

# AI ON SPEED: GPU REVOLUTION

...

# INTRODUCTION

Typically, a computer has one or more processors. These processors perform all the calculations needed to run a program. Traditionally, these processors, known as CPUs (Central Processing Units), manage various tasks, from simple calculations to complex data processing. However, with the rising demand for computational power—especially in fields like artificial intelligence and deep learning—CPUs alone are no longer enough. This is where GPUs (Graphics Processing Units) come into play. Originally designed for rendering graphics, GPUs excel at handling many tasks simultaneously. This parallel processing capability makes them ideal for training deep learning models and processing large datasets.

In this presentation, we'll explore the CUDA ecosystem, including key tools like cuDNN and TensorRT, and how they enhance performance in AI applications. We will also discuss NVIDIA DGX systems, which are optimized for high-performance computing and AI workloads.

# CONTENTS

I – From CPUs to GPUs : Evolution & Architectures

II – Performance and Real-world Application of GPUs in AI

III – CUDA: The Key to High-Performance

IV – CUDA in Action : Ecosystem and Tools

V – Exploring Alternatives : OpenCL and others

VI – Challenges and Future of GPU Computing in AI

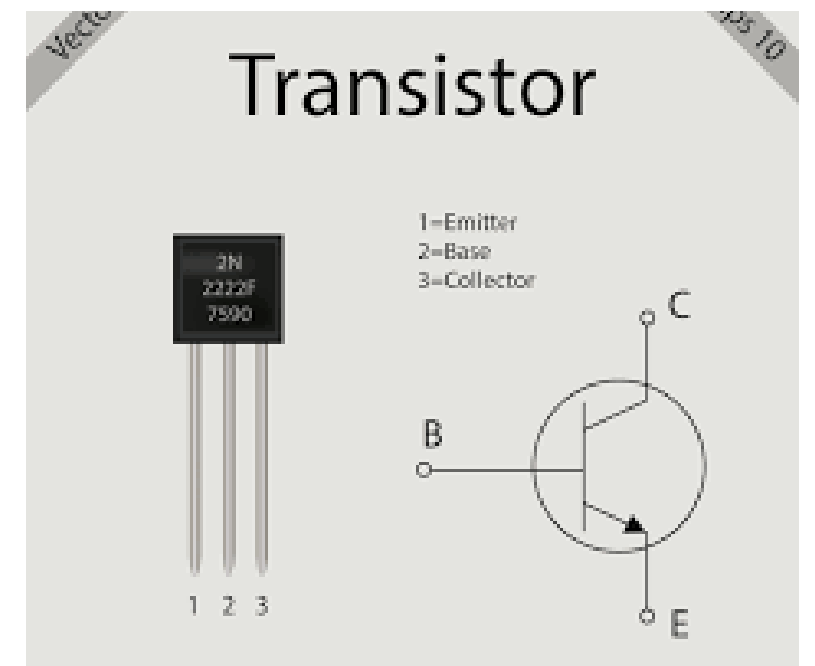


# Definitions

**Vacuum tubes** also known as electron tubes or thermionic valves, are devices that control electric current flow in a high vacuum between electrodes to which an electric potential difference has been applied



**Transistor** is a semiconductor electronic component used to control or amplify electrical voltages and currents.



# I - From CPUs to GPUs

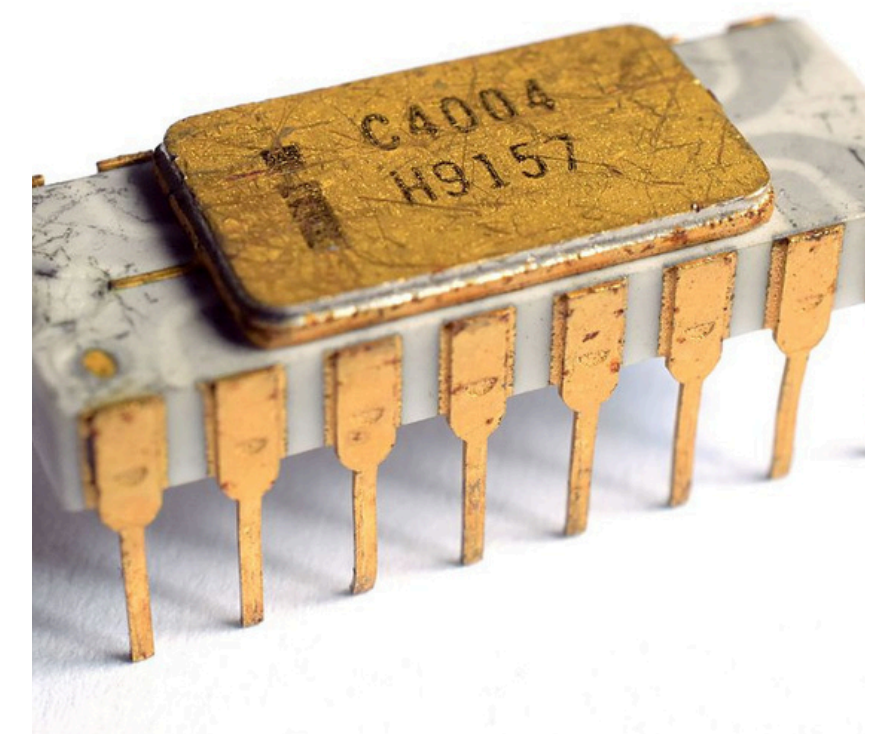


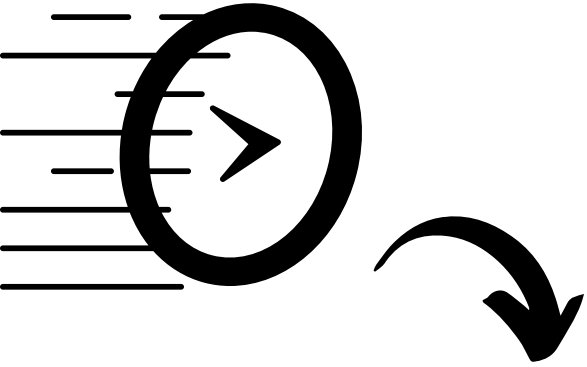


**ENIAC**

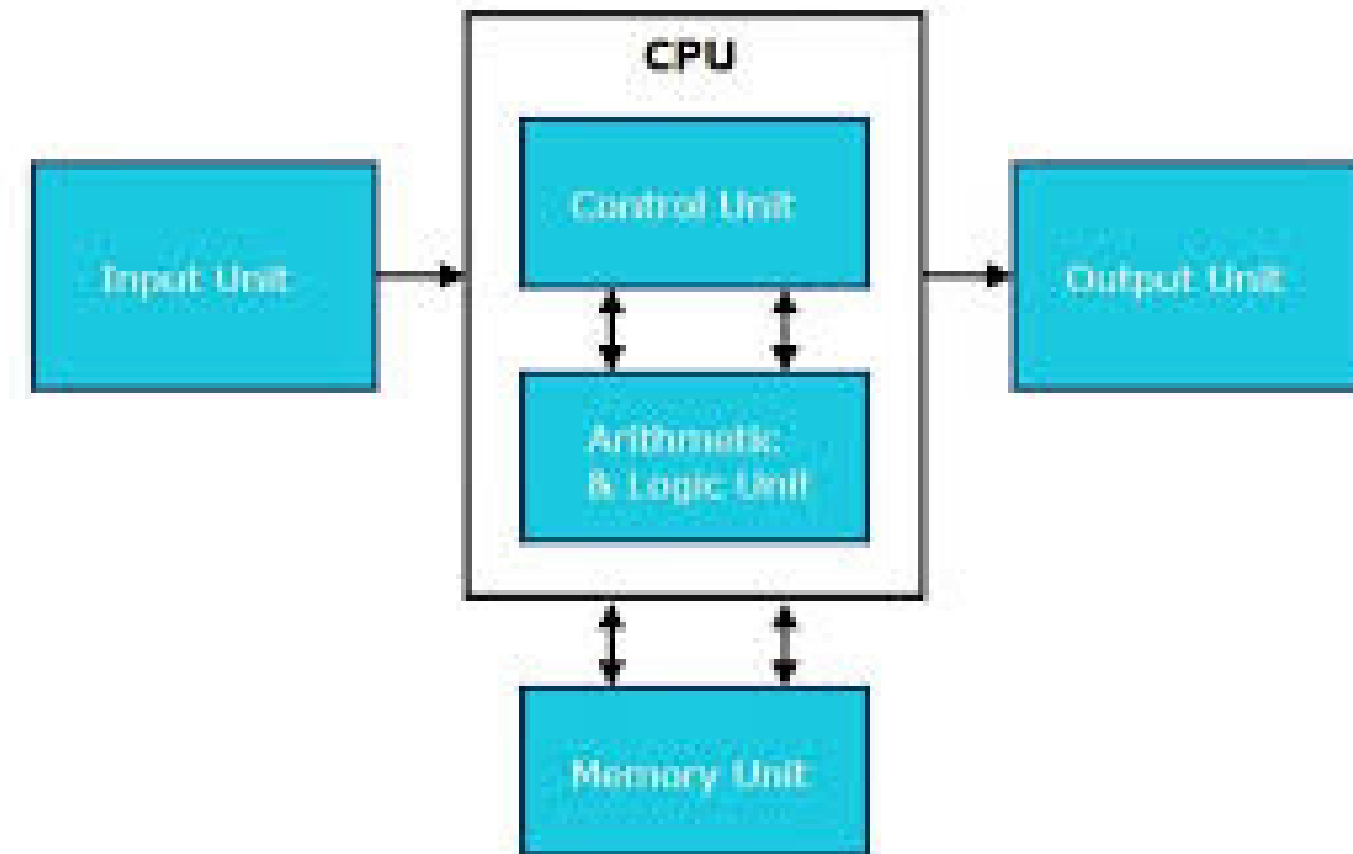
The history of processors begins in the 1940s, with the first electronic computers like the ENIAC, which used vacuum tubes to perform calculations. The ENIAC (Electronic Numerical Integrator and Computer), created between 1943 and 1945 by John Presper Eckert and John Mauchly at the University of Pennsylvania, was the first general-purpose digital electronic computer. Designed for the U.S. Army to calculate ballistic trajectories during World War II, it used about 17,468 vacuum tubes and operated in binary. The ENIAC, unveiled in 1946, could perform complex mathematical operations much faster than mechanical machines of the time, although its manual programming and massive size (27 tons) made it impractical compared to modern computers.

The machines were bulky, not very efficient, and consumed a lot of energy. The invention of the transistor in 1947 by William Shockley and his colleagues was a major turning point. Transistors replaced vacuum tubes and made circuits smaller. In the 1960s, integrated circuits appeared, combining multiple transistors on a single silicon chip. Then, in 1971, Intel released the 4004, the first commercial microprocessor. This 4-bit processor integrated all the components of a central processor onto a single chip, a revolution in the industry.





Central Processing Unit - CPU, is the brain of the computer. It executes program instructions and handles basic operations such as calculations, logic, and data management. Its main role is to take data, process it, and return a result or action. Every application or task performed on a computer goes through the processor.



It is composed of several key elements: the control unit, which directs the execution of instructions, the ALU (Arithmetic & Logic Unit) which performs mathematical and logical operations, and the registers, which serve as temporary memory to store data. The CPU operates in clock cycles, retrieving instructions from memory (fetch cycle), decrypting them (decode cycle), and then executing them (execute cycle). The performance of the CPU depends on several factors, including the clock frequency and the number of cores, allowing faster and parallel execution of tasks.



**THE GPU**



In the 1980s, with the explosion of arcade games and the arrival of the first personal computers, the video game and computer industry began to realize the limitations of traditional CPUs, such as those from Intel and IBM, in handling real-time graphics. The games of this era, while revolutionary, were often constrained by the limited computing power of central processors, which had to handle both game logic and graphics rendering. Companies like Intel and IBM dominated the chip market at the time, but these CPUs were not optimized for the massively parallel graphics tasks demanded by video games and emerging 3D software.



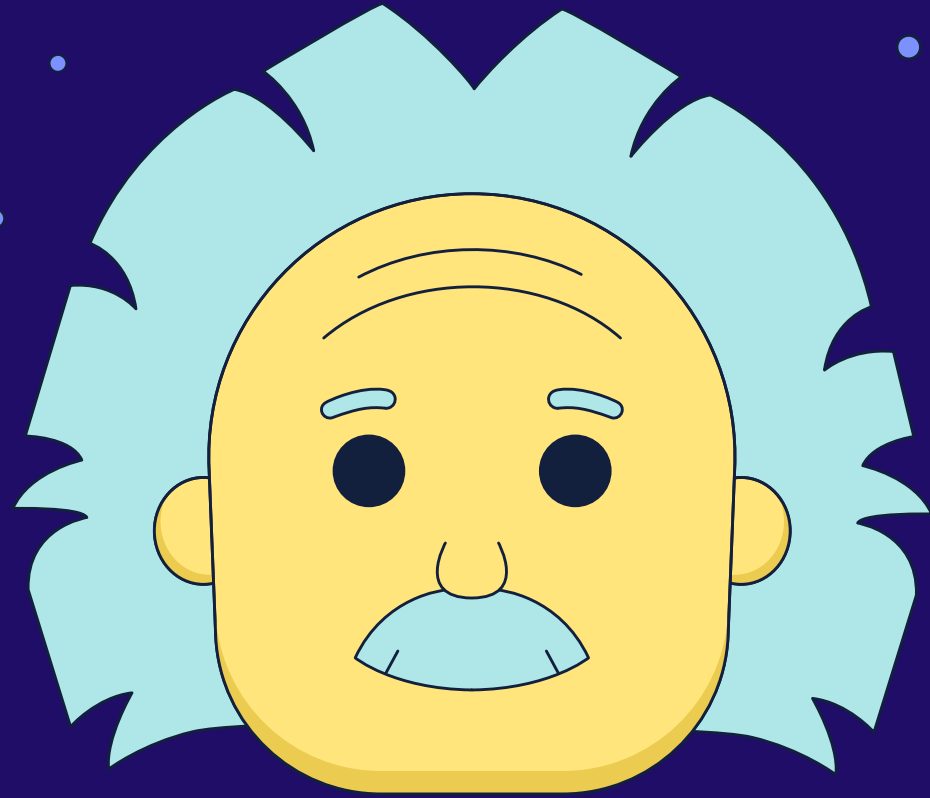
It was in this context that the race for graphics performance accelerated. Intel had integrated basic graphics capabilities into its processors, but this was no longer sufficient to meet the growing demands of game and graphic design software developers. The need for a specialized processor became apparent, and some companies like 3dfx Interactive attempted to meet this demand with dedicated graphics cards. However, it was NVIDIA, founded in 1993, that would truly revolutionize the industry with the introduction of the GeForce 256 in 1999.



Unlike previous graphics cards that were primarily used to accelerate image rendering, the GeForce 256 introduced a new architecture capable of handling complex operations directly on the card, without relying on the CPU.

- Transform & Lighting (T&L) Engine
- 32 MB DDR Memory
- 125 million triangles per second: 3D geometry processing capability.
- Multitexturing: Apply multiple textures in a single pass.
- 480 million pixels per second Fill Rate: Pixel processing speed for fast, smooth rendering.

## These GPUs introduced the foundations of modern GPUs

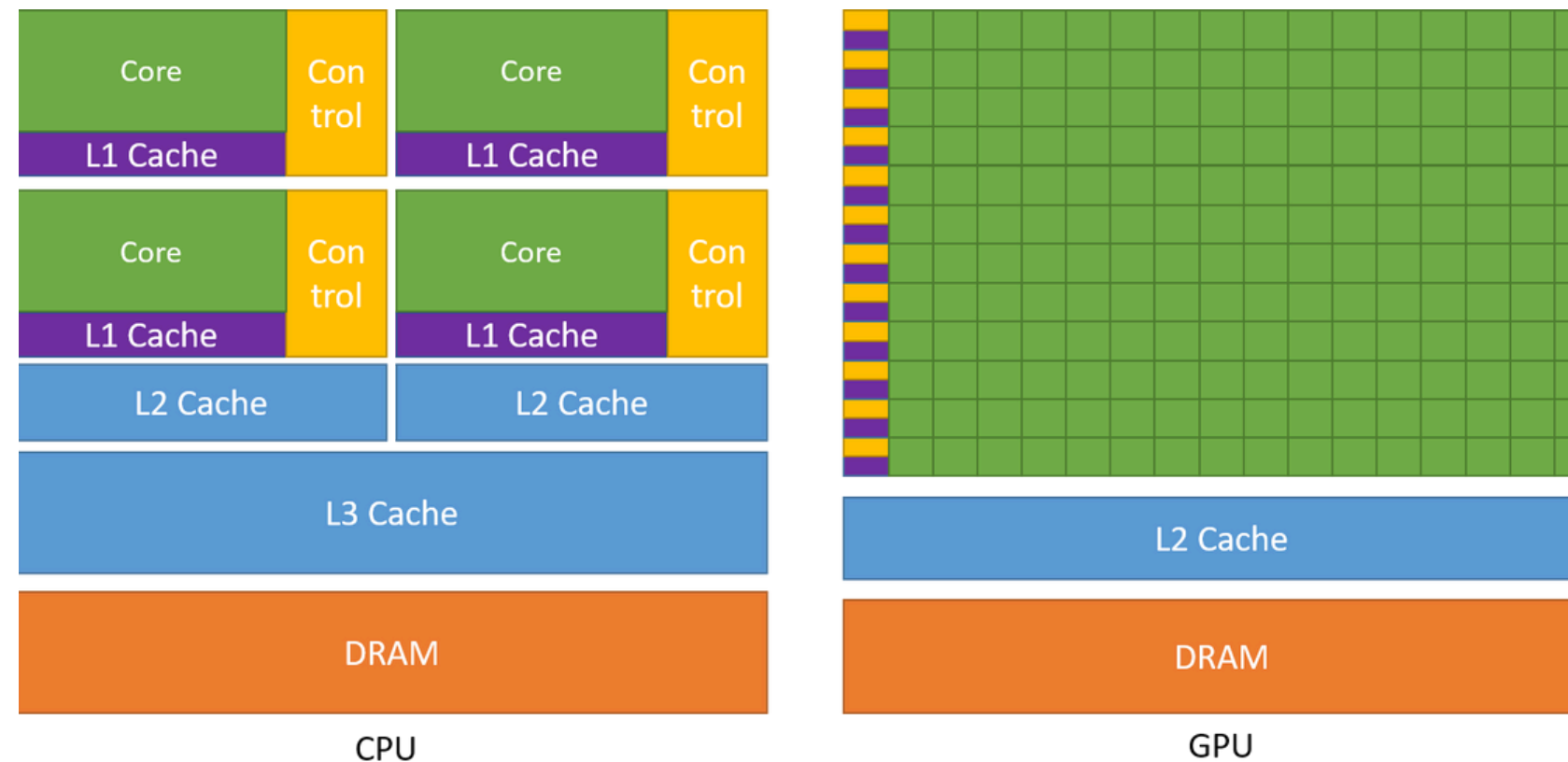


**Let's look CPU as 10 to 20 Einstein**

**And GPU as 10000 me**



# CPU VS GPU OVERVIEW



## CPU

- few cores optimized for serial processing
- lower memory bandwidth (but direct access to more memory)
- latency optimized cores (faster at a single task but can only perform few at the same time)
- more instructions but slower execution

## GPU

- hundreds/thousands of smaller, more efficient cores optimized for multiple tasks simultaneously
- great for compute-intensive parts of the application
- higher memory bandwidth (but direct access to less memory than CPU)
- throughput optimized cores (slower at single tasks but can perform more at the same time)
- limited number of instructions available but faster execution

**Note :** In a GPU accelerated application the compute-intensive parts are offloaded to the GPU, while the remaining of the application runs on the CPU.

# GPU in AI

- **Parallel Processing for Massive Data:**
  - In AI, especially deep learning, large datasets and complex models require high computational power. GPUs are designed with thousands of cores, enabling them to perform many calculations in parallel. This is crucial for tasks like matrix multiplication in neural networks, which are at the heart of AI model training.
  - Example: Training a deep neural network (DNN) involves calculating millions of weight updates across multiple layers. A GPU can compute these updates simultaneously, significantly speeding up the process compared to a CPU.
- **Faster Model Training:**
  - Training an AI model on a large dataset using only a CPU can take days or even weeks. With a GPU, the same task might be completed in hours. This reduces the time-to-iteration, allowing researchers and developers to experiment more rapidly with different architectures or hyperparameters.
  - GPUs are essential for keeping up with the demands of deep learning frameworks like TensorFlow or PyTorch, which benefit from their ability to handle large-scale matrix operations.
- **Support for Deep Learning Libraries:**
  - Many AI libraries, such as TensorFlow, PyTorch, and Keras, are optimized for GPUs. These frameworks have built-in support for leveraging the computational power of GPUs, making it easier for AI practitioners to train complex models.
- **Efficient Use of Memory:**
  - GPUs have dedicated, high-bandwidth memory (VRAM), which is crucial for handling the large datasets used in AI. This allows them to load large chunks of data and perform computations faster, reducing bottlenecks in AI workflows.



- **AI Model Inference:**
  - Beyond training, GPUs are also beneficial for inference—the process of using a trained AI model to make predictions on new data. A GPU can process these predictions much faster, enabling real-time applications such as autonomous driving, object detection in video streams, or voice assistants.
  - Example: Self-driving cars require the real-time processing of images, which is only possible because GPUs can handle massive amounts of data simultaneously.
- **Scalability and High-Performance Computing:**
  - In cloud environments, multiple GPUs can be used in tandem to scale AI workloads across clusters, further enhancing speed and efficiency.
  - GPUs are often employed in distributed computing, where large-scale AI models are trained across several machines, each equipped with multiple GPUs.
- **Energy Efficiency:**
  - Compared to CPUs, GPUs are generally more power-efficient for the types of parallel processing tasks common in AI. This makes them not only faster but also more cost-effective, particularly in environments where large-scale model training is required.
  - Energy efficiency is a key consideration for cloud providers and data centers offering AI as a service, where GPUs are essential for maintaining high performance at manageable costs.
- **Innovation in Specialized AI Hardware:**
  - The success of GPUs in AI has inspired the development of more specialized AI hardware, such as TPUs (Tensor Processing Units) by Google, but GPUs remain the most widely accessible and versatile hardware for both research and production environments.

# **II - Performance and Real-world Application of GPUs in AI**



Over the past 30 years, GPUs have accelerated research, contributing to many causes in every major field. Let's take a look :

- <https://www.nvidia.com/en-us/drivers/social-impact-gpu/>
- <https://www.nvidia.com/en-us/accelerated-applications/?filter=eyJ3b3JrbG9hZHMlOlsiR2VuZXJhdGl2ZSBBSAAvIEExMTXMiXSwiQWNjZWxlcmlmF0aW9uX1R5cGUlOlsiR1BVIEFjY2VsZXJhdGVkl19>
- <https://drive.google.com/drive/u/2/folders/1Vy6Qzz1UhabMH4DyxN07QdfY8cVSt6sr>






# III - CUDA : The Key to High-Performance



In November 2006, NVIDIA introduced CUDA, a general purpose parallel computing platform and programming model that leverages the parallel compute engine in NVIDIA GPUs to solve many complex computational problems in a more efficient way than on a CPU.

CUDA comes with a software environment that allows developers to use C++ as a high-level programming language. Other languages, application programming interfaces, or directives-based approaches are supported, such as FORTRAN, DirectCompute, OpenACC.



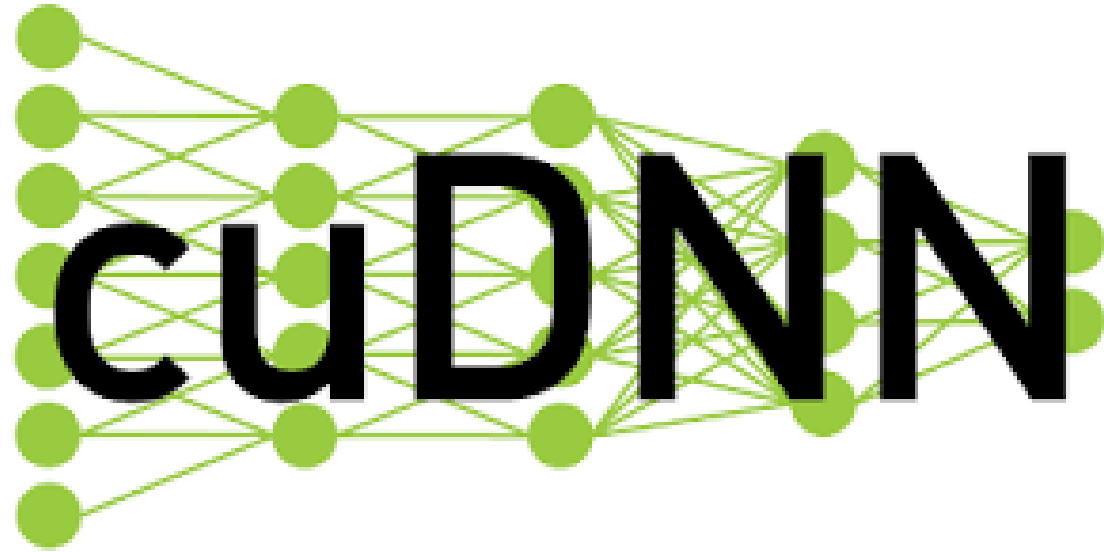
GPU Computing Applications						
Libraries and Middleware						
cuDNN TensorRT	cuFFT cuBLAS cuRAND cuSPARSE	CULA MAGMA	Thrust NPP	VSIPL SVM OpenCurrent	PhysX OptiX iRay	MATLAB Mathematica
Programming Languages						
C	C++	Fortran	Java Python Wrappers	DirectCompute	Directives (e.g. OpenACC)	
<div><h3>CUDA-Enabled NVIDIA GPUs</h3></div>						
NVIDIA Ampere Architecture (compute capabilities 8.x)					Tesla A Series	
NVIDIA Turing Architecture (compute capabilities 7.x)		GeForce 2000 Series	Quadro RTX Series		Tesla T Series	
NVIDIA Volta Architecture (compute capabilities 7.x)	DRIVE/JETSON AGX Xavier		Quadro GV Series		Tesla V Series	
NVIDIA Pascal Architecture (compute capabilities 6.x)	Tegra X2	GeForce 1000 Series	Quadro P Series		Tesla P Series	
<div><div><p>Embedded</p></div><div><p>Consumer Desktop/Laptop</p></div><div><p>Professional Workstation</p></div><div><p>Data Center</p></div></div>						



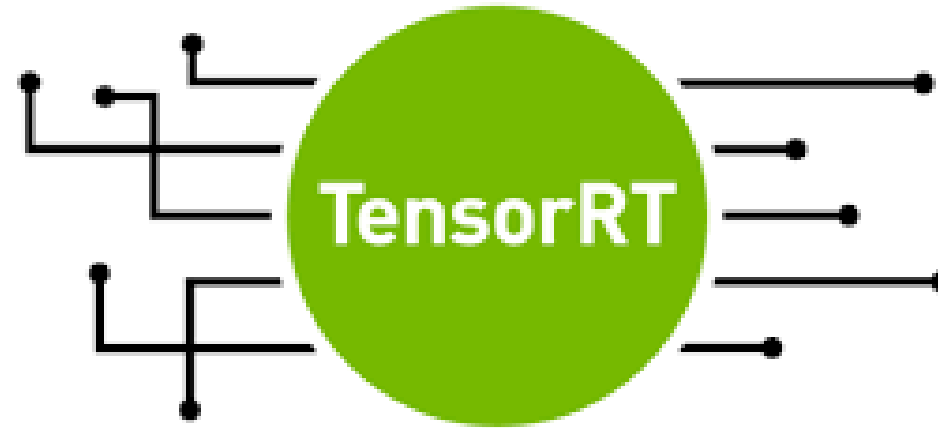
- Setup CUDA environment
  - Local environment
    - Linux
    - Windows
    - Other
  - Google Colab
- Practical examples

# IV - CUDA in Action : Ecosystem and Tools

NVIDIA owns several tools that allows developers to build, run, and optimize software that can leverage NVIDIA GPUs for parallel computing.



Deep learning models involve heavy matrix computations. cuDNN optimizes these operations to run faster on NVIDIA GPUs, ensuring that even large models can be trained efficiently. By utilizing cuDNN, AI practitioners can significantly reduce training times, often by several orders of magnitude, compared to non-optimized implementations. It also ensures seamless integration with popular deep learning frameworks like TensorFlow and PyTorch.



Once a deep learning model is trained, it needs to make predictions in real-time for tasks like autonomous driving, AI-based diagnostics, or natural language processing (NLP) in chatbots. TensorRT fine-tunes models to execute faster and more efficiently on GPUs. TensorRT optimizes neural networks by using techniques like precision calibration (e.g., converting 32-bit floats to 16-bit or 8-bit), kernel auto-tuning, and layer fusion. These optimizations help maximize throughput and reduce latency.



Writing efficient CUDA code requires a deep understanding of how the GPU operates. Nsight tools provide detailed insights into how well the GPU is being utilized, allowing developers to make necessary adjustments to enhance performance. Nsight helps in identifying inefficient memory access patterns, underutilized cores, and long kernel execution times. By addressing these issues, developers can optimize their CUDA applications, ensuring maximum performance from the GPU.



The Nvidia DGX represents a series of servers and workstations designed by Nvidia, primarily geared towards enhancing deep learning applications through the use of general-purpose computing on graphics processing units (GPGPU). These systems typically come in a rackmount format featuring high-performance x86 server CPUs on the motherboard.

The core feature of a DGX system is its inclusion of 4 to 8 Nvidia Tesla GPU modules, which are housed on an independent system board. These GPUs can be connected either via a version of the SXM socket or a PCIe x16 slot, facilitating flexible integration within the system architecture. To manage the substantial thermal output, DGX units are equipped with heatsinks and fans designed to maintain optimal operating temperatures.

This framework makes DGX units suitable for computational tasks associated with artificial intelligence and machine learning models.

# V- Exploring Alternatives : OpenCL and others



While CUDA has become a standard for GPU programming, especially on NVIDIA hardware, there are several alternatives that can work on different hardware or provide different programming models.

Here are some:



OpenCL is an open standard maintained by the Khronos Group, and it supports cross-platform parallel programming. It is designed to work across different types of hardware, including CPUs, GPUs, and FPGAs from different vendors (like AMD, Intel, and NVIDIA)



Vulkan is a low-level, cross-platform API for graphics and compute tasks, succeeding OpenGL. It offers fine control over GPU resources for high performance but is more complex to program than higher-level APIs like Direct3D or OpenGL.

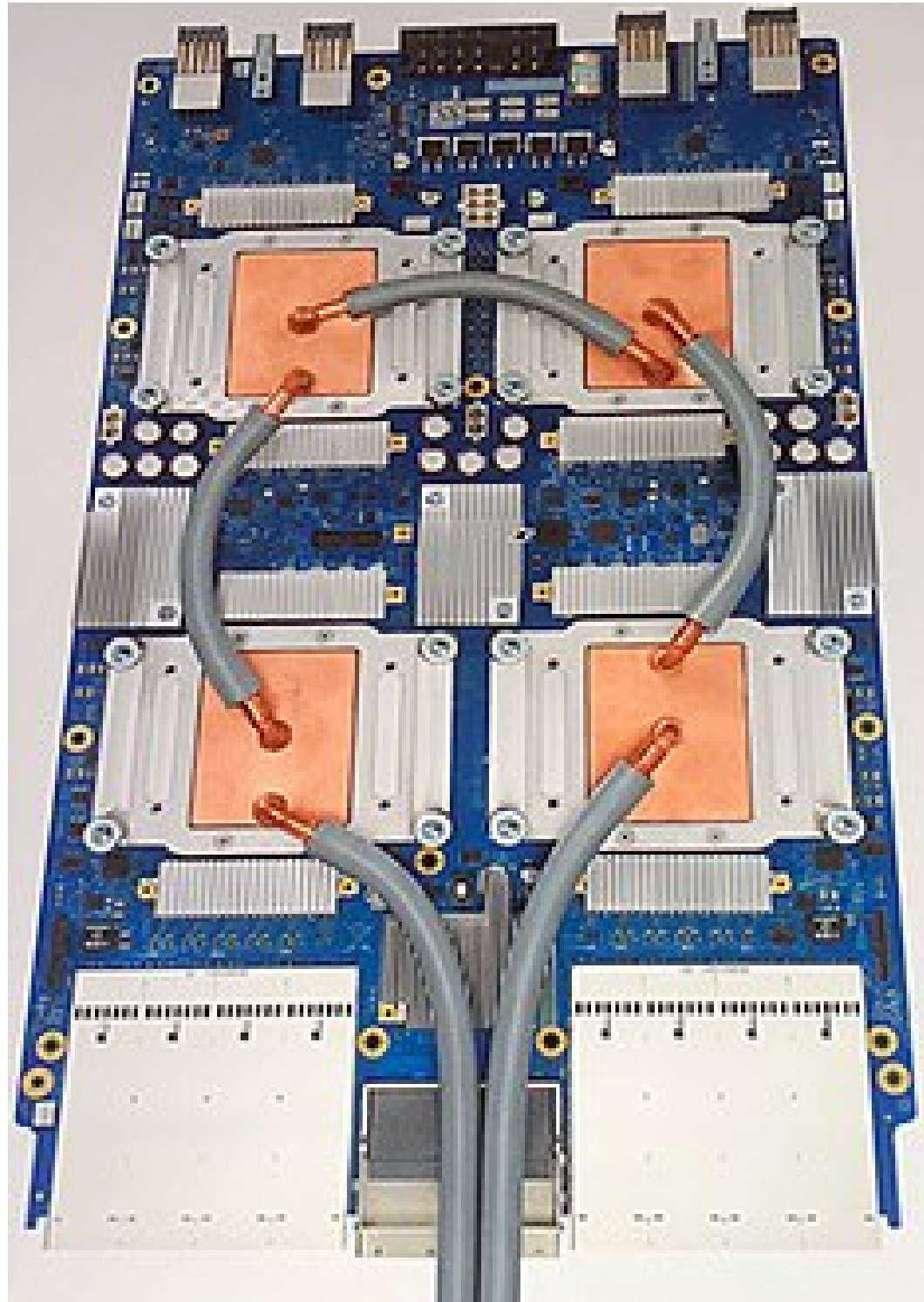


Metal is Apple's low-level API for graphics and compute, optimized for macOS and iOS devices. It provides direct GPU access for high-performance tasks like gaming and machine learning, but is exclusive to Apple's ecosystem.

# **VI- Challenges and Future of GPU Computing in AI**

# LIMITS OF CUDA

- **CUDA exclusively compatible with NVIDIA GPUs:** Indeed, this constitutes a limit since applications developed with CUDA are inefficient/unusable on GPUs from other manufacturers (AMD or Intel). In addition, the cost of GPUs also constitutes a barrier to this technology.
- **Complexity of programming in CUDA:** You need in-depth knowledge of the concepts of parallel computing, GPU architecture as well as mastery of a syntax which can be very complex for developers.
- **Restriction of CUDA to C/C++, Fortran:** Although third-party libraries (e.g. PyCuda) try to make this technology versatile in terms of programming language, support for other languages may be limited (Resurrection of primitive languages among much more sophisticated languages).
- **Far too much power consumption:** NVIDIA GPUs used with CUDA can indeed consume a lot of power. As an illustration, a data center using 100 NVIDIA A100 GPUs could consume 30,000W exclusively for the GPUs (at a Max Thermal Design Power (TDP) of 300W for this model. NVIDIA is increasingly perfecting its GPUs, the new models are excessively energy-intensive. For NVIDIA, "We had to be cunning to increase power without exploding consumption, starting with the integration of a variable frequency of the GPU. This system makes it possible to vary the frequency so as to leave the GPU at an appropriate temperature or to limit the power consumption of the graphics card.
- **GPU are too expensive**

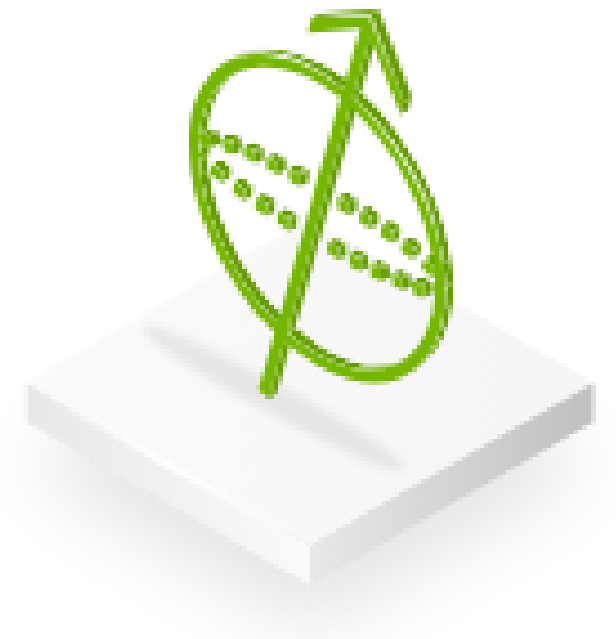


A Tensor Processing Unit (TPU) is an application-specific integrated circuit (ASIC) developed by Google specifically to accelerate neural network-based artificial intelligence systems. TPUs were designed to accelerate deep learning workloads built in TensorFlow. Unlike GPUs, a TPU implements only the bare minimum needed to perform the desired matrix operations. Similarly, TPUs rely on an adjunct CPU to do most of their work. This approach allows TPUs to achieve higher speeds than GPUs at lower power costs. NVIDIA is taking note of TPUs, and it's likely that future versions of their GPUs will resemble TPUs.

# CUDA & QUANTUM COMPUTING

NVIDIA cuQuantum is an SDK of optimized libraries and tools for accelerating quantum computing workflows. With NVIDIA Tensor Core GPUs, developers can use cuQuantum to accelerate quantum circuit simulations based on state vector and tensor network methods by orders of magnitude. (see = “there”)

cuQuantum integrates features such as simulation optimization or even support for hybrid calculations. CUDA is therefore useful on these functional points by making it possible to simulate larger and more complex quantum circuits in less time thanks to massive parallelization of GPUs (acceleration of quantum simulations) and facilitate the implementation of complex algorithms (the classic steps are processed by the GPU and the quantum steps are processed by a quantum computer/simulator).



## Key benefits of cuQuantum:

- Flexible: Choose the best approach for your work, from multiple proven methods.
- Scalable: Leverage the power of multinode, multi-GPU clusters, on-premise or cloud.
- Fast: Simulate bigger problems faster, and get more work done sooner.



## Hybrid Applications

Pharma, Chemistry, Weather, Finance, Logistics, and More

## CUDA Quantum

The Platform for Hybrid Quantum-Classical Computing

## System-Level Toolchain

NVQ++

NVIDIA GPUs

Quantum Resource

NVIDIA cuQuantum or Partner QPU

**NVIDIA Architecture for Hybrid Quantum-Classical Computing**

# CONCLUSION

In conclusion, **CUDA** is a powerful development environment and API designed by NVIDIA to harness GPU parallelism for computationally intensive applications. It accelerates complex tasks such as machine learning, image processing, and scientific simulations. GPUs, with their ability to process data on a massively parallel scale, play a crucial role in machine learning. While it's possible to become proficient in machine learning without writing any GPU code, building a deep intuition about the underlying processes becomes challenging when relying solely on high-level abstractions. CUDA enables developers to write efficient, high-performance kernels that fully leverage the power of modern hardware, especially as models grow in complexity.

However, while CUDA offers high performance, it is limited to NVIDIA GPUs, which can restrict code portability across platforms. This limitation requires developers to carefully manage hardware resources, and as demonstrated in the "Hello World" example, adjustments may be necessary to handle larger data sizes efficiently. Despite these challenges, CUDA remains a leading choice for developers aiming to optimize performance on NVIDIA GPU platforms and gain a deeper understanding of how to maximize computational power at a low level.

**Thanks for attention...!!!**

# RESSOURCES

- To understand more about CPU, please take a look at this [video](#)
- [Here is a demo of CPU vs GPU by Mythbusters](#)

Then, some links that will help you to master in depth subjects we taks about:

- <https://docs.nvidia.com/>
- <https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html>
- <https://gfxcourses.stanford.edu/cs149/fall24>
- [https://edoras.sdsu.edu/~mthomas/docs/cuda/cuda\\_by\\_example.book.pdf](https://edoras.sdsu.edu/~mthomas/docs/cuda/cuda_by_example.book.pdf)
- <https://youtube.com/playlist?list=PL5XwKDZZlwaY7t0M5OLprpkJUlrF8Lc9j&si=6mK3xdnxydPryiAE>
- [https://youtu.be/86FAWCzle\\_4?si=VA10ODALNbZfd1QY](https://youtu.be/86FAWCzle_4?si=VA10ODALNbZfd1QY)