Introduction to Parallel Architectures

5 Hours

GPU as Parallel Computers

Since 2003, the semiconductor industry has settled on two main trajectories for designing microprocessor (MP):

1. The multi-core trajectory:

- It seeks to maintain the execution speed of sequential programs while moving into multiple cores.

- *Multicore trajectory* began as two-core processors, with the number of cores doubling every generation of MP.
  - Eg: Intel Corei7 MP which has 4 processor cores, each of which is an out-of-order, multiple instruction issue processor implementing the full x86 instruction set.
  - This MP supports *hyperthreading* with two *hardware threads* per core and is designed to maximize the exec speed of sequential programs.

2. The many-core (many-thread) trajectory:

- It focuses more on the *execution throughput* of parallel applications

- The many-cores began as a *large number of much smaller cores*, and, once again, the number of cores doubles with each generation
  - Eg: The NVIDIA GeForce GTX 280 graphics processing unit (GPU) with 240 cores, each of which is a heavily multithreaded

- Many-core processors, especially the GPUs, dominated the race of Floating-point (FP) performance. The ratio between many-core GPUs and multicore CPUs for peak floating-point calculation throughput is about 10 to 1.

GPU vs CPU FP performance

- This gap has motivated many app developers to *move the computationally intensive parts of their software to GPUs for execution*.

- When there is more work to do, there is more opportunity to divide the work among cooperating parallel workers.

Why there is such a large performance gap between many-core GPUs and general-purpose multicore CPUs?

- The answer lies in the differences in the fundamental design philosophies between the two types of processors

GPU vs CPU architecture

CPU design principle:

- The design of a CPU is optimized for sequential code performance

- It makes use of sophisticated control logic to allow instructions from a single thread of execution to execute in parallel

- The large cache memories are provided to reduce the instruction and data access latencies of large complex applications → *latency-oriented design*

- Memory bandwidth limit the speed of applications by limiting the rate at which data can be delivered from the memory system to processors.

Why there is such a large performance gap between many-core GPUs and general-purpose multicore CPUs?

GPU design principle:

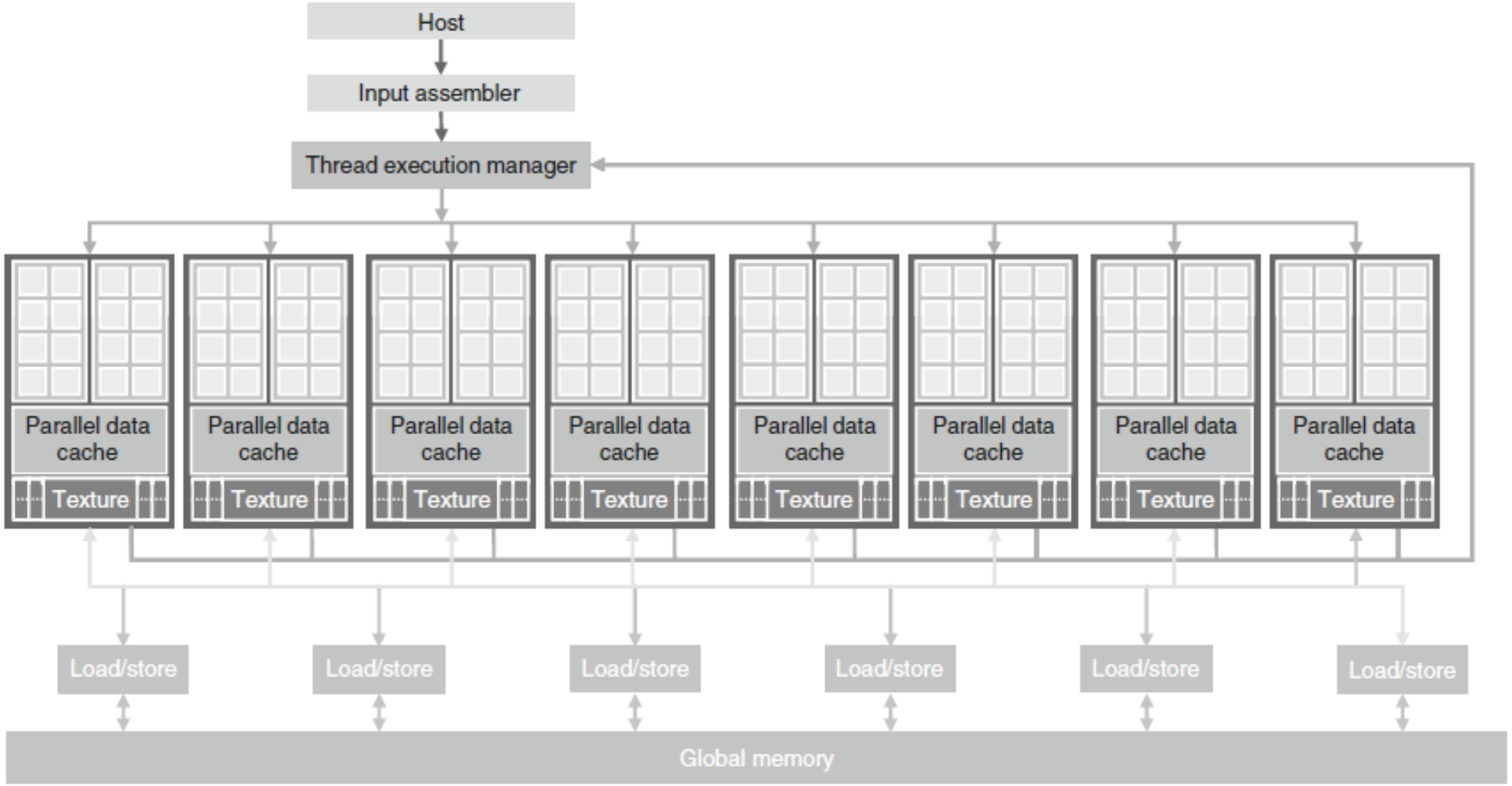
- The design philosophy of the GPUs is shaped by the fast growing video game industry, which requires the ability to perform a massive number of floating-point calculations per video frame

- GPU performs well by *executing massive numbers of threads*

- GPU hardware takes advantage of a large number of execution threads to find work to do when some of them are waiting for long-latency memory accesses

- Small cache memories are provided to help the bandwidth requirements of applications, so multiple threads that access the same memory data need not always access the DRAM → *throughput-oriented design*

- Most apps will use both CPUs and GPUs, executing the sequential parts on the CPU and numerically intensive parts on the GPUs



Architecture of a CUDA-capable GPU.

Architecture of a Modern GPU

Modern GPU architecture

- It is organized into an array of highly threaded *streaming multiprocessors* (SMs)
- In the figure, two SMs form a building block; however, the number of SMs in a building block can vary from one generation of CUDA GPUs to another generation
- Each SM has a number of *streaming processors* (SPs) that share control logic and instruction cache
- Each GPU currently comes with up to 4 GB of graphics double data rate (GDDR) DRAM, referred to as *global memory*

GDDR DRAMs

- The GDDR DRAMs differ from the system DRAMs on the CPU motherboard in that they are essentially the frame buffer memory that is used for graphics processing
- For graphics applications, GDDR DRARs hold *video images*, and *texture information* for *three-dimensional (3D) rendering*
- For normal computing they function as *very-high-bandwidth, off-chip memory*, though with somewhat more latency than typical system memory. **For massively parallel applications, the higher bandwidth makes up for the longer latency**

CUDA-paper