GPU Programming using CUDA and C++

Yuan Wang

2025

Contents

Unlike the conventional CPU-based architecture, where CPU cores are much more limited, GPU has many more cores, even in consumer products. A comparison of CPU and GPU architecture is illustrated in Fig.~**??**. The terminologies "core" for both CPU and GPU mentioned earlier, though, have different implications. For instance, dual physical cores in an INTEL CPU can have up to four virtual threads, known as hyperthreading, and each physical core consists of an arithmetic logic unit (ALU), a logic control unit, and caches, while the NVIDIA CUDA core usually refers to a large number of execution units following the concept of "Single Instruction, Multiple Data" (SIMD)~[**?**].[1] Thanks to the SIMD feature, CUDA is used for the first time for deep convolutional neural networks, known as AlexNet, in image identification in 2012~[**?**].

Generally speaking, each thread of a CPU comes with an execution of a task at a fast speed, but is limited by the number of threads running concurrently; however, there are thousands of threads running in parallel in GPU but with a slower execution speed per thread. There are always advantages and disadvantages when choosing the preferable architecture for executing certain tasks.

Figure~**??** depicts the main difference between a CPU and a GPU in that unlike a CPU where each core has its control unit and can access the CPU's fast memory (cache), in a GPU, cores are organized into groups, with each group controlled by a single control unit and sharing access to a cache. Both the CPU and the GPU are connected to dynamic random-access memory (DRAM); the CPU directly accesses the system's DRAM, while the Nvidia GPU, designed for CUDA, operates using its dedicated memory, often referred to as GPU global memory[2], a specific variant of DRAM, to handle its computations.

Before attempting to go into detail, keep in mind that parallel computing is not confined to a particular type of hardware, such as a GPU. Instead, it is a concept that applies to any hardware capable of concurrent operations, including multicore CPUs, which can also perform parallel processing up to a certain extent. In other words, parallel programming is fundamentally about algorithm design rather than the specifics of the underlying hardware. For instance, in a parallel computing network, various units can be interconnected, and each unit can comprise different elements, such as CPUs, GPUs, Field Programmable Gate Arrays (FPGAs), or other computational units.

[1]The NVIDIA CUDA core usually means an execution unit that can perform a single-precision floating-point (FP32) operation per clock cycle, and there is no direct comparison of threads and cores between CPU and GPU.

[2]It is worthy noting that the memory configurations differ for different hardware archetectures. For example, the unified memory that the CPU and GPU can access is part of the Apple M1 series chips.