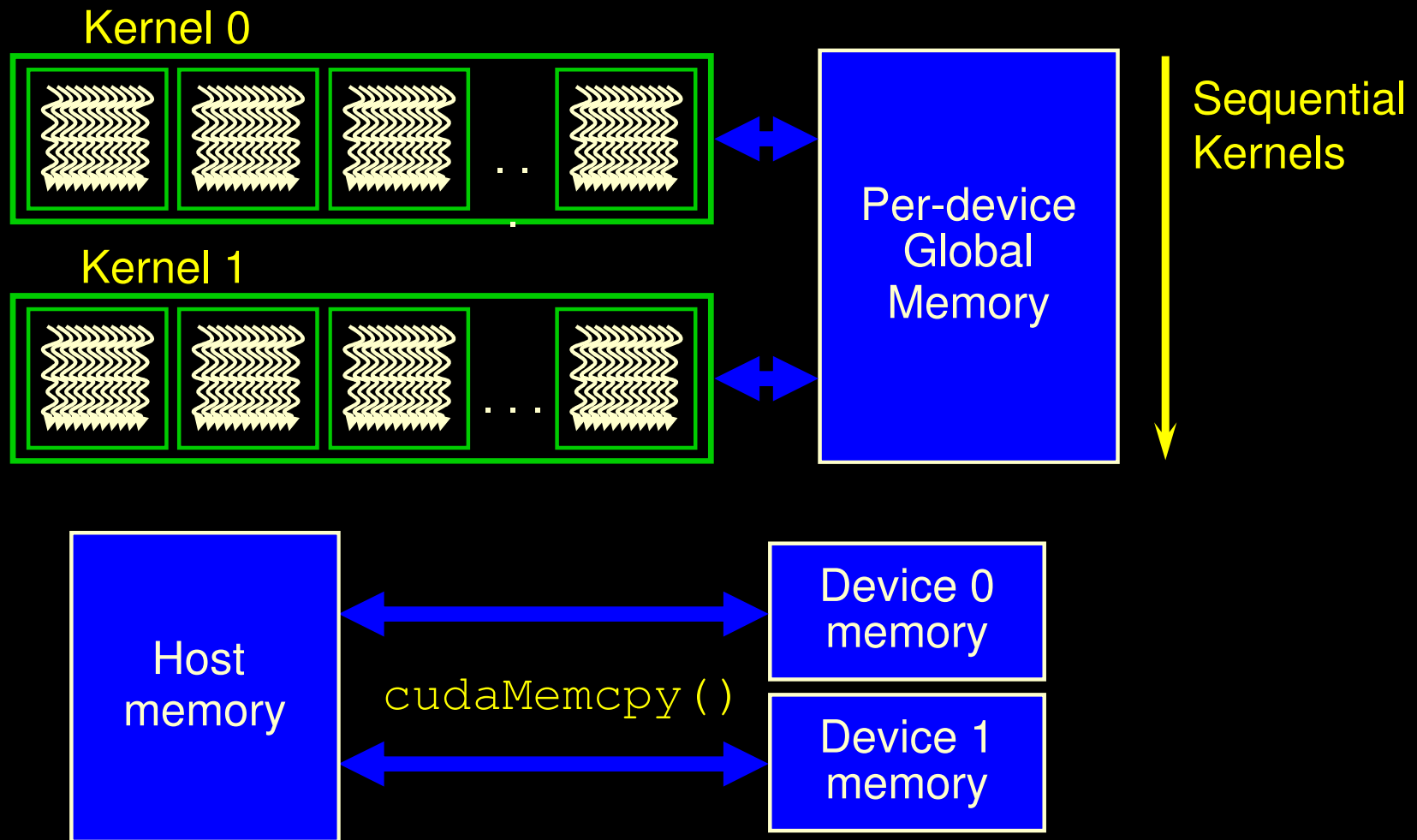


# *Szálak kommunikációja*

- Szálak párhuzamosan futnak.
- Sorrendjük nem meghatározható
- Csak egy blokkban levő szálak tudnak kommunikálni egymással.
- A blokkok futási sorrendje nem meghatározott.
- warp általában 32 szál



# Host kommunikáció



# C nyelv kiegészítései

Függvény típus módosítói:

```
__global__ void my_kernel() { }  
__device__ float my_device_func() { }
```

Változók típusmódosítói::

```
__constant__ float  
my_constant_array[32];  
__shared__ float  
my_shared_array[32];
```

Beépített változók:

```
dim3 gridDim; // Grid dimension  
dim3 blockDim; // Block dimension  
dim3 blockIdx; // Block index  
dim3 threadIdx; // Thread index
```

# *C/C++ megkötések*

- Kernel kód csak a GPU memóriáját éri el
- Nem lehet rekurzió
- Nincs változó argumentumszám
- Nincs static
- Nincs dinamikus polimorfizmus

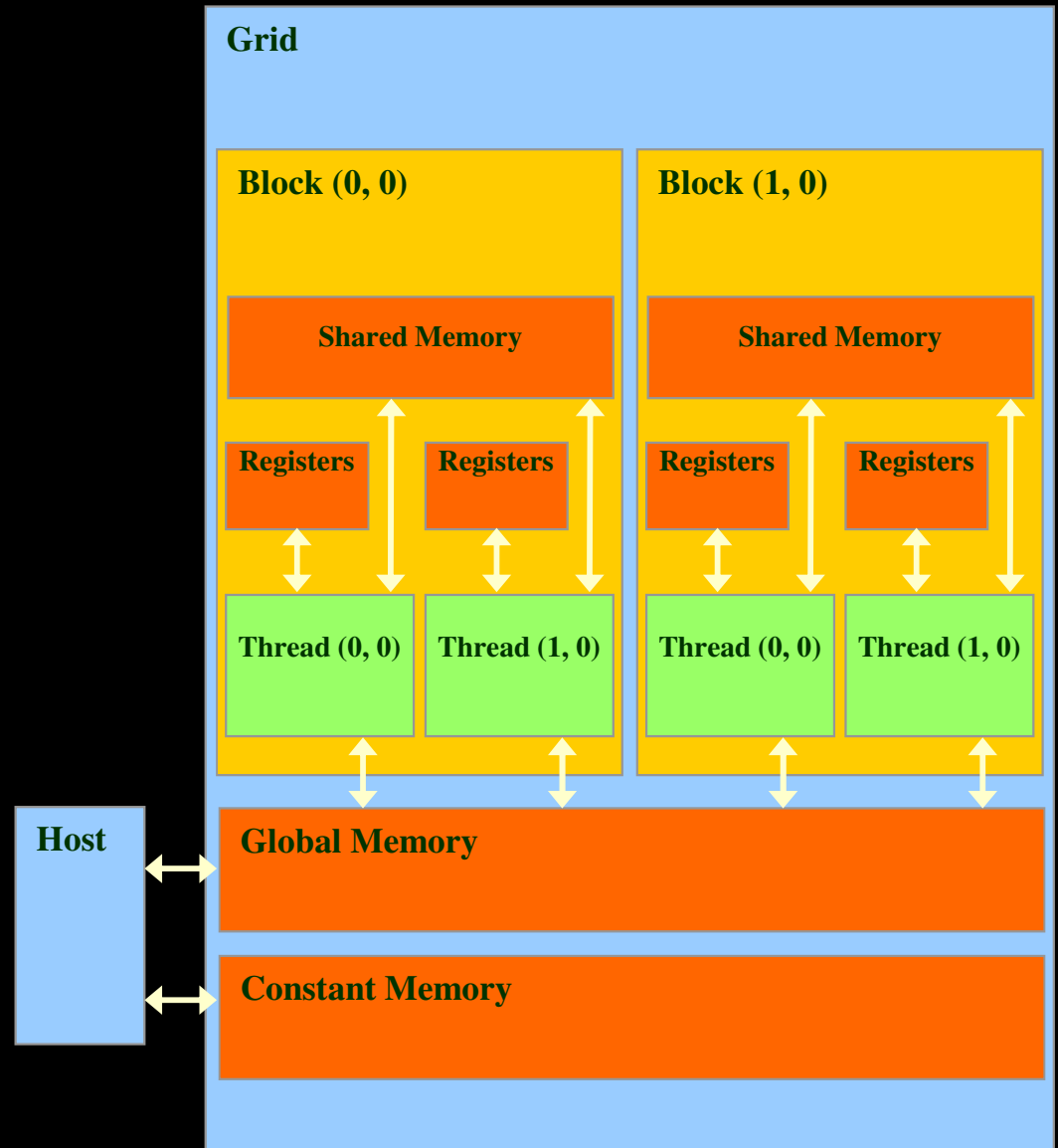
# Függvény deklaráció

	Hol hajtódik végre?	Honnan hívható?
<b>__device__</b> float DeviceFunc ()	device	device
<b>__global__</b> void KernelFunc ()	device	host
<b>__host__</b> float HostFunc ()	host	host

# Memória elérés

Minden thread tudja:

- Írni/olvasni a thread regisztereket
- Írni/olvasni a lokális memóriát
- Írni/olvasni a shared memóriát
- Írni/olvasni a globális memóriát
- Olvasni a konstans memóriát



# Változó helye

Variable declaration	Memory	Scope	Lifetime
<code>int var;</code>	register	thread	thread
<code>int array_var[10];</code>	local	thread	thread
<code>__shared__ int shared_var;</code>	shared	block	block
<code>__device__ int global_var;</code>	global	grid	application
<code>__constant__ int constant_var;</code>	constant	grid	application

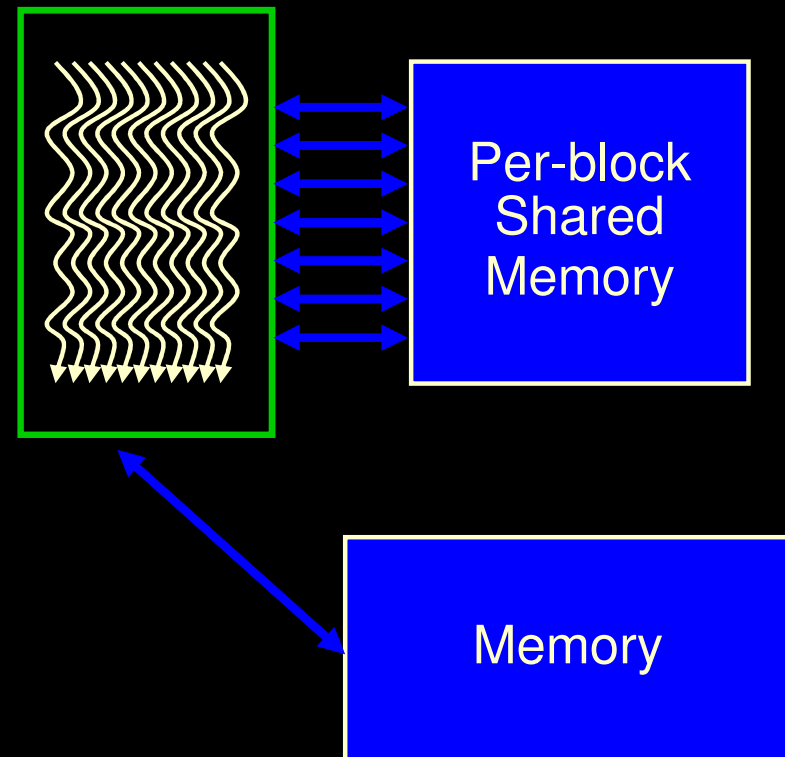
# *Elérés sebessége*

Variable declaration	Memory	Penalty
<code>int var;</code>	register	<b>1x</b>
<code>int array_var[10];</code>	local	<b>100x</b>
<code>__shared__ int shared_var;</code>	shared	<b>1x</b>
<code>__device__ int global_var;</code>	global	<b>100x</b>
<code>__constant__ int constant_var;</code>	constant	<b>1x</b>

# Példa

- Vektor négyzetgyöke
- Bemenet/kimenet:
  - input, output vektor
- Host program
- Device program

Threadblock





# Példa (sqrt #1)

```
// Device code
__global__ void sqrtVec(const double* in,
                        double* out, int N) {
    int i = blockDim.x * blockIdx.x + threadIdx.x;
    if (i < N)
        out[i] = sqrt(in[i]);
}
```

```
int main() {
    ....
    sqrtVec<<<20, 256>>>(d_inp, d_res, SIZE);
    ....
}
```

# Példa (sqrt #2)

```
int main() {
    double *h_inp, *h_res, *d_inp, *d_res;
    size_t size = 100 * sizeof(double);
    h_inp = (double*)malloc(size);
    h_res = (double*)malloc(size);
    cudaMalloc((void**)&d_inp, size);
    cudaMalloc((void**)&d_res, size);
    for (int i = 0; i < SIZE; i++) h_inp[i] = i*i;
    cudaMemcpy(d_inp, h_inp, size,
               cudaMemcpyHostToDevice);
    sqrtVec<<<1, 100>>>(d_inp, d_res, 100);
    cudaMemcpy(h_res, d_res, size,
               cudaMemcpyDeviceToHost);
    .... cudaTreadExit();
    return 0;
}
```

# Versenyhelyzet

```
threadId:0
```

```
vector[0] = 5;
```

```
...
```

```
a = vector[0];
```

```
threadId:1917
```

```
vector[0] = 1;
```

```
...
```

```
a = vector[0];
```

Mi az a változó értéke a 0. ill. 191. számban ?

# *Atomi művelet*

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

CAS:

```
int compare_and_swap(int* register,  
                    int oldval, int newval){  
    int old_reg_val = *register;  
    if(old_reg_val == oldval)  
        *register = newval;  
    return old_reg_val;  
}
```

# *Szinkronizációs veszély*

```
__global__  
void global_max(int* values, int*  
gl_max) {  
    int i = threadIdx.x  
        + blockDim.x * blockIdx.x;  
    int val = values[i];  
    atomicMax(gl_max, val);  
}
```

**A gyakori közös memória hozzáférés miatt a szálak elakadnak, szinkronozódnak.**

# *Kicsit jobb max. keresés*

```
__global__
void global_max(int* values, int* max,
               int *regional_maxes, int num_regions)
{
    int i = threadIdx.x
          + blockDim.x * blockIdx.x;
    int val = values[i];
    int region = i % num_regions;
    if(atomicMax(&reg_max[region], val) <
        val) {
        atomicMax(max, val);
    }
}
```

# Vezérlés divergencia

A ClearSpeed-nél már megismert probléma, ami rontja a párhuzamosítás hatékonyságát.

