CUDA

What is CUDA?

CUDA is a parallel computing platform and programming model created by **NVIDIA**. It allows developers to use **NVIDIA GPUs** to accelerate computing tasks, especially those that can be run in parallel.

Traditionally, CPUs handle general-purpose tasks. But CPUs have a small number of powerful cores. GPUs, on the other hand, have thousands of smaller, efficient cores designed to handle multiple operations at the same time — perfect for parallel tasks.

Why Use CUDA?

CUDA lets you **write code in C, C++, or Python** that runs on the **GPU**, instead of just the CPU. This is useful for:

- Scientific computing
- Machine learning
- Image/video processing
- Simulations
- Cryptography
- Big data analytics

CUDA Architecture Basics

Component	Description
Host	The CPU and its memory (RAM).
Device	The GPU and its memory.
Kernel	A function written in CUDA that runs on the GPU.
Thread	Smallest unit of execution on GPU.
Block	A group of threads.
Grid	A group of blocks.

When you run a kernel, you launch **many threads** organized into **blocks**, which are in turn grouped into a **grid**.

How CUDA Works

Example: Vector Addition

You want to add two vectors `A` and `B` of size `N`.

1. Allocate memory on the host (CPU).
2. Allocate memory on the device (GPU).
3. Copy data from host to device.
4. Call the CUDA **kernel** to perform vector addition (each thread adds one element).
5. Copy the result back to the host.
6. Free memory.

CUDA Code Structure

```
__global__ void addVectors(float *A, float *B, float *C, int N) {
    int idx = threadIdx.x + blockIdx.x * blockDim.x;
    if (idx < N) C[idx] = A[idx] + B[idx];
}
```

- `__global__` tells CUDA this function runs on the **device** (GPU).
- `threadIdx`, `blockIdx`, `blockDim` are built-in variables to locate a thread's position.

You launch this kernel like:

```
int threadsPerBlock = 256;
int blocksPerGrid = (N + threadsPerBlock - 1) / threadsPerBlock;
addVectors<<<blocksPerGrid, threadsPerBlock>>>(d_A, d_B, d_C, N);
```

Key Features

Memory Types

- **Global Memory**: Accessible by all threads, slow but large.
- **Shared Memory**: Faster, shared within a block.
- **Registers**: Fastest, but private to each thread.

Synchronization

CUDA provides synchronization methods to control thread behavior inside blocks.

Performance Benefits

CUDA lets you:

- Run thousands of threads in parallel.
- Achieve massive speedup on computationally heavy problems.
- Offload tasks to GPU while CPU handles others.

CUDA Toolkit

To write and run CUDA programs, install:

- **NVIDIA drivers**
- **CUDA Toolkit** (includes compiler `nvcc`, profiler, libraries)
- Optional: **cuDNN**, **Thrust**, **cuBLAS**, **cuFFT** for ML and numerical tasks.

Development Tools

- **Visual Studio (Windows)**
- **VS Code with CUDA extension**
- **Linux Terminal + `nvcc`**
- **Nsight Profiler** for performance analysis

Real-World Use Cases

- **NVIDIA RAPIDS**: Big Data on GPU
- **TensorFlow / PyTorch**: Deep learning libraries use CUDA
- **Blender / Adobe**: GPU acceleration for rendering
- **Medical Imaging**, **Oil & Gas Simulations**, etc.

Summary

Aspect	CPU	GPU (with CUDA)
Cores	Few, complex	Many, simple
Strength	Serial tasks	Parallel tasks
Coding	Regular C/C++	C/C++ with CUDA extensions
Ideal for	General-purpose apps	High-performance computation