**Overview of GPU Hardware and Memory Hierarchies**

GPUs (Graphics Processing Units) are specialized hardware designed for high-throughput parallel processing, particularly suited for data-parallel tasks such as image processing, machine learning, and scientific simulations. Their performance is heavily influenced by their hardware structure and memory hierarchy.

**1. GPU Hardware Architecture Overview**

Key Components:

* **Streaming Multiprocessors (SMs):** Core processing units in a GPU, each containing multiple CUDA cores.
* **CUDA Cores:** Simple arithmetic logic units (ALUs) that perform floating-point and integer operations.
* **Warp Scheduler:** Schedules groups of 32 threads (warps) to execute instructions in lockstep.
* **Shared Memory:** Fast on-chip memory shared among threads in a block.
* **Registers:** Fastest memory; used for storing temporary thread-specific variables.
* **Global Memory (DRAM):** Off-chip memory, accessible by all threads.
* **Texture/Constant Memory:** Special read-only memory with caching for faster repeated access.

Example NVIDIA GPU Architecture:

* 64–128 SMs
* Each SM has:
  + 64–128 CUDA cores
  + Shared memory (e.g., 64 KB per SM)
  + 256 KB L1 cache (merged with shared memory)
* L2 Cache shared among all SMs
* Global memory: 16–48 GB GDDR6 or HBM2

**2. GPU Memory Hierarchy**

Memory Levels and Characteristics:

| **Memory Type** | **Scope** | **Access Speed** | **Visibility** | **Usage Example** |
| --- | --- | --- | --- | --- |
| Register | Per-thread | Fastest (~1 cycle) | Private to thread | Temporary thread variables |
| Shared Memory | Per-block (SM) | Fast (~10 cycles) | Shared among threads | Intermediate results in block |
| Global Memory | Device-wide | Slow (400–800 cycles) | All threads | Input/output data for kernels |
| Constant Memory | Device-wide (read-only) | Faster than global (cached) | All threads | Constants used by all threads |
| Texture Memory | Device-wide | Cached | Threads with spatial access | Images, matrices |
| Local Memory | Per-thread (in global space) | Slow | Private to thread | Spill-over from registers |

Memory Access Tip: Threads should access consecutive memory addresses (coalesced access) to maximize throughput.

**3. Warp Execution and SIMD**

* GPUs use the SIMT (Single Instruction, Multiple Threads) model.
* A warp is a group of 32 threads that execute the same instruction.
* Divergence (e.g., using if-else within a warp) can cause serialization and degrade performance.

**4. Data Transfer Considerations**

| **Transfer Type** | **Interface** | **Latency / Bandwidth** |
| --- | --- | --- |
| Host to Device (CPU↔GPU) | PCIe or NVLink | High latency, limited bandwidth |
| Device Internal Transfers | L2, SM links | Low latency, high bandwidth |

Optimization Note: Minimize data transfer between CPU and GPU; do as much computation as possible on the GPU.

**5. Performance Optimization Strategies**

* Use shared memory for frequently reused data.
* Minimize global memory access and overlap it with computation using streams.
* Align data for coalesced memory access.
* Avoid warp divergence in control structures.

**6. Summary Table**

| **Memory Level** | **Scope** | **Speed** | **Typical Use** |
| --- | --- | --- | --- |
| Registers | Thread | Fast | Local thread variables |
| Shared Memory | Block (SM) | Fast | Shared calculations in blocks |
| L1 / L2 Cache | SM / Device | Medium | Temporary caching |
| Global Memory | Device-wide | Slow | Input/output data |
| Constant Memory | Device-wide | Cached | Global read-only constants |
| Host Memory | Host CPU | Very Slow | Initial/final data transfer |