# Introduction to programming general purpose GPU accelerated applications utilizing CUDA

## Table of Contents



# 1. Introduction

## 1.1. Why?

The architecture of GPUs allow for a very different computational workload compared to CPUs. CPUs were originally are modeled to consist of one core which computes things in a linear fashion… TODO. While nowadays, due to a slow down in single core performance, has lead to many CPU manufacturers to release CPUs with multiple cores, the number of these cores are still not in any way comparable to those of GPUs. TODO

TODO

GPUs are utilized in a variety of well known workloads, among others:

- Machine Learning (Neural Networks)
- Rendering Engines & Image Processing
- Simulation Software (Fluid, Medical, Financial, Weather, etc.)
- Crypto Mining (Computation of Cryptographic Hashes)

TODO Basically any computation which can be represented well in the format of matrices and matrix-computations can usually be implemented efficiently utilizing GPU hardware.

## 1.2. GPU architecture

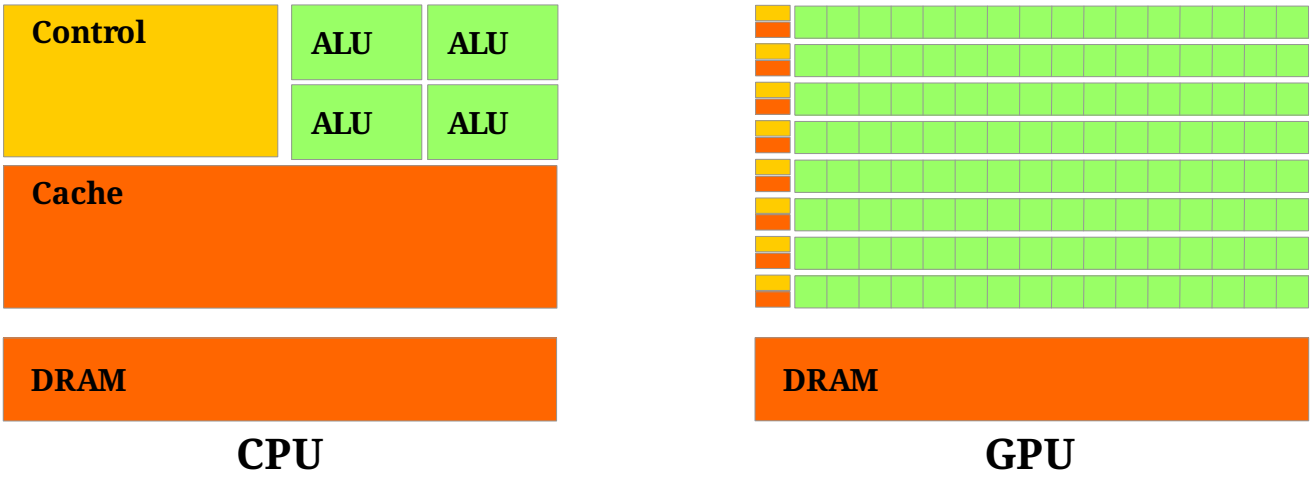


*Figure 1. Simplified comparison of CPU vs GPU architecture, NVIDIA, CC BY 3.0 https://creativecommons.org/licenses/by/3.0, via Wikimedia Commons*

CPUs typically feature a small number of powerful cores, designed for versatile, single-threaded performance. They excel in handling tasks with intricate branching and diverse instruction sets. In contrast, GPUs comprise a multitude of smaller and simpler cores, emphasizing parallel processing. Their architecture, with shallow instruction pipelines, enables the execution of the same instructions across multiple data points simultaneously, called Single Instruction Multiple Data (SIMD)[1], making GPUs highly efficient for data-parallel workloads.

Memory hierarchy is another differentiating factor. CPUs incorporate a complex hierarchy with various levels of cache to minimize memory access latency. This design suits low-latency tasks and diverse data types. GPUs, on the other hand, prioritize high memory bandwidth over complex hierarchies, which makes them ideal for data-intensive parallel processing but less effective for latency-sensitive tasks.

## 1.3. Prerequisites

To run the executables provided in the repository you will need a device which has an Nvidia GPU and also somewhat recent Nvidia drivers installed.

If you want to run the provided code directly or modify it you will also need to install the Nvidia CUDA Developer Toolkit Version 12.2, found on the Nvidia website. The toolkit also seamlessly integrates into Microsoft Visual Studio 2022/2019 so using this IDE is recommended.

# 2. CUDA "Hello World" [1]

For this example we will be looking at the simple task of vector addition, where every nth element of vector a and b are added together to create vector c. The c++ code would look something like this:

```
void vecAdd(float* h_A, float* h_B, float* h_C, int n)
{
    for (int i = 0; i < n; i++) h_C[i] = h_A[i] + h_B[i];
}
```

[1] The idea for the below example and the code comes from the book "Programming Massively Parallel Processors". This was done to demonstrate a maximally simple program as a first introduction to CUDA.

[1] Cuda utilizes Single Program Multiple Data (SPMD), the main difference being that not the exact same instruction has to be executed in each thread, but the same program, this allows for branching and more.