Parallel Communication Patterns

Map - one to one correspondence between input and output

Gather(many to one) - Gather data from multiple locations and then write it to one memory location(convolution)

Scatter(ont to many) - Data form a single location is written to multiple locations

(Several threads attempt to write at the same place at the same time) - may cause problems

Stencil(several to one) - Date Reuse - Many Thread accessing and using the same data

Transpose - Reorder Data elements in Memory

Reduce

Sort/Scan

Sharing Memory//Safely - GPU Hardware -

Summary of Programming Model -

Kernels - C/C++ function - Performed by many threads

Each kernel may have different number of threads per thread blocks

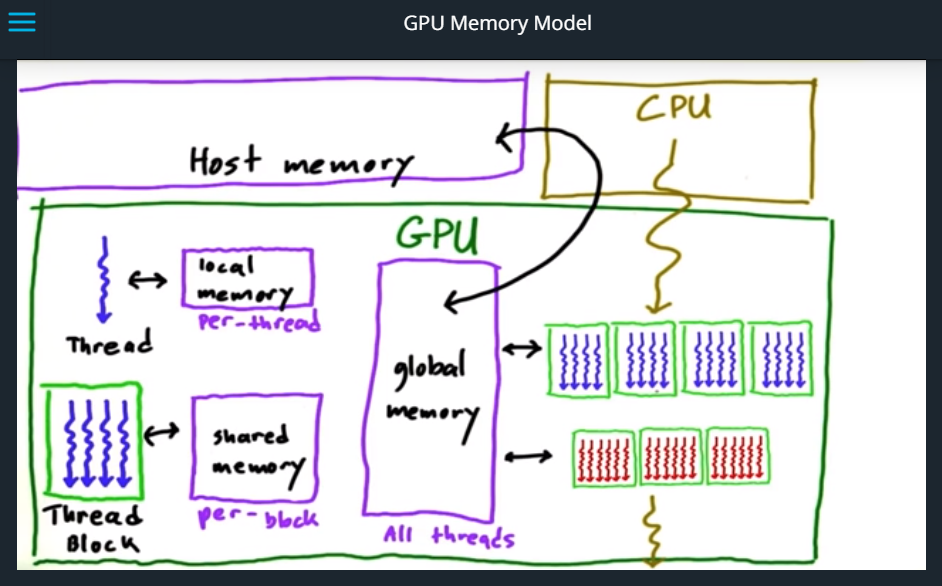
Thread Blocks and GPU Hardware -

* CUDA GPU - Streaming Multiprocessors (SM)
* SM -> Simple Processors/Memory
* GPU is responsible for allocating blocks to SMs
* All the SMs run in parallel and independently.
* A SM may run multiple blocks but A block cannot be run on multiple SMs
* Thread in different thread blocks must not cooperate with each other.
* Threads in a single thread block may cooperate with each other
* Programmer cannot specify any order of execution for the thread blocks.
* No assumptions about what block will run on which SM
* No communication between blocks - “DeadLocks”
* // force the printf()s to flush
* cudaDeviceSynchronize();

When and Where - CUDA gaurantees -

* Al threads in a block run on the same SM at the same time
* All blocks in a kernel finish before any blocks from the next kernel run

Memory Model -

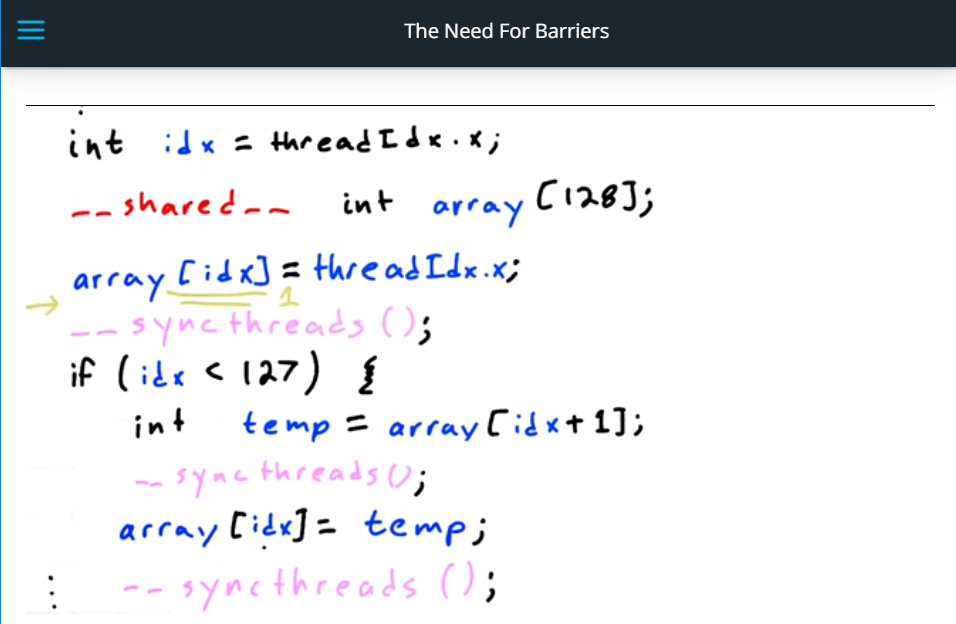
1. Local Memory - local variables/parameters
2. Shared Memory
3. Global Memory

Synchronization -

Threads need to synchronize.

Barrier - Point in the program where all the threads stop and wait for others, and when all the threads have arrived at the barrier than the move further.

Use \_\_syncthreads() to put barrier - sync threads within a block.. Why?



CUDA a hierarchy of

* Computation
* Memory

**Minimize time spent on memory -**

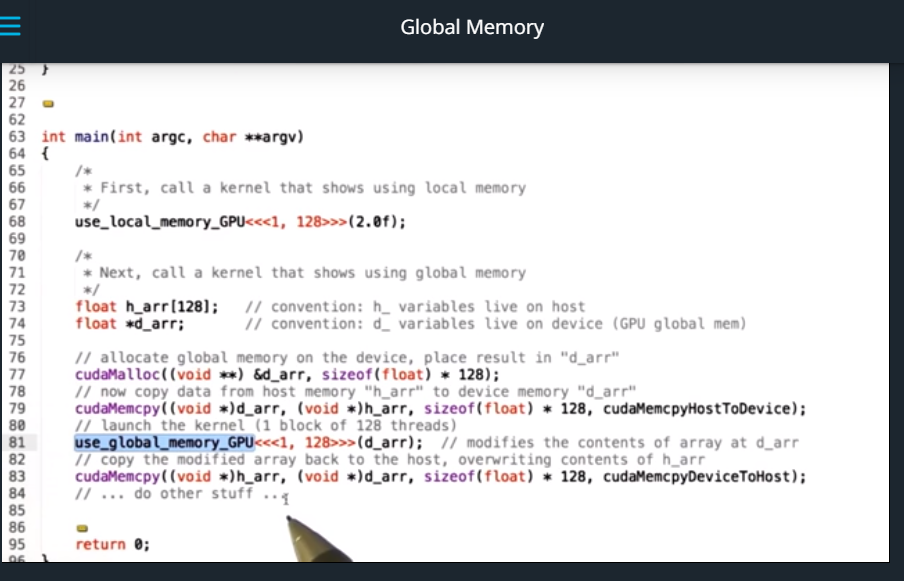
1. Move frequently-accessed data to fast memory

Local > shared >> global >>CPU(host) (In terms of speed)

Local - In register or in L2 cache

Convention - h\_ -> host

D\_ -> device



* Coalesce global memory accesses
  + GPU most efficient when threads read or write contiguous memory locations.

Atomics - Read Guide for different atomic functions atomicAdd(ptr, val) for example.

Limitations -

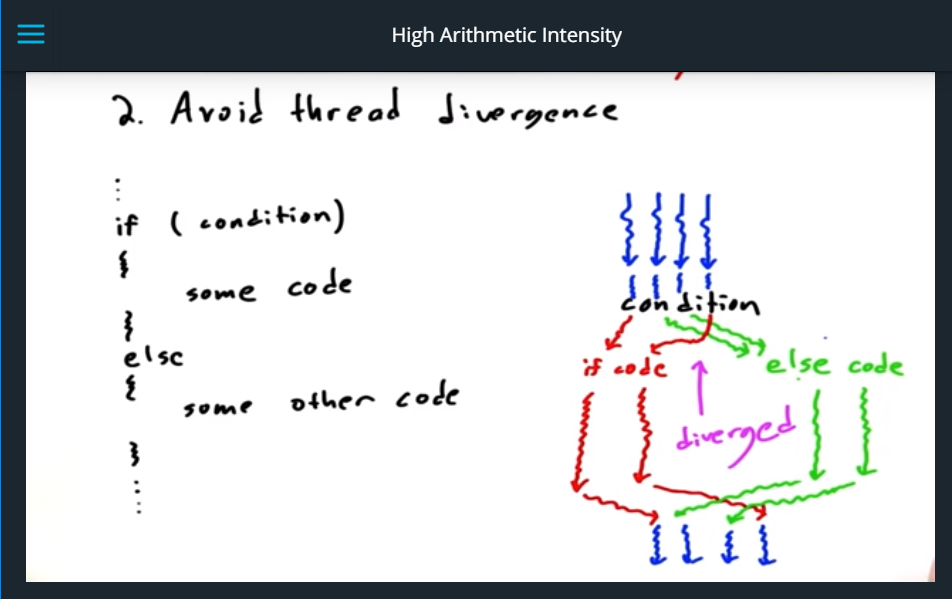
* Only certain operations, data types(integers)
* Implement any atomic using CAS()
* Still no ordering constraints. (Non associativity of floating points)
* Serializes access to memory
* Atomics take time!

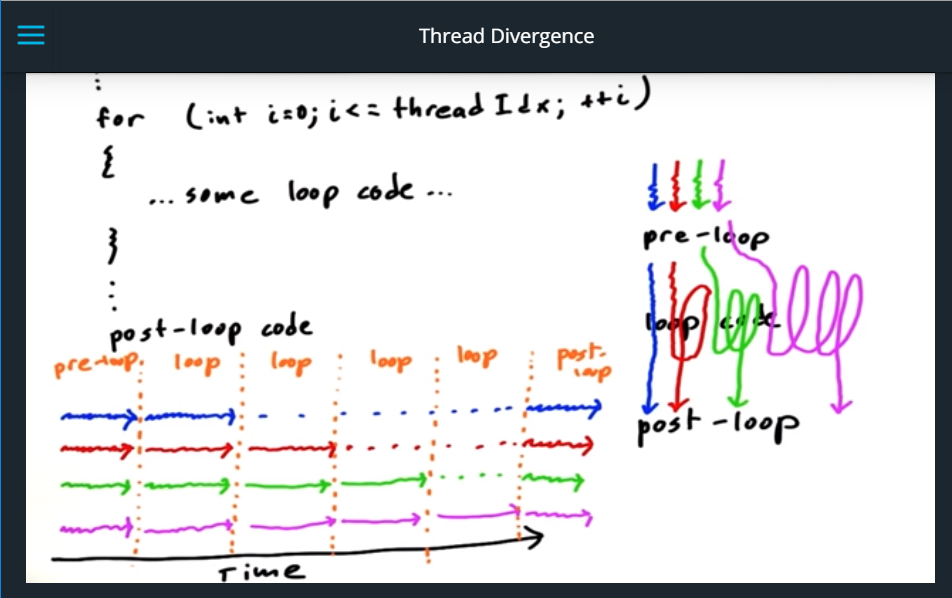
Thread Divergence -

When kernel have if statements / loops.

In loops some threads just sit around without doing anything.

Loop divergence slows the execution down.





Summary -

