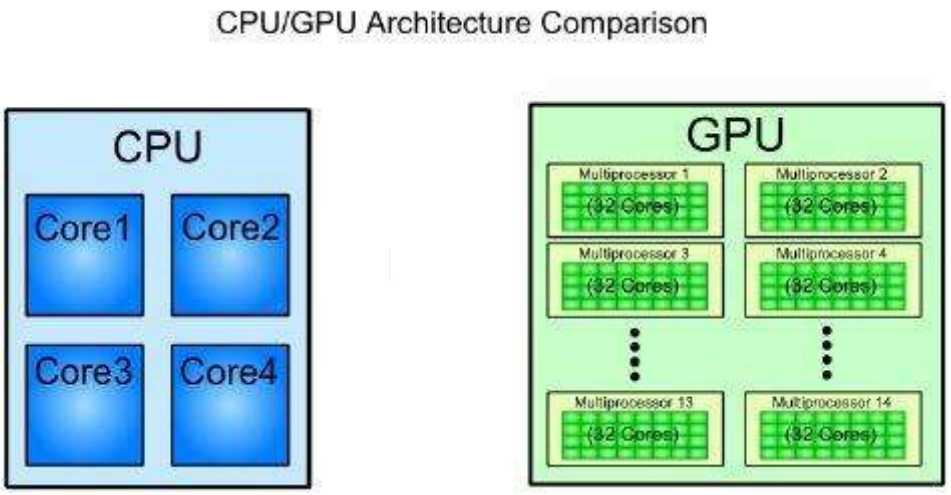
5.1. Introduction to GPU Computing

What Is GPU Computing?

GPU computing is the use of graphical processing units (GPUs) as **coprocessors** to CPUs. This helps to increase the compute power in high-performance and **complex** computing environments.

Many-core processors that are designed to operate on large chunks of data, in which CPUs prove inefficient. A GPU comprises **many cores** (that almost double each passing year), and **each core** runs at a clock speed significantly **slower** than a CPU's clock. GPUs focus on execution throughput of **massively-parallel** programs. For example, the **Nvidia GeForce GTX 280 GPU has 240 cores,** each of which is a heavily **multithreaded**, in-order, single-instruction issue processor (**SIMD** − single instruction, multiple-data) *that shares its control and instruction cache with seven other cores*. When it comes to the total Floating Point Operations per Second (FLOPS), GPUs have been leading the race for a long time now.



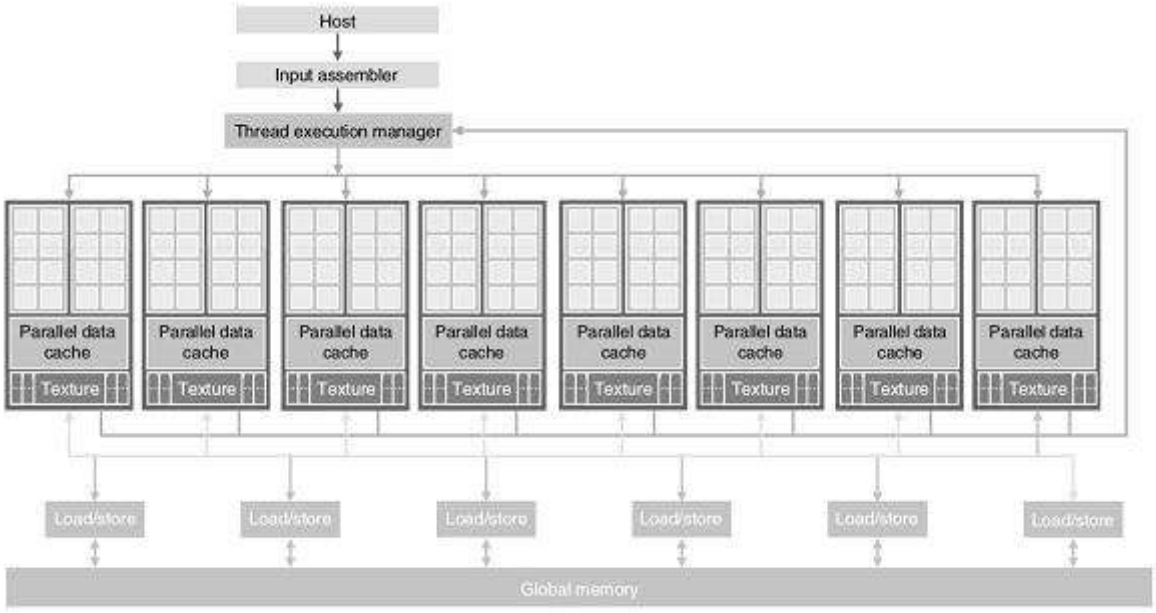
GPUs do not have **virtual** memory, **interrupts**, or means of addressing devices such as the keyboard and the mouse. They are terribly inefficient when we do not have **SPMD** (Single Program, Multiple Data). Such programs are best handled by CPUs, and may be that is the reason why they are still around. For example, a CPU can calculate a **hash** for a string much, much faster than a GPU, but when it comes to computing several **thousand hashes**, the GPU wins. As of data from 2009, the ratio b/w GPUs and multi-core CPUs for peak **FLOP** calculations is about 10:1. Such a large performance gap forces the developers to outsource their data-intensive applications to the GPU.

GPUs are designed for **data intensive applications.** This is emphasized-upon by the fact that the **bandwidths** of GPU **DRAM**(Dynamic Random Access Memory) has increased tremendously by each passing year, but not so much in case of CPUs. Why **GPUs** adopt such a design and CPUs do not? Well, because GPUs were originally designed for **3D** rendering, which requires holding **large** amount of texture and polygon data. Caches cannot hold such large amount of data, and thus, the only design that would have increased rendering performance was to **increase** the **bus** width and the **memory clock**. For example, the Intel **i7,** which currently supports the largest memory bandwidth, has a memory bus of width 192b and a memory clock upto 800MHz. The **GTX 285** had a bus width of 512b, and a memory clock of 1242 MHz.

CPUs also would **not** benefit greatly from an increased memory bandwidth. Sequential programs typically do not have a 'working set' of data, and most of the required data can be stored in L1, L2 or **L3 cache**, which are **faster** than any **RAM**. Moreover, CPU programs generally have more **random memory access** patterns, unlike massively-parallel programs, that would not derive much benefit from having a wide memory bus.

GPU Design

Here is the architecture of a **CUDA** (or Compute Unified Device Architecture) capable GPU −



There are 16 streaming multiprocessors (SMs) in the above diagram. Each SM has 8 streaming processors (SPs). That is, we get a total of 128 SPs. Now, each SP has a MAD unit (Multiply and Addition Unit) and an additional MU (Multiply Unit). The GT200 has 240 SPs, and exceeds 1 TFLOP of processing power.

Each SP is massively threaded, and can run thousands of threads per application. The G80 card supports 768 threads per SM (note: not per SP). Since each SM has 8 SPs, each SP supports a maximum of 96 threads. Total threads that can run: 128 * 96 = 12,228. This is why these processors are called 'massively parallel'.

The G80 chips has a memory bandwidth of 86.4GB/s. It also has an 8GB/s communication channel with the CPU (4GB/s for uploading to the CPU RAM, and 4GB/s for downloading from the CPU RAM).

Memories

At this point, it becomes essential that we understand the difference between different types of memories.

DRAM stands for Dynamic RAM. It is the most common RAM found in systems today, and is also the slowest and the least expensive one. The RAM is named so because the information stored on it is lost, and the processor has to refresh it several times in a second to preserve data.

SRAMstands for Static RAM. This RAM does not need to be refreshed like DRAM, and it is significantly faster than DRAM (the difference in speeds comes from the design and the static nature of these RAMs). It is also called the microprocessor cache RAM.

So, if SRAM is faster than DRAM, why are DRAMs even used? The primary reason for this is that for the same amount of memory, SRAMs cost several times more than DRAMs. Therefore, processors do not have huge amount of cache memories. For example, the Intel 486 line microprocessor series has 8KB of internal SRAM cache (on-chip). This cache is used by the processor to store frequently used data, so that for each request, it does not have to access the much slower DRAM.

VRAM (Video RAM) stands for Video RAM. It is quite similar to DRAM but with one major difference: it can be written to and read from simultaneously. This property is essential for better video performance. Using it, the video card can read data from VRAM and send it to the screen without having to wait for the CPU to finish writing it into the global memory. This property is of not much use in the rest of the computer, and therefore, VRAM are almost always used in video cards. Also, VRAMs are more expensive than DRAM, and this is one of the reasons why they have not replaced DRAMs yet.

The ratio of floating-point calculation to global memory access operation is 1:1, or 1.0. We will refer to this ratio as the compute to global memory access (CGMA) ratio, defined as the number of floating-point calculations performed for each access to the global memory within a region of a CUDA program.