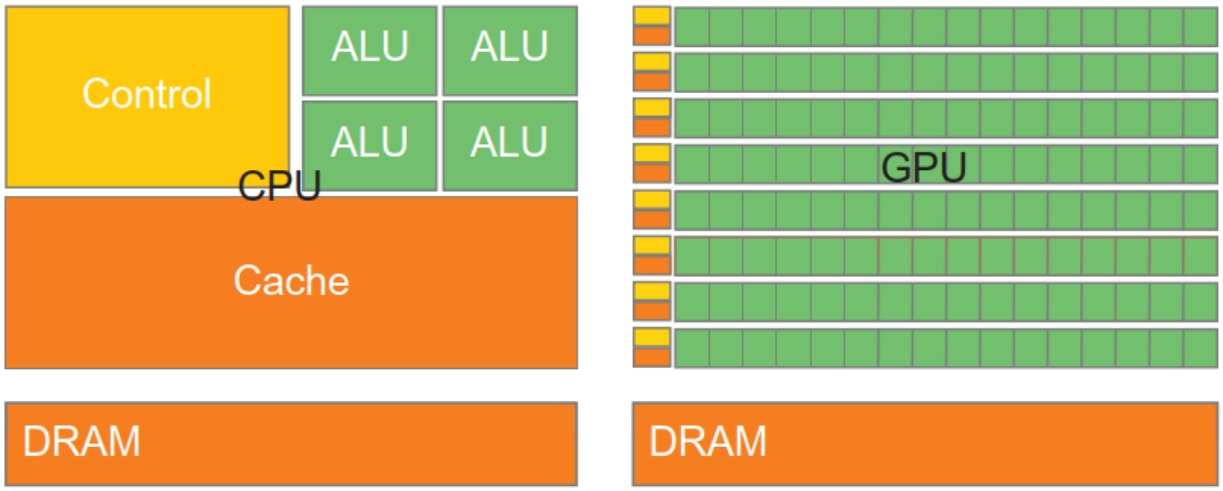
# GPU Series – I

# CPU vs GPU




(Img Src: Programming Massively Parallel Processors by David B.Kirk, Wen-mei W.Hwu)

**Let's see the basic difference between a CPU and a GPU.**

- CPU is a general-purpose processor designed for a wide range of tasks, highlighting a few powerful cores optimized for low-latency, sequential processing with complex control logic and large caches.
- A GPU is specialized for parallel processing tasks, such as graphics rendering and data-intensive computations, with hundreds to thousands of simpler cores optimized for high throughput.
- While CPUs are great for tasks that require fast, single-threaded performance and complex decision-making, GPUs are good in handling many tasks simultaneously, making them ideal for applications like 3D rendering, machine learning, and scientific simulations.
- The CPU focuses on versatility and quick execution of individual tasks, whereas the GPU focuses on maximizing the total execution throughput of numerous parallel tasks.

Architecturally,

| Aspect | CPU | GPU |
|---|---|---|
| **Core Architecture** | Few powerful cores designed for complex tasks. | Hundreds to thousands of simpler cores designed for high parallelism. |
| **Control Unit** | Sophisticated control units with branch prediction, out-of-order execution, and advanced instruction pipelining. | Simpler control logic optimized for parallel processing rather than complex decision-making. |
| **Cache System** | Large multi-level caches (L1, L2, and sometimes L3) to minimize memory access latency. | Smaller, specialized caches for managing massive data throughput. |
| **Execution Model** | Optimized for low-latency execution, handling fewer tasks very quickly. | Optimized for high throughput, capable of executing many tasks concurrently. |
| **Memory Access** | Complex memory hierarchy to ensure fast access to data. | High-bandwidth memory systems designed to handle large volumes of data efficiently (e.g., GDDR6). |

| | | |
|---|---|---|
| **Parallelism** | Focuses on instruction-level parallelism (ILP) and simultaneous multithreading (SMT). | Focuses on data-level parallelism (DLP) and thread-level parallelism (TLP). |

Understanding the architectural differences between CPUs and GPUs highlights why GPUs are particularly well-suited for tasks involving massive data throughput and parallel processing. Unlike CPUs, which are optimized for low-latency, sequential task execution, GPUs are designed to handle numerous parallel tasks simultaneously, making them ideal for graphics rendering and other data-intensive computations. This architectural difference is crucial when considering the specific demands placed on a GPU, particularly in the context of moving large amounts of data to and from its main DRAM to meet the requirements of the graphics frame buffer.

A GPU must be capable of moving extremely large amounts of data in and out of its main DRAM because of graphics frame buffer requirements. The frame buffer is a dedicated block of memory that holds the image data that is currently being displayed or is about to be displayed on the screen. This includes:

- **Color Data**: The color information for each pixel.
- **Depth Data**: Information about the distance of each pixel from the viewer (used for 3D rendering).
- **Stencil Data**: Used for masking certain parts of the rendering process.
- **Accumulation Data**: Used for special effects like motion blur.

The frame buffer must be large enough to store this data for the entire display resolution, and it needs to be updated rapidly to maintain smooth video playback and responsive graphics rendering.

**What are these graphics frame buffer requirements?**

Imagine you're watching a high-definition video or playing a video game. The GPU has to keep track of every tiny dot (pixel) on your screen, including its color and depth in a 3D space. All this data is stored in the frame buffer, a special part of the GPU's memory. To ensure that what you see is smooth and realistic, the GPU must quickly move this large amount of data in and out of its memory.

Another critical aspect of GPU performance involves floating-point calculations per video frame.

**What is Floating-Point Calculations per video frame? How does it depend on the processors/computation?**

Floating-point calculations per video frame refer to the number of mathematical operations involving floating-point numbers (numbers with decimals) that the GPU must perform to render a single frame of video. This includes calculations for lighting, shading, physics simulations, texture mapping, and more. The complexity and quality of these calculations depend on the GPU's processing power and architecture.

Given the extensive computational requirements of rendering each frame, GPUs need to balance two critical performance metrics: latency and throughput.

**Reducing Latency vs Increase Throughput, which is better and why?**

- **Latency** refers to the time it takes to complete a single task from start to finish.
- **Throughput** is the number of tasks that can be completed in a given amount of time.

Reducing latency is beneficial for tasks that require quick response times (like interactive applications), while increasing throughput is better for tasks that benefit from handling many operations simultaneously (like data processing).

- **Reducing Latency**: Often requires high-speed, specialized hardware which can consume more power and occupy more chip area.
- **Increasing Throughput**: Can be achieved by parallel processing, which can be more power-efficient and use chip area more effectively.

Think of latency as the time it takes to get a single pizza delivered to your house. Throughput, on the other hand, is like the total number of pizzas that can be delivered in an hour. Reducing the time for a single delivery (low latency) might mean using a very fast, but expensive and fuel-hungry car. Delivering many pizzas in the same time (high throughput) might involve using several regular cars more efficiently. For GPUs, focusing on throughput is usually better because they can handle lots of tasks at once without needing super-fast individual components, saving power and space on the chip.

This brings us to the design aspect. Based on the CPU and GPU, we have either Throughput-oriented design or latency-oriented design. We shall talk about throughput-oriented design.

**What is Throughput-oriented design?**

Throughput-oriented design aims to maximize the total execution throughput of many threads, even if individual threads take longer to execute. GPUs are designed this way to handle the parallel nature of graphics and data processing tasks.

- **CPU Design**: Optimized for low latency to execute a single thread as quickly as possible.
- **GPU Design**: Optimized for high throughput to execute many threads concurrently, even if each thread runs slower compared to a CPU.

**More Threads for GPUs**: GPUs are powerful with more threads because they are designed to handle many parallel tasks, making the most out of their architecture.

**Less Threads for CPUs**: CPUs perform better with fewer threads because they are optimized for quick, sequential task execution with minimal latency.

For GPUs, having more threads (or tasks) to work on at the same time is beneficial, like having many assembly lines in a factory to produce multiple products simultaneously. For CPUs, fewer threads are better because they are like a master craftsman focusing on one project at a time, ensuring it's completed as quickly and efficiently as possible.

~~ To be Continued ~~

---

**Reference**: Programming Massively Parallel Processors by David B.Kirk, Wen-mei W.Hwu