CUDA/GPU Programming Model

Bin ZHOU @ NVIDIA & USTC

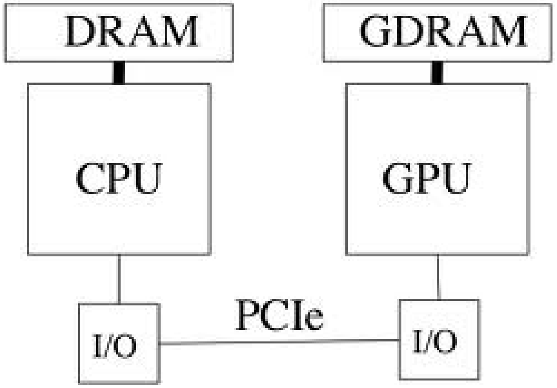
Jan. 2015

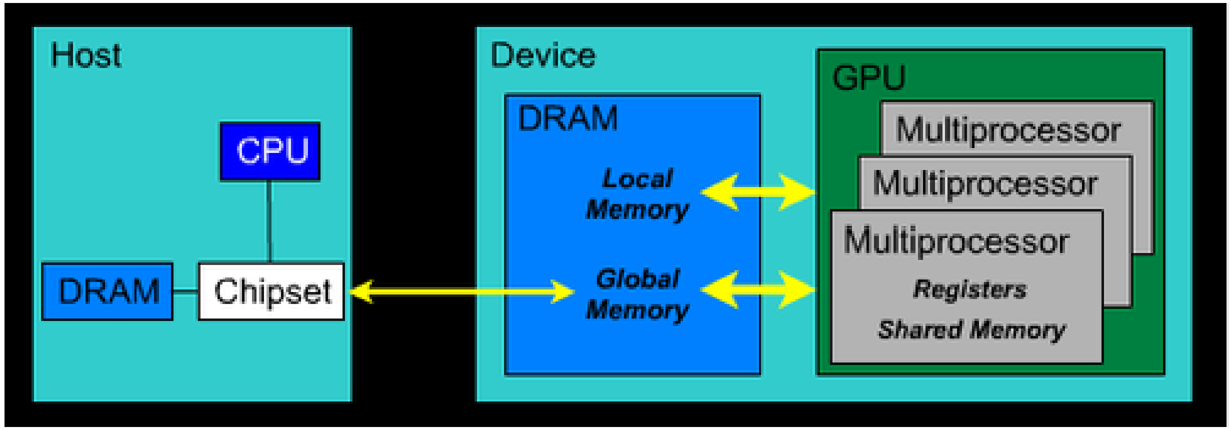
Contents

- CPU&GPU Interaction
- GPU Thread Organization （important）
- GPU Memory Hierarchy
- Some Basic Programming

CPU-GPU Interaction

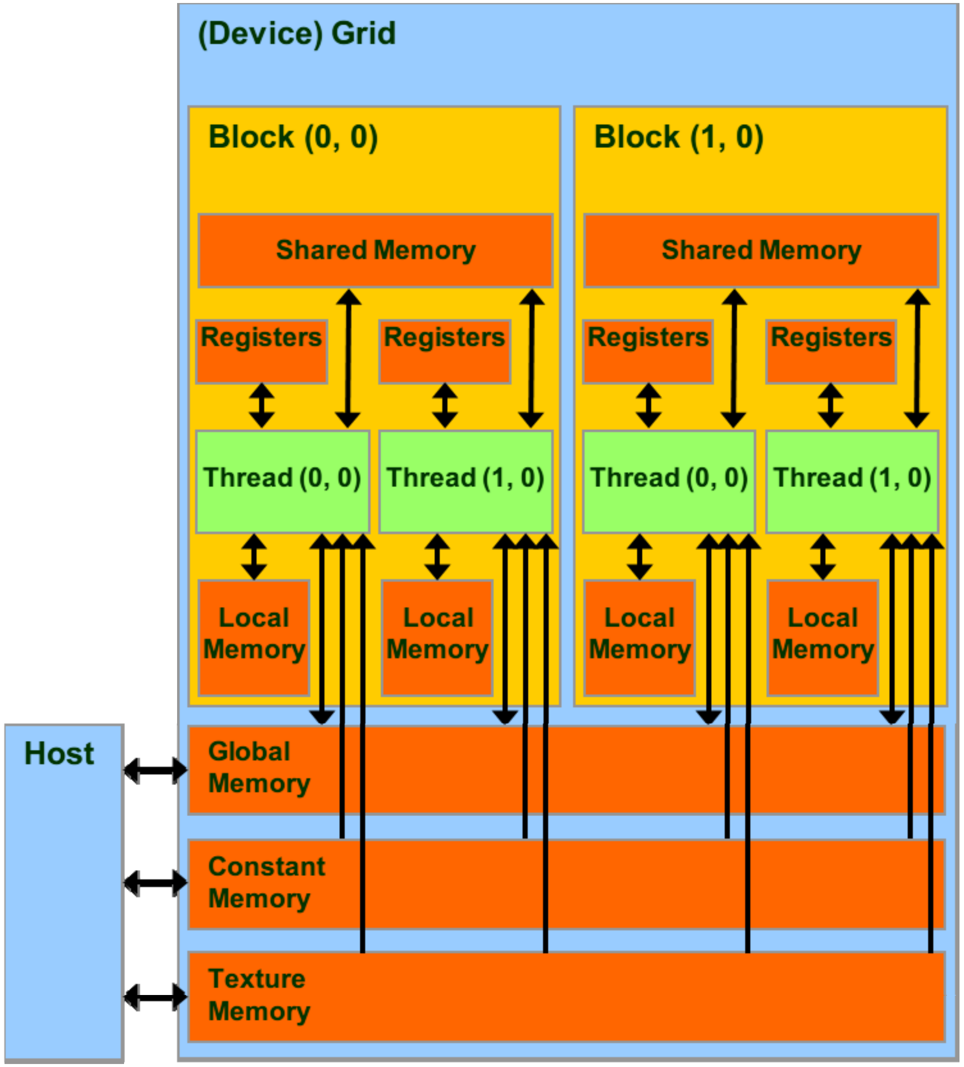
- Separate Physical Memory Space
- Via PCIE Bus (8GB/s～16GB/s)
- Communication Overhead



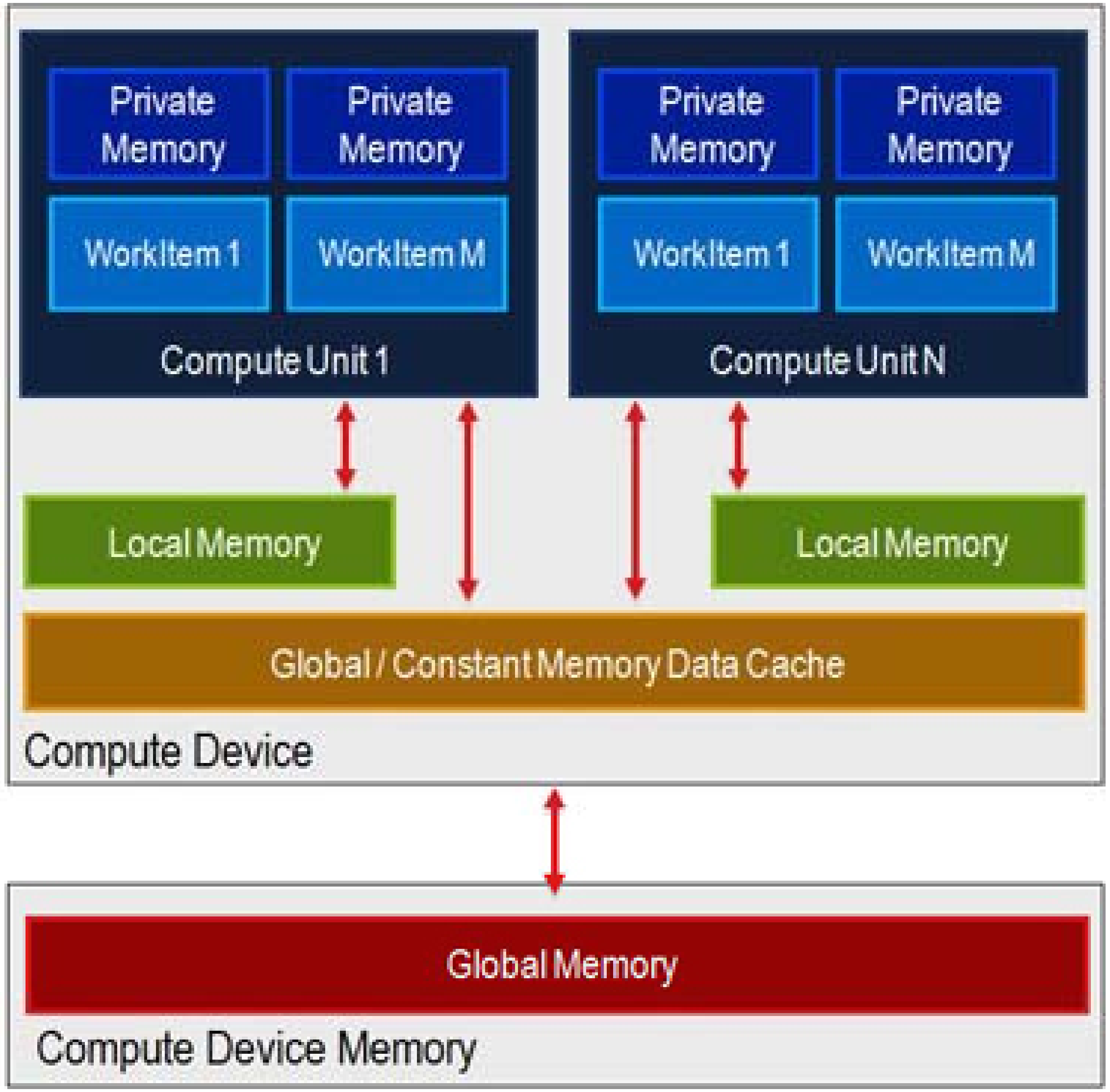


© NVIDIA Corporation

GPU Memory Hierarchy (CUDA View)



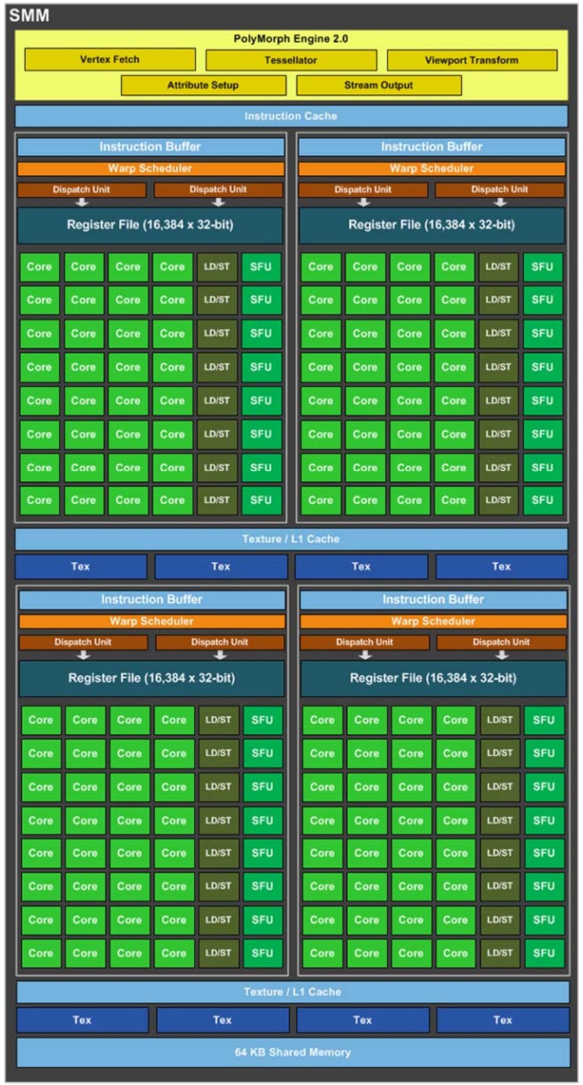
GPU Memory Hierarchy (OpenCL View)

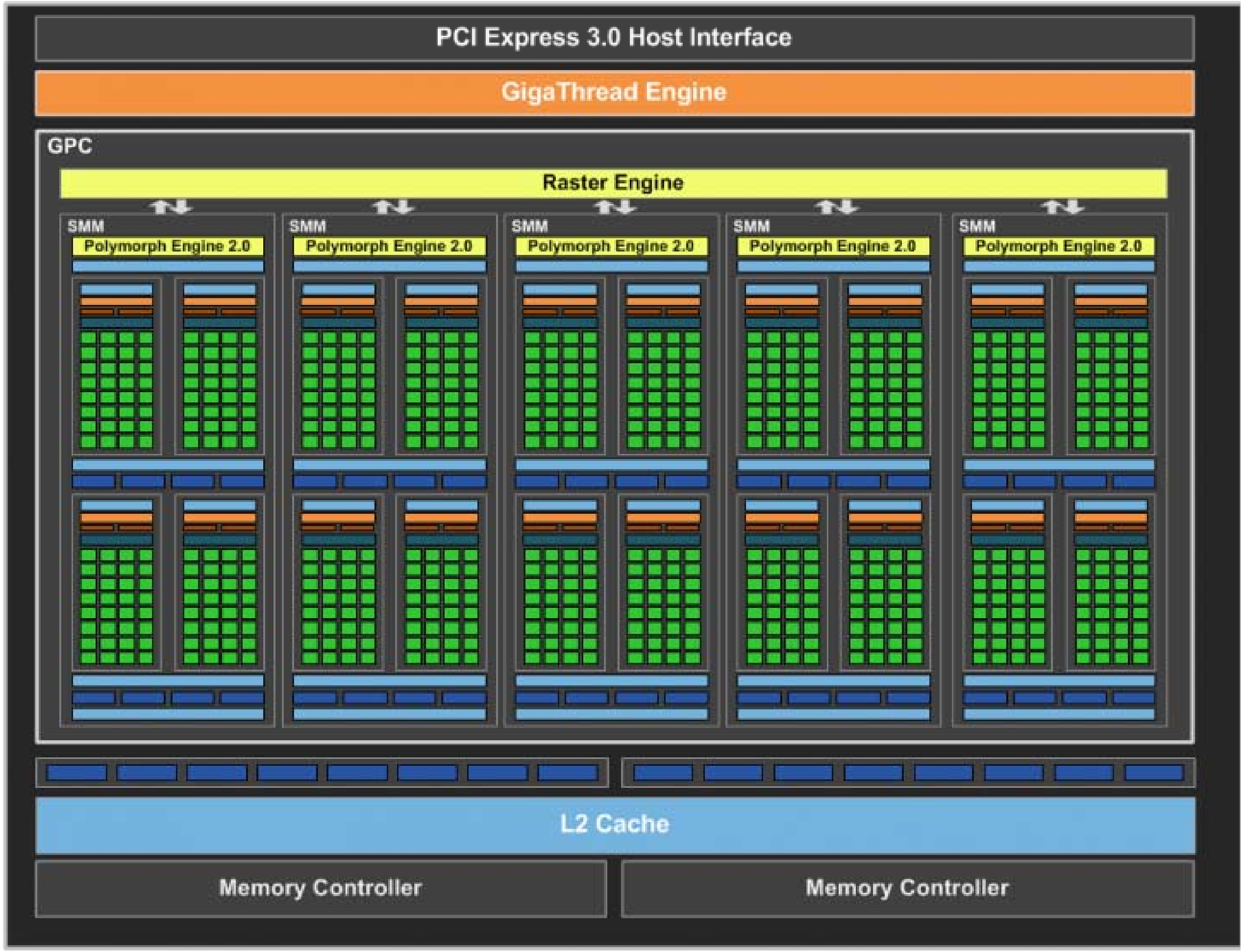


Memory Access Speed

- Register – dedicated HW – single cycle
- Shared Memory – dedicated HW – single cycle
- Local Memory – DRAM, no cache – *slow*
- Global Memory – DRAM, no cache – *slow*
- Constant Memory – DRAM, cached, 1…10s…100s of cycles, depending on cache locality
- Texture Memory – DRAM, cached, 1…10s…100s of cycles, depending on cache locality
- Instruction Memory (invisible) – DRAM, cached

GPU Architecture Review



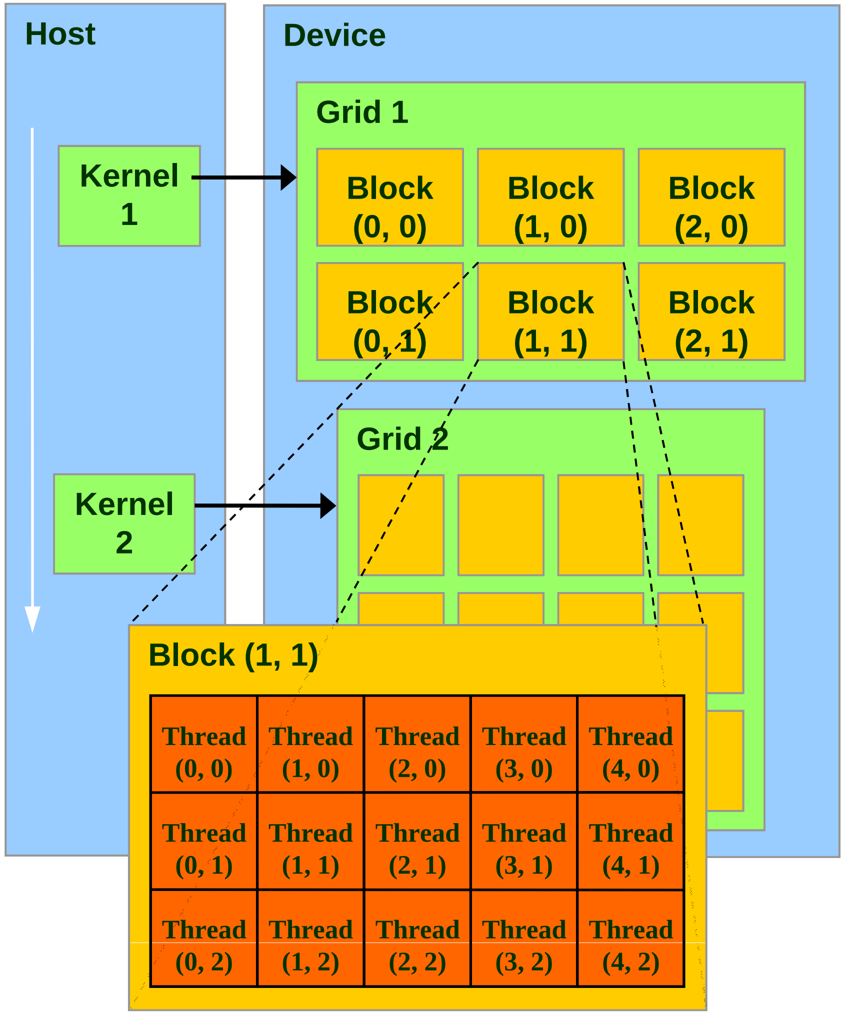


CUDA Programming Model

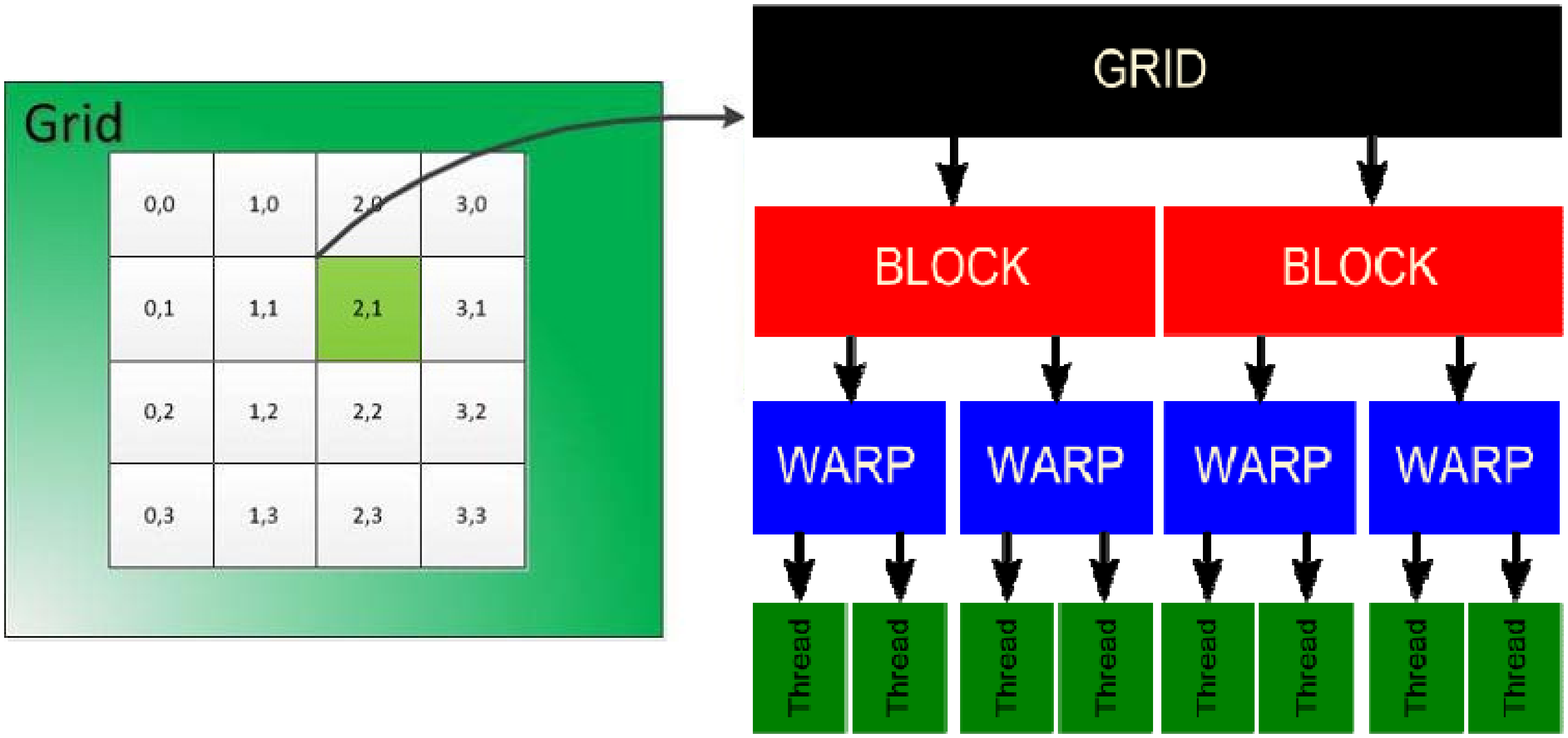
- The GPU is viewed as a compute device that:
  - Is a coprocessor to the CPU or host
  - Has its own DRAM (device memory)
  - Runs many threads in parallel
    - Hardware switching between threads (in 1 cycle) on long-latency memory reference
    - Overprovision (10000s of threads) → hide latencies
- Data-parallel portions of an application are executed on the device as kernels which run in parallel on many threads
- Differences between GPU and CPU threads
  - GPU threads are extremely lightweight
    - Very little creation overhead
  - GPU needs 10000s of threads for full efficiency
    - Multi-core CPU needs only a few

Thread Batching: Grids and Blocks

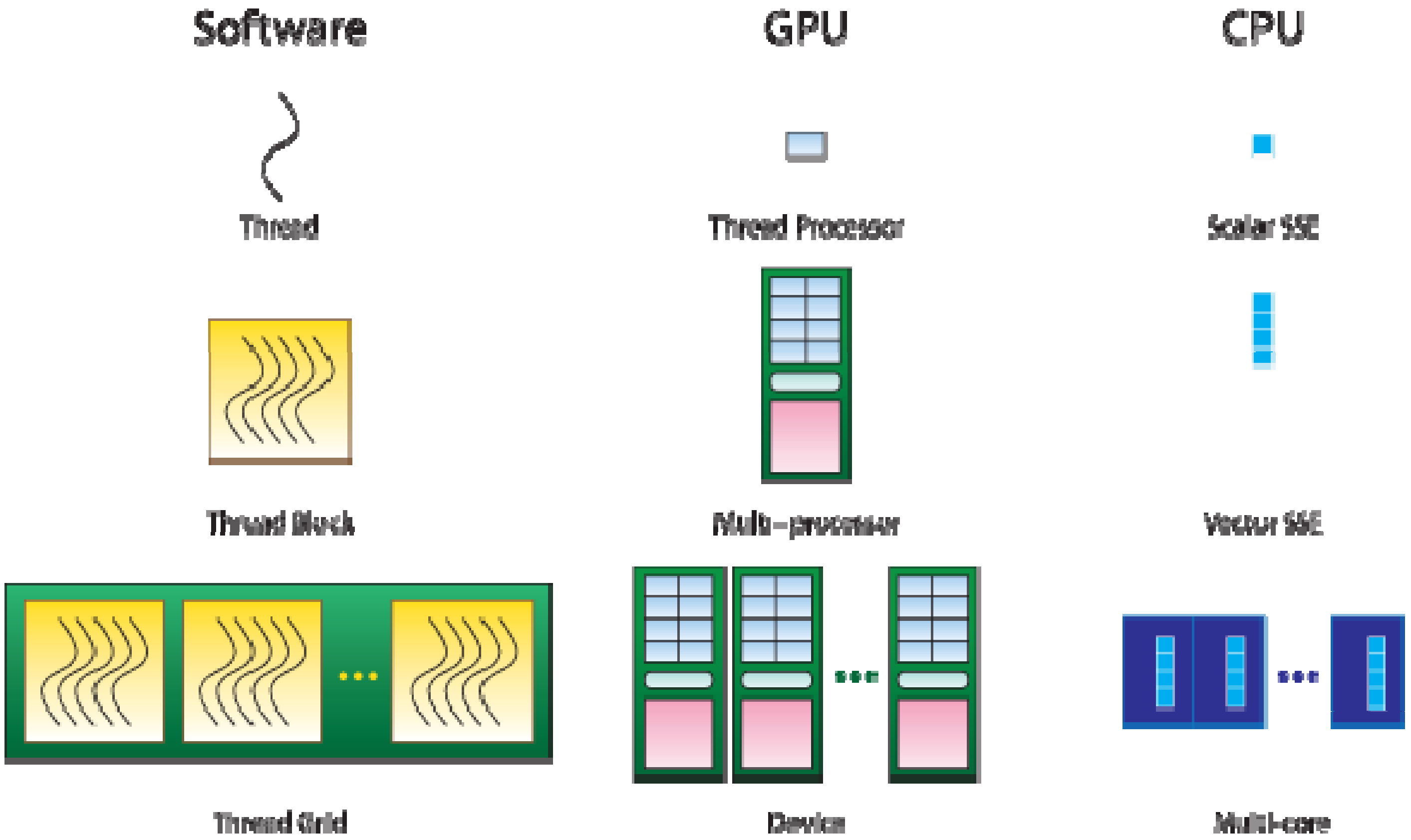
- Kernel executed as a grid of thread blocks
  - All threads share data memory space
- Thread block is a batch of threads, can cooperate with each other by:
  - Synchronizing their execution:
    For hazard-free shared memory accesses
  - Efficiently sharing data through a low latency shared memory
- Two threads from two different blocks cannot cooperate
  - (Unless thru slow global memory)



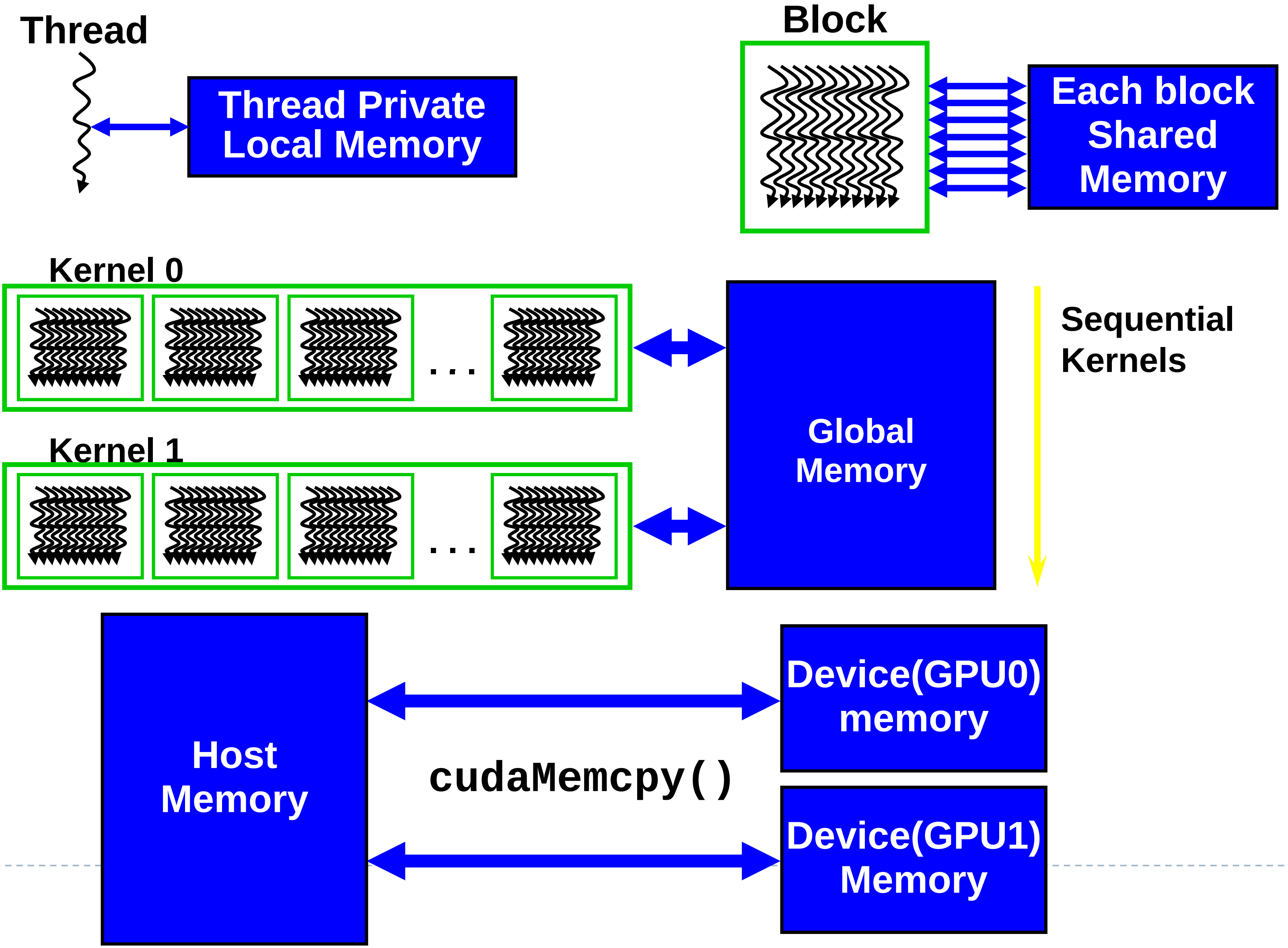
GPU Threads Organization



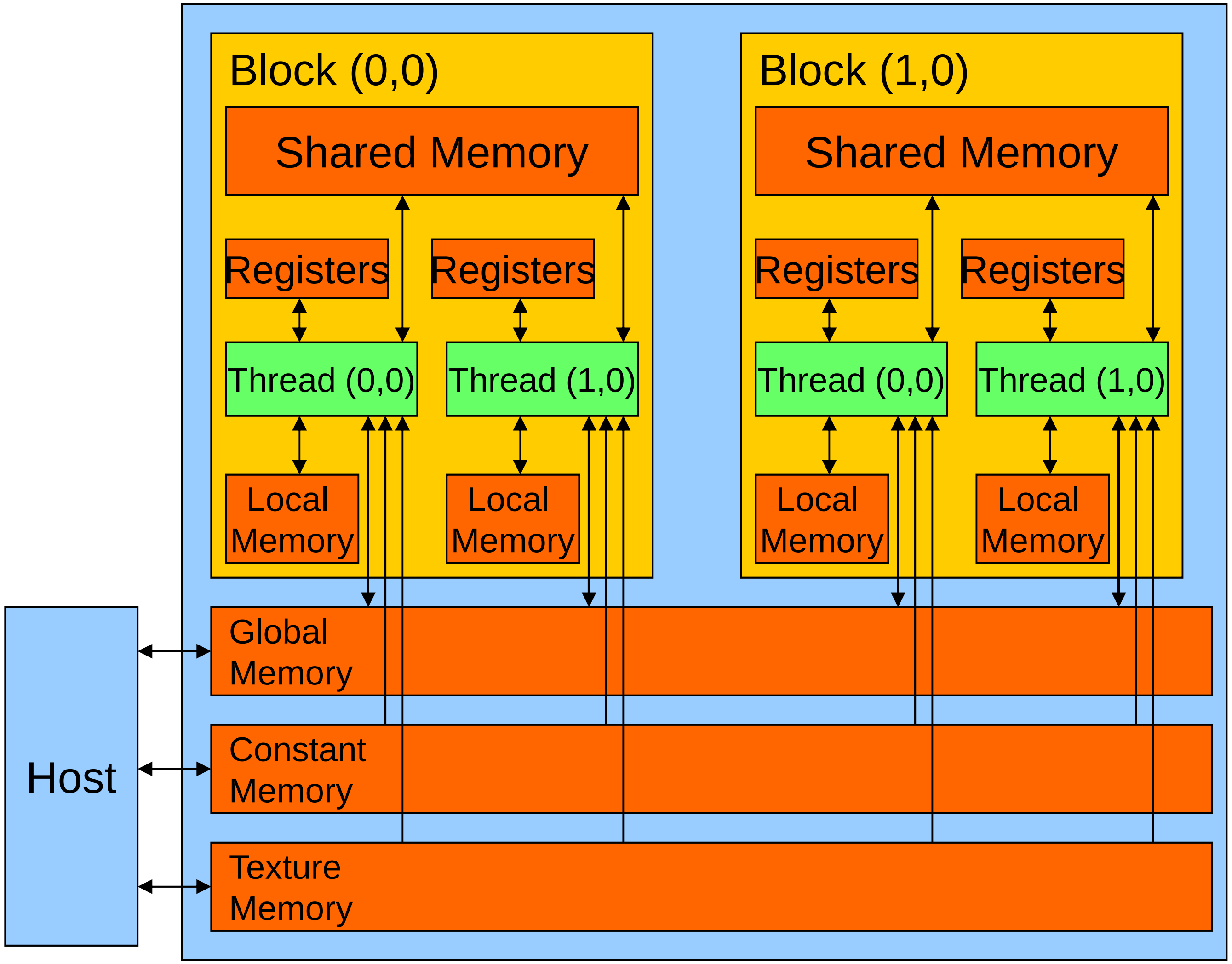
GPU Threads Mapping to Hardware



GPU Memory with Threads

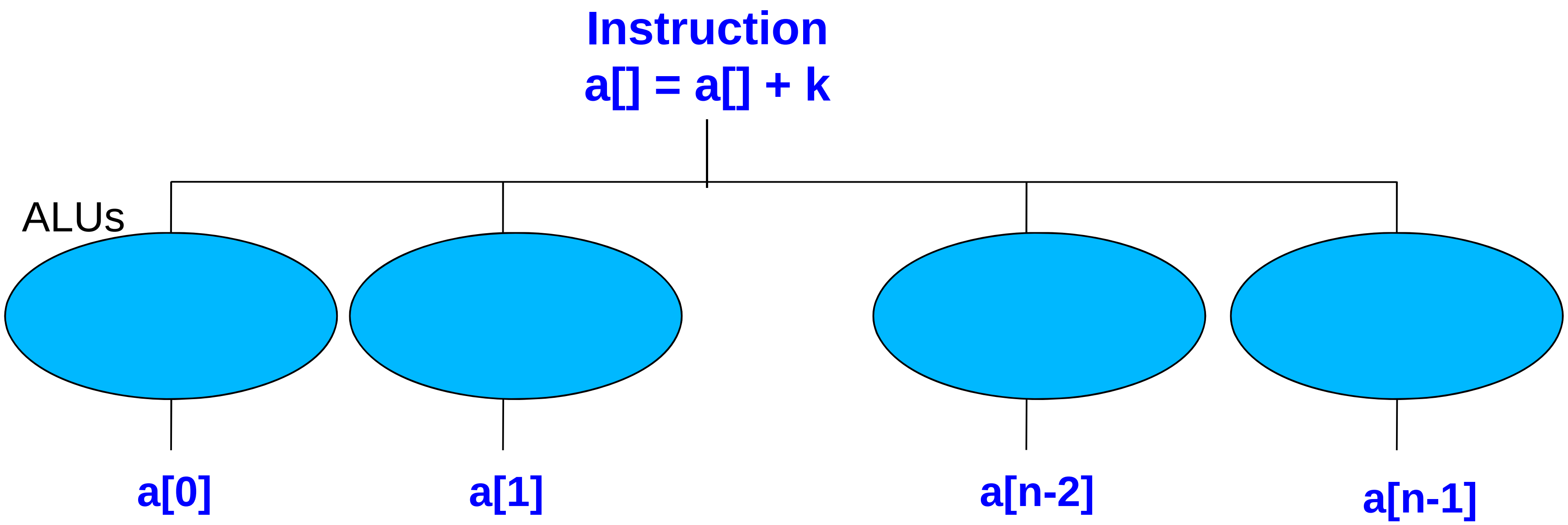


GPU Memory Hierarchy Recall



SIMD (Single Instruction Multiple Data)

Similar Idea with Data Partition/Different Level



Extended C

- **Declspecs**
  - **global, device, shared, local, constant**
- **Keywords**
  - **threadIdx, blockIdx**
- **Intrinsics**
  - **__syncthreads**
- **Runtime API**
  - **Memory, symbol, execution management**
- **Function launch**

```
__device__ float filter[N];

__global__ void convolve (float *image)  {

  __shared__ float region[M];
  ...

  region[threadIdx] = image[i];


  __syncthreads()
  ...
  image[j] = result;
}

// Allocate GPU memory
void *myimage = cudaMalloc(bytes)


// 100 blocks, 10 threads per block
convolve<<<100, 10>>> (myimage);
```

CUDA Function Declarations

	Executed on the:	Only callable from the:
`__device__ float DeviceFunc()`	device	device
`__global__ void KernelFunc()`	device	Host
`__host__ float HostFunc()`	host	Host

- `__global__` defines a kernel function
  - Must return `void`
- `__device__` and `__host__` can be used together