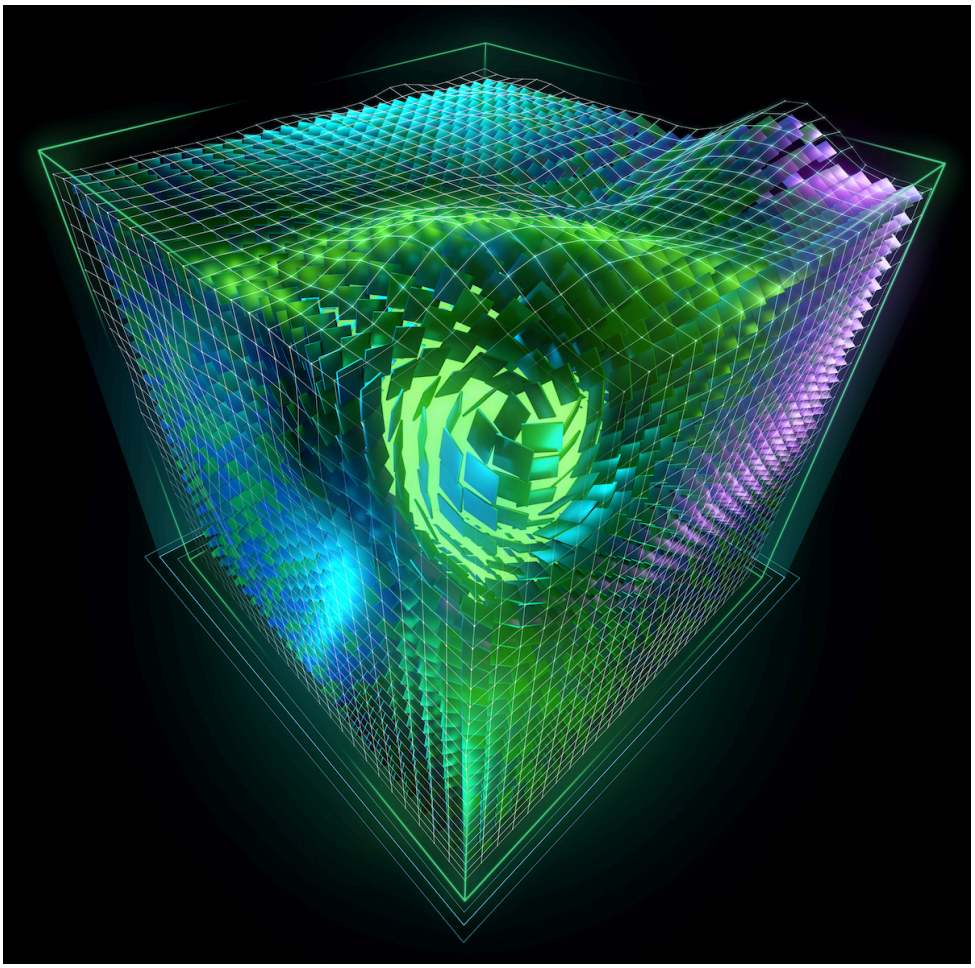
Chapter 1: GPU Computing Basics

Introduction

A graphics processing unit (GPU) is a specialized electronic circuit initially designed to accelerate computer graphics and image processing. Graphics processing units have specialized processing cores that you can use to speed computational processes.

Although GPUs were initially designed to process images and visual data, they are now being adopted to enhance other computational processes, such as scientific computing and machine learning. This is because GPUs can be effectively used in parallel for massive distributed computational processes.

The GPU Programming landscape is vast, and understanding both GPU programming concepts and Python GPU programming paradigms can be daunting. In this guide, we hope to demystify the ecosystem and lead developers to the right solution to fit their particular accelerated computing needs.



Modes of parallelism

The primary benefit of GPUs is parallelism or simultaneous processing of parts of a whole. There are four architectures used for parallel processing implementations, including:

- Single instruction, single data (SISD)
- Single instruction, multiple data (SIMD)
- Multiple instructions, single data (MISD)
- Multiple instructions, multiple data (MIMD)

Most CPUs are multi-core processors, operating with an MIMD architecture. In contrast, GPUs use a SIMD architecture. This difference makes GPUs well-suited to highly parallel processes which require the same process to be performed for numerous data items.

General purpose GPU programming

Related to GPUs' original purpose, these processors previously required users to understand specialized languages, like OpenGL. These languages were used only for GPUs, making them impractical to learn and creating a barrier to use.

In 2007, with the launch of the NVIDIA CUDA framework, this barrier was broken, providing wider access to GPU resources. CUDA is based on C and provides an API that developers can use to apply GPU processing to other tasks.

The Pros and Cons of CPU vs. GPU

When structuring your program to run on the CPU or GPU, remember that these two processors do not have the same architecture. Instructions cannot be shared between the two units, and they will not automatically shift work from one to the other. The programmer must intentionally move data and processing from one to the other either explicitly or implicitly through the use of a GPU-enabled software package.

CPUs

Central Processing Units (CPUs) are the most common type of processor used in computers today, and have long been the workhorse for data analytics tasks. CPUs consist of multiple cores that can execute instructions sequentially and in parallel, making them versatile for a wide range of applications.

CPU Strengths

CPUs have several strengths:

- Versatility: CPUs handle a range of general-purpose tasks like logical comparisons, data transfer between registers, and file management. CPUs are faster than GPUs when handling operations like data processing in RAM, I/O operations, and operating system administration.
- Multi-tasking: CPUs can run multiple tasks simultaneously.
- High clock speeds: CPUs typically have higher clock speeds than GPUs, which enables them to execute single-threaded tasks faster.
- Flexibility: CPUs can multitask between several activities, and switch between contexts quickly.
- Cache memory: CPUs have a large local cache memory.
- Compatibility: CPUs are compatible with all types of systems, while GPUs may require specialized hardware.

CPU Weaknesses

Despite their strengths, CPUs have some limitations:

- Limited parallel processing: CPUs are not designed for highly parallel workloads.
- High power consumption: CPUs can consume a lot of power, which can increase operating costs.
- Memory bandwidth limitations: CPUs have limited memory bandwidth compared to GPUs, which can result in slower performance when working with large datasets.
- Parallel processing: CPUs are less adept at tasks that require millions of identical operations.

- Limited development: CPUs are a very mature technology that has less potential for improvement, while GPUs do.

Overall, CPUs are a versatile and reliable choice for many tasks, but may not be the best option for compute-intensive, highly parallel workloads or large-scale operations with limited time and budget constraints.

GPUs

Graphics Processing Units (GPUs) were originally designed for rendering graphics and images, but in recent years have emerged as powerful accelerators for other general computing tasks. GPUs consist of thousands of processing cores that can execute instructions in parallel, making them ideal for computationally-intensive tasks.

GPU Strengths

GPUs have several strengths:

- Parallel processing: GPUs can process thousands of calculations simultaneously on many data points in parallel. GPUs have hundreds of cores that allow massively parallel calculations, like matrix multiplication.
- High memory bandwidth: GPUs can access and manipulate data more quickly.
- Energy efficiency: GPUs may consume less power than CPUs, resulting in lower operating costs.
- Customizable architectures: GPUs can be customized for specific applications and workloads, making them flexible and adaptable.
- Scientific computing use cases —GPUs provide massive acceleration for specialized tasks like generative ai, deep learning, and big data manipulation.

GPU Weaknesses

Despite their strengths, GPUs have some limitations, including:

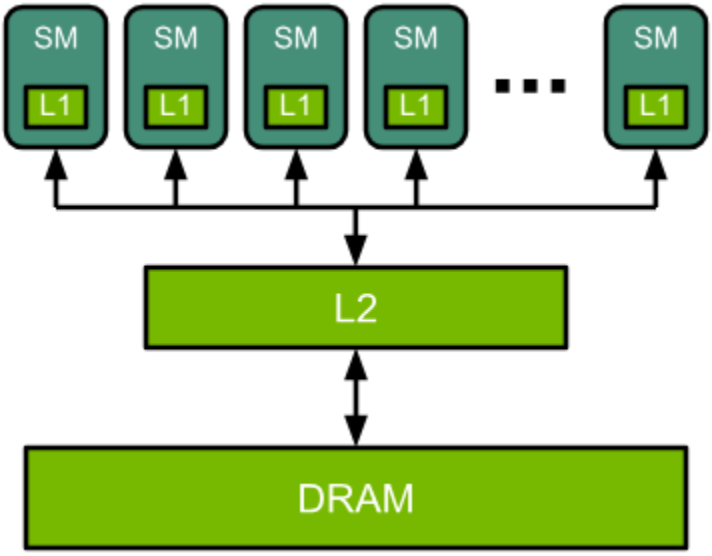
- Multi-tasking limitations: GPUs can repeatedly perform one task at a massive scale, but cannot perform general purpose computing tasks like the CPU.
- Complexity: GPUs struggle with processing tasks like branching logic, sequential operations, or other complex programming patterns.
- Software compatibility: GPUs can require additional programming or optimizations to work effectively.

Overall, GPUs are a powerful and efficient option for tasks that require high performance and parallel processing. This makes them ideal for accelerating scientific Python codes that tend to perform repetitive operations on large amounts of data.

GPU Architecture

To understand how to get the most out of your GPU, we need to understand the basics of the GPU architecture. For the sake of simplicity, we can think of a single GPU device as consisting of multiple clusters of streaming multiprocessors.

A Simplified View of GPU Architecture



A single GPU device consists of multiple Processor Clusters (PC) that contain multiple Streaming Multiprocessors (SM). Each SM accommodates a layer-1 instruction cache layer with its associated cores. Typically, one SM uses a dedicated layer-1 cache and a shared layer-2 cache before pulling data from global GDDR-5 (or GDDR-6 in newer GPU models) memory. Its architecture is tolerant of memory latency.

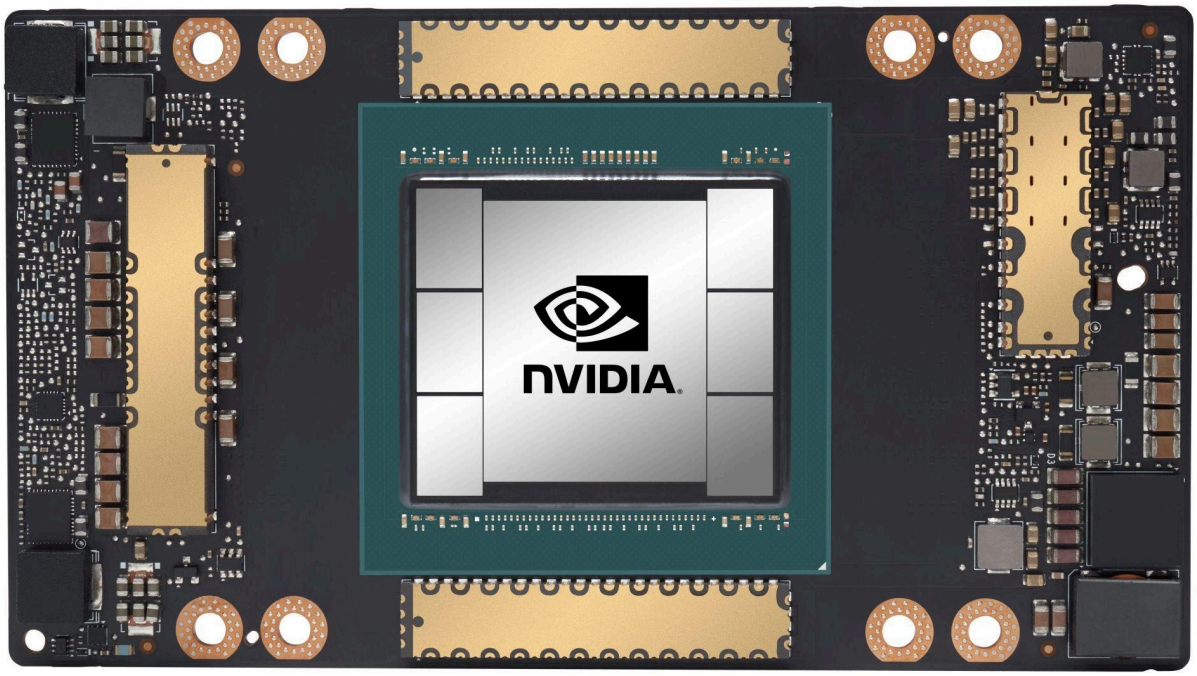
In contrast with a CPU, a GPU works with fewer, and relatively small, memory cache layers due to a larger number of transistors dedicated to computation. The GPU cares less how long it takes to retrieve data from memory as long as that memory access ‘latency’ is hidden with massively parallel computations.

A GPU is optimized for data parallel throughput computations. Looking at the numbers of cores it quickly shows you the possibilities of parallelism that it is capable of.

This is the power of GPU programming. If we are able to take advantage of the high number of cores while optimizing the data transfer, we have massively accelerated our application.

Evolution of Nvidia GPUs

Many options, you can choose from consumer-grade GPUs, data center GPUs, and managed workstations.



Consumer-Grade GPUs

Consumer GPUs are not appropriate for large-scale deep learning projects, but can offer an entry point for implementations. These GPUs enable you to supplement existing systems cheaply and can be useful for model building or low-level testing.

For the most up to date information about NVIDIA's GeForce graphics cards, go to: https://www.nvidia.com/en-us/geforce/graphics-cards/

Data Center GPUs

Data center GPUs are the standard for production deep learning implementations. These GPUs are designed for large-scale projects and can provide enterprise-grade performance.

For the most up to date information about NVIDIA's Data Center solutions, go to: https://www.nvidia.com/en-us/data-center/data-center-gpus/

DGX Servers

NVIDIA DGX servers are enterprise-grade, full-stack solutions. These systems are designed specifically for machine learning and deep learning operations. Systems are plug-n-play, and you can deploy on bare metal servers or in containers.

For the most up to date information about NVIDIA's DGX servers, go to: https://www.nvidia.com/en-us/data-center/dgx-platform/

Common GPU-Accelerated Applications

GPUs are commonly used for tasks such as:

- Scientific simulations and modeling
- Machine Learning and Neural Networks
- Deep Learning

Scientific computing

GPU's can be used in tandem with a CPU to quickly carry out numerically-intensive operations like scientific simulations and modeling. GPU design traditionally focuses on maximizing floating point units and doing multidimensional array operations. They are particularly well-suited for applications where linear math or matrix operations are useful. This is made possible due to the GPU's TensorCore unit for optimized matrix multiplication

GPU Implementations exist for many scientific computing problems including molecular modeling, fluid dynamics, astrophysics, and climate simulations.

Deep Learning

Neural networks learn from massive amounts of data in an attempt to simulate the behavior of the human brain. During the training phase, a neural network scans data for input and compares it against standard data so that it can form predictions and forecasts.

Because neural networks work primarily with massive data sets, training time can increase as the data set grows. While it's possible to train smaller-scale neural networks using CPUs, CPUs become less efficient at processing these large volumes of data, causing training time to increase as more layers and parameters are added.

Neural networks form the basis of deep learning (a neural network with three or more layers) and are designed to run in parallel, with each task running independently of the other. This makes GPUs more suitable for processing the enormous data sets and complex mathematical data used to train neural networks.

Deep Learning with Massively Parallel Inputs

Once NVIDIA introduced CUDA, several deep learning frameworks were developed, such as PyTorch and TensorFlow. These frameworks abstract the complexities of programming directly with CUDA and have made GPU processing accessible to modern deep learning implementations.

A deep learning model is a neural network with three or more layers. Deep learning models have highly flexible architectures that allow them to learn directly from raw data. Training deep learning networks with large data sets can increase their predictive accuracy.

CPUs are less efficient than GPUs for deep learning because they process tasks in order one at a time. As more data points are used for input and forecasting, it becomes more difficult for a CPU to manage all of the associated tasks.

Deep learning requires a great deal of speed and high performance and models learn more quickly when all operations are processed at once. Because they have thousands of cores, GPUs are optimized for training deep learning models and can process multiple parallel tasks up to three times faster than a CPU.

Resources

NVIDIA GPU Architectures: https://www.nvidia.com/en-us/technologies/
NVIDIA CUDA-Enabled Processors: https://developer.nvidia.com/cuda-gpus
NVIDIA Accelerated Applications Catalog:
https://www.nvidia.com/en-us/accelerated-applications/
NVIDIA GPU Cloud Applications Catalog https://catalog.ngc.nvidia.com/