PARALLEL PROGRAMMING...

Parallel Programming: Overview

GOAL

Sequential Parallel Programming

Hardware Architecture

Architecture of a CPU versus GPU

AMD ROCm and CUDA Platforms

GPGPU (General Purpose computation on Graphics Processing Units)

Programming Interface for parallel computing
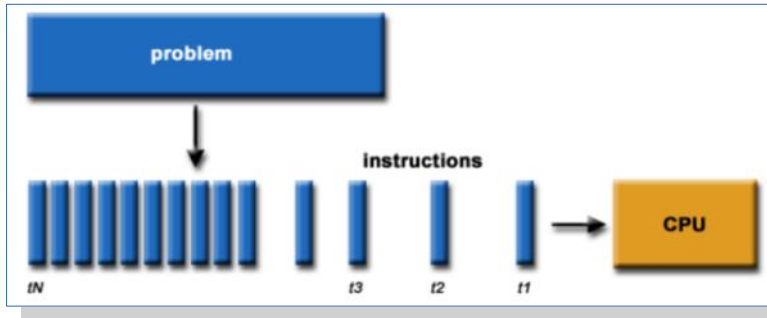
MPI (Message Passing Interface)

OpenMP (Open Multi-Processing)

Sequential / Parallel Programming

Traditional **C**omputing:

- Programs are broken into series instruction.
- Executed by a single processor sequentially.
- No coordination required

Parallel programming is the use of several calculation :

- Programs are broken co-current subprograms
- Executed on multiple processors in parallel
- Each party is still discovered in instructions
- Coordination required between processors
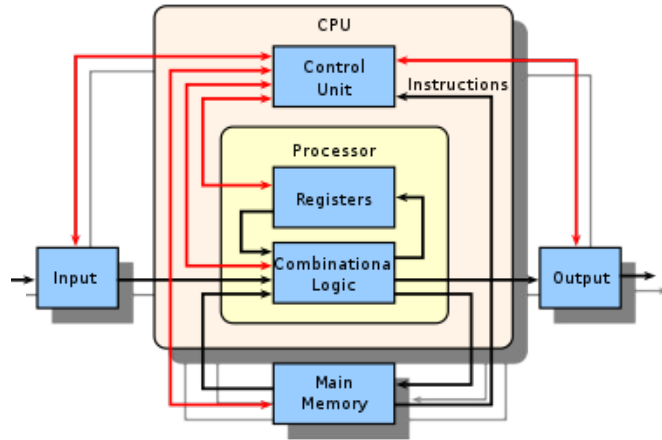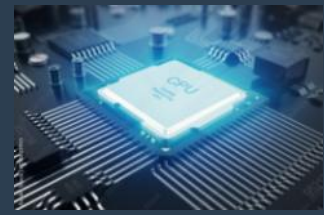- Instructions of each party are executed in parallel
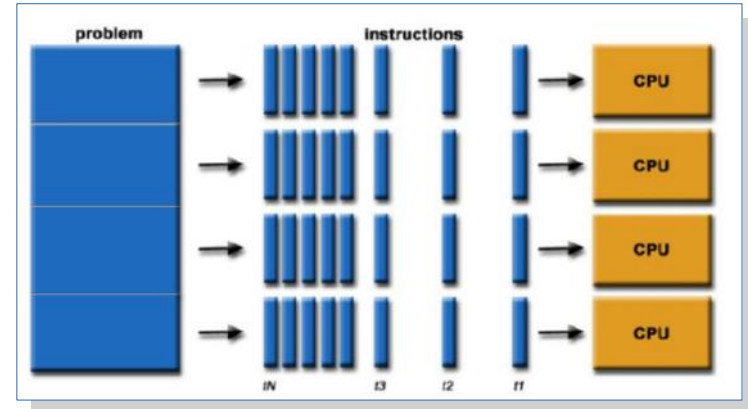
Hardware **A**rchitecture
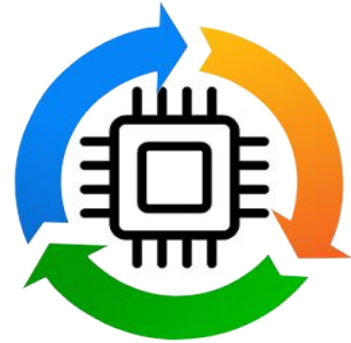
CPU (Central Processing Unit)

A **CPU** (**C**entral **P**rocessing **U**nit) is the most important processor in a given computer. It executes instructions of a computer program, such as *arithmetic*, *logic*, *controlling*, and *input/output (I/O)* operations.

It constitutes the **physical heart** of the entire **computer system.**

Linked with various peripheral equipment, including input/output devices and auxiliary storage units.

CPU: Different architectures of the processor

CPU Architecture

There is a classification of the **different CPU architectures.**

Five in number, they are used by programmers depending on the desired results:
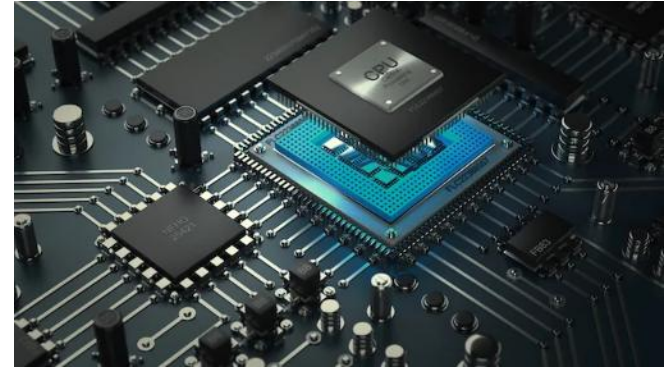
CISC: very complex addressing.

RISC: simpler addressing and instructions performed on a single cycle.

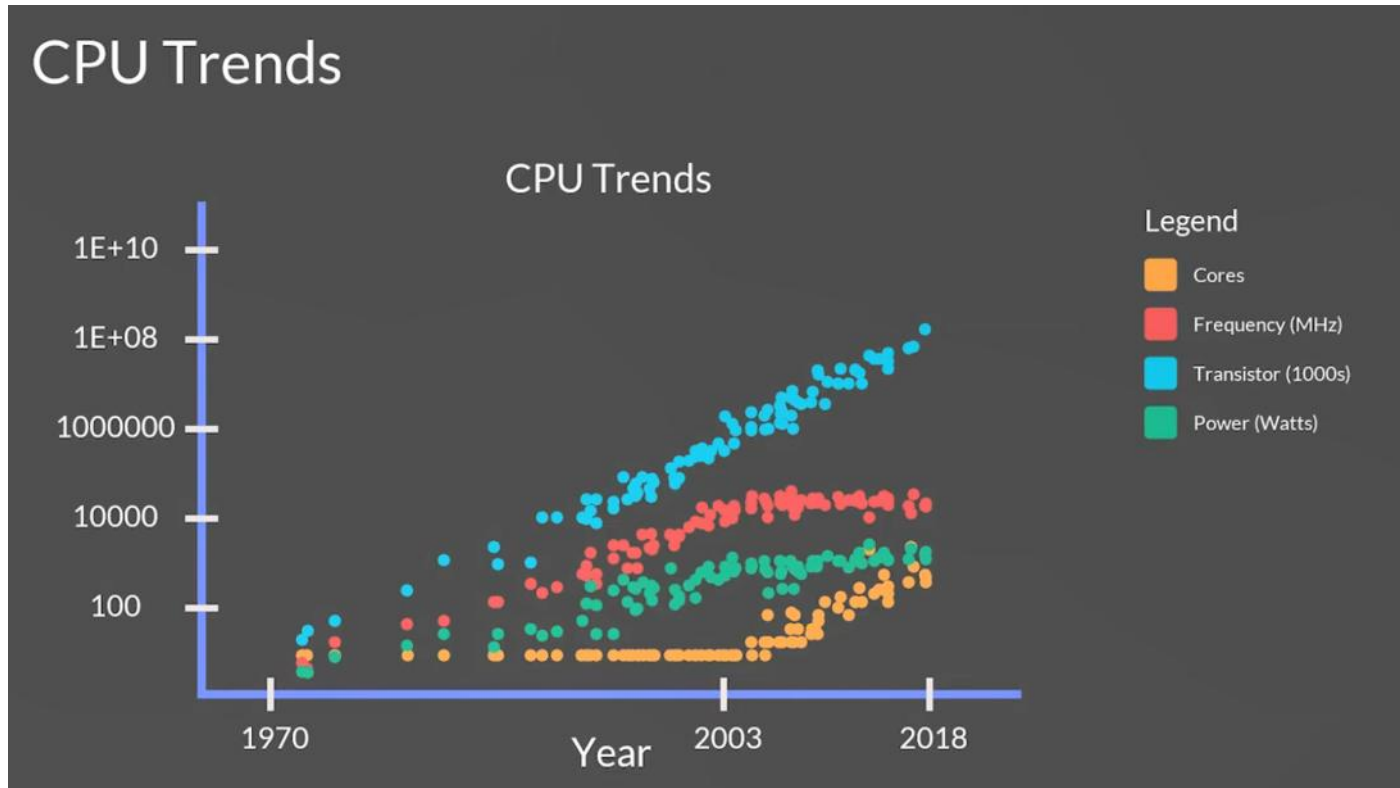**VLIW:** long, but simpler instructions.

vectorial: contrary to the processing in number, the instructions are vectorial.

dataflow: data is active unlike other architectures.
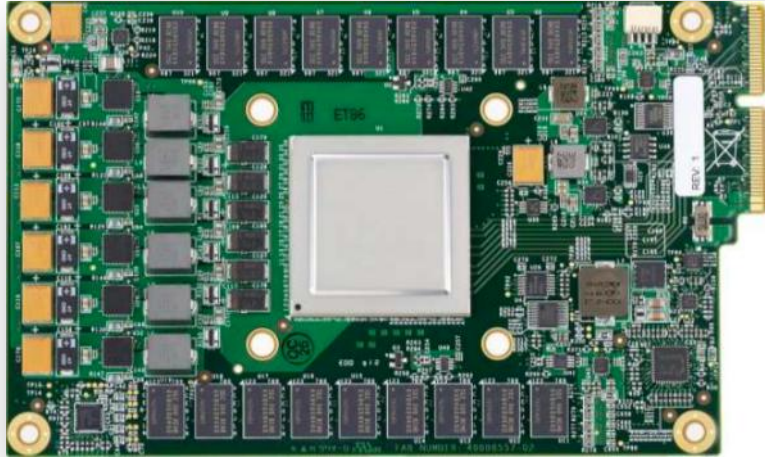
CPU (Central Processing Unit): Trends
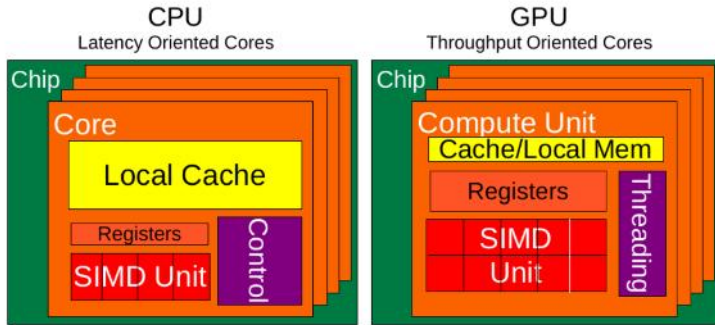
GPU (Graphics Processing Unit)

A **GPU** is uses to **speed up the process** of creating and rendering computer graphics, designed to accelerate graphics and image processing.

It is the most important hardware.

But have later been used for ***non-graphic calculations*** involving embarrassingly parallel problems due to their parallel structure.

CPU vs GPU

CPU is composed of just a few cores with lots of cache memory that can handle a few software threads at a time.

In contrast, a **GPU** is composed of hundreds of cores that can handle thousands of threads simultaneously.

GPU is specialized for compute intensive, highly data parallel computation

- More area is dedicated to processing
- Good for high arithmetic intensity programs with a high ratio between arithmetic operations and memory operations.

CPU
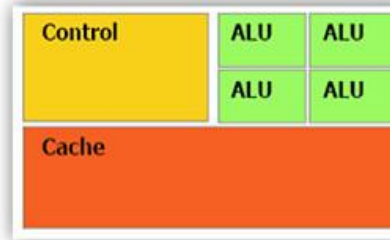
- Low compute density
- Complex control logic
- Large caches (L1$/L2$, etc.)
- Optimized for serial operations
  - Fewer execution units (ALUs)
  - Higher clock speeds
- Shallow pipelines (<30 stages)
- Low Latency Tolerance
- Newer CPUs have more parallelism

GPU
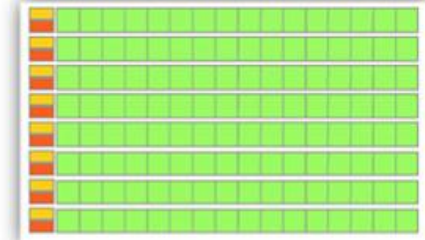
- High compute density
- High Computations per Memory Access
- Built for parallel operations
  - Many parallel execution units (ALUs)
  - Graphics is the best known case of parallelism
- Deep pipelines (hundreds of stages)
- High Throughput
- High Latency Tolerance
- Newer GPUs:
  - Better flow control logic (becoming more CPU-like)
  - Scatter/Gather Memory Access
  - Don't have one-way pipelines anymore

CPU vs GPU Comparison

	CPU: Latency-oriented design	GPU: Throughput Oriented Design
Clock	High clock frequency	Moderate clock frequency
Caches	Large sizes Converts high latency accesses in memory to low latency accesses in cache	Small caches To maximize memory throughput
Control	Sophisticated control system Branch prediction to reduce latency due to branching Data loading to reduce latency due to data access	Single controlled No branch prediction No data loading
Powerful Arithmetic Logic Unit (ALU)	Reduced operation latency	Numerous, high latency but heavily pipelined for high throughput
Other aspects	Lots of space devoted to caching and control logic. Multi-level caches used to avoid latency Limited number of registers due to fewer active threads Control logic to reorganize execution, provide ILP, and minimize pipeline hangs	Requires a very large number of threads for latency to be tolerable
Beneficial aspects for applications	CPUs for sequential games where latency is critical. CPUs can be 10+X faster than GPUs for sequential code.	GPUs for parallel parts where throughput is critical. GPUs can be 10+X faster than GPUs for parallel code.

GPU (Graphics Processing Unit)

- **GPU** is the chip in computer video cards, PS3, Xbox, etc
  - Designed to realize the 3D graphics pipeline
    - Application → Geometry → Rasterizer → Image
- **GPU** development:
  - Fixed graphics hardware
  - Programmable vertex/pixel shaders
  - **GPGPU**
    - General purpose computation (beyond graphics) using GPU in applications other than 3D graphics
    - GPGPU can be treated as a co-processor for compute intensive tasks
      - With sufficient large bandwidth between CPU and GPU.

GPU Performance
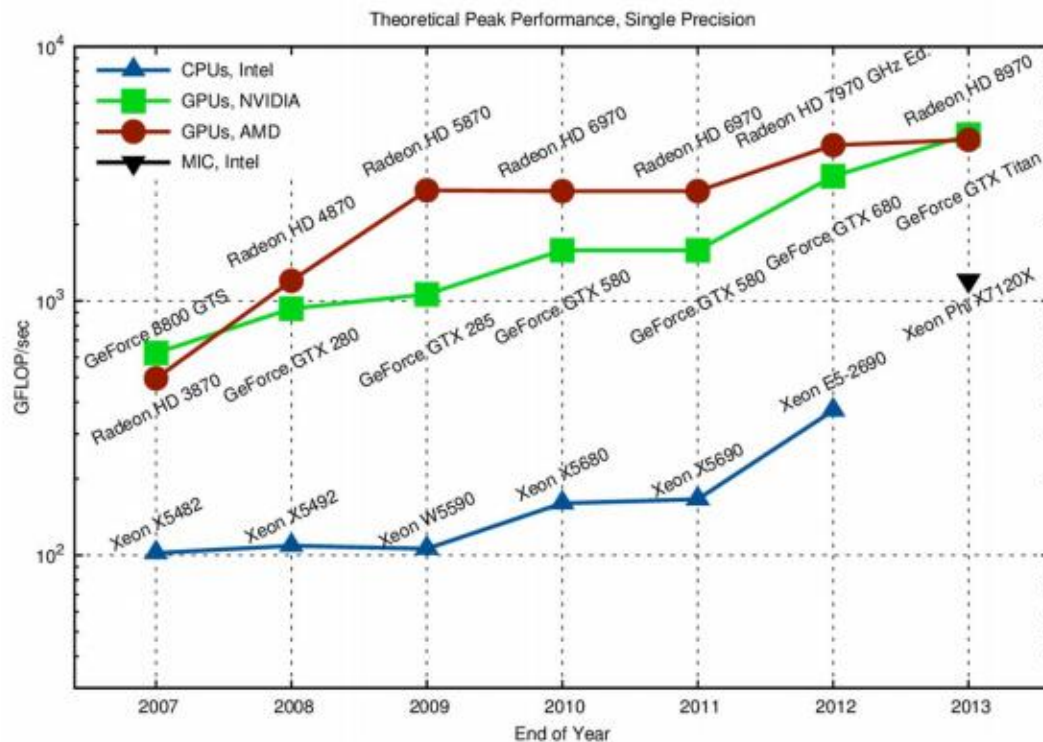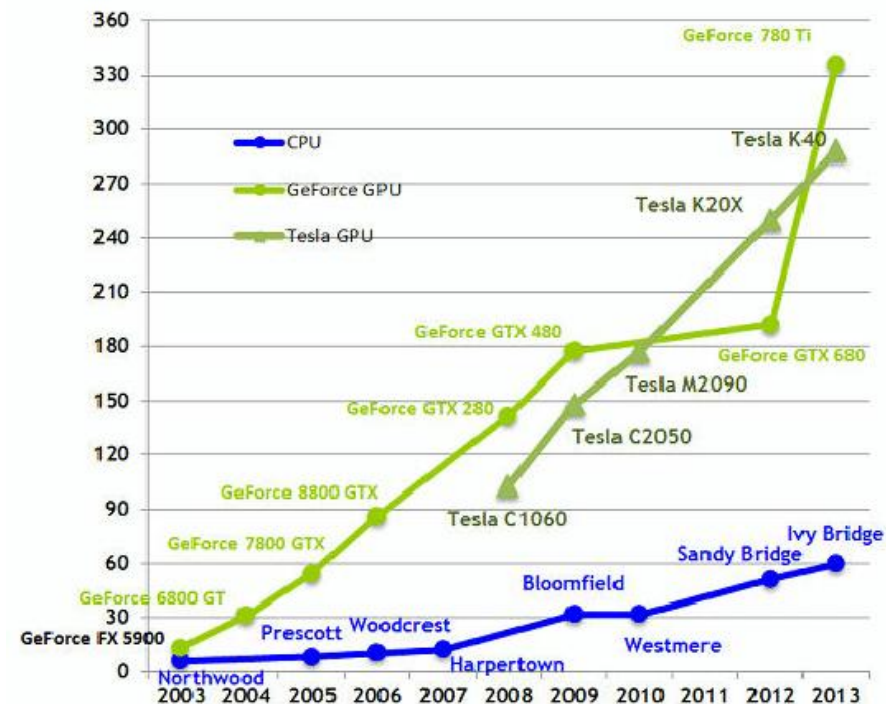
GPU Memory Bandwidth

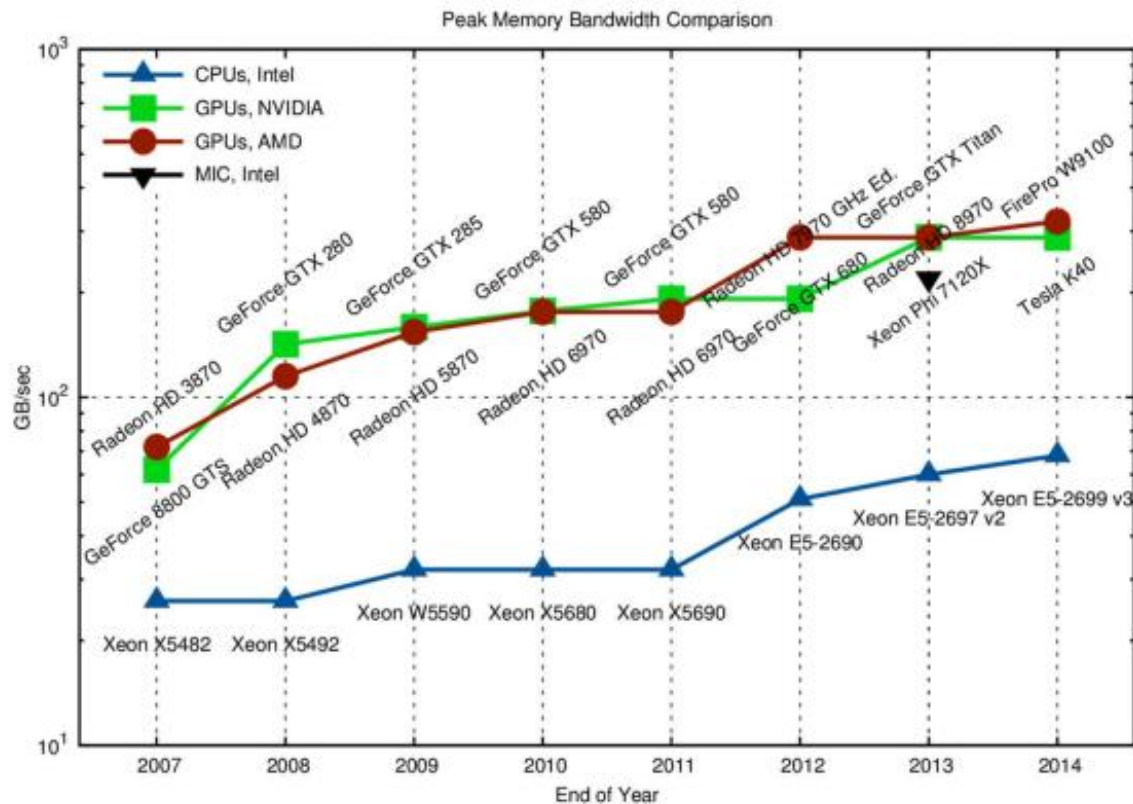
GPU memory bandwidth is a measure of the data transfer speed between a GPU and the system across a bus, such as PCI Express (PCIe) or Thunderbolt.

Peak Memory Bandwidth Comparison

- CPUs, Intel
- GPUs, NVIDIA
- GPUs, AMD
- MIC, Intel

CPU and GPU CONNECTION

Processing flow on CUDA
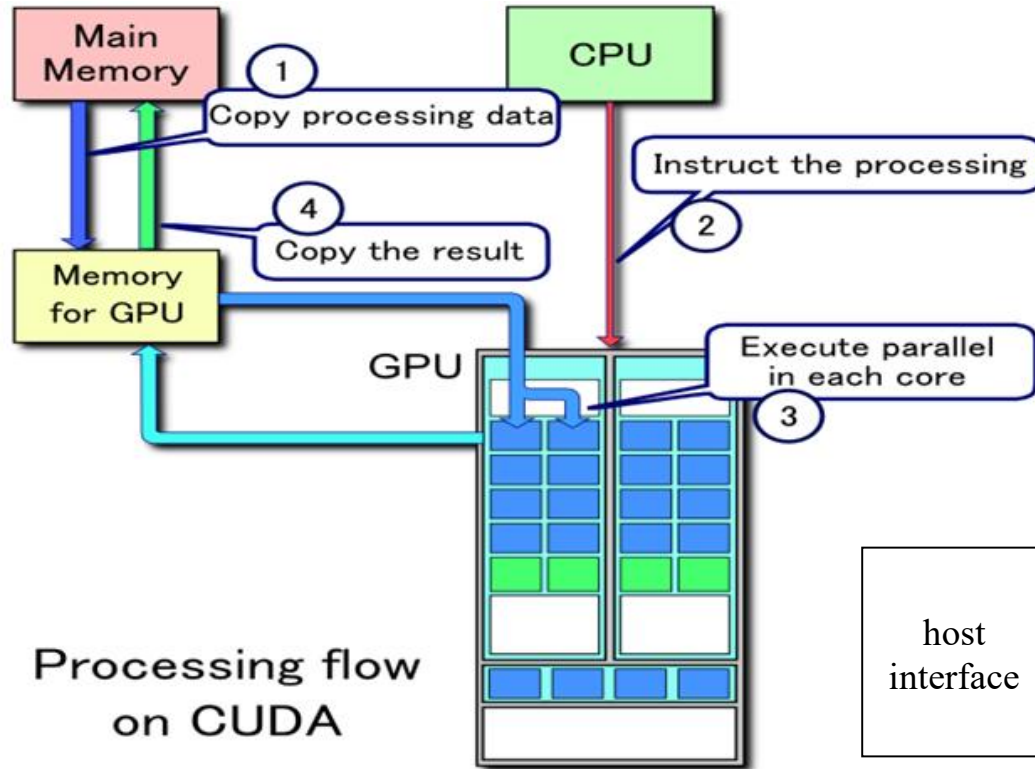
The GPU pipeline

▸ **GPU** receives geometry information from the **CPU** as an input and provides a picture as an output

host interface	vertex processing	triangle setup	fragment processing	memory interface

Native GPU code: HIP/CUDA

CUDA *from NVIDIA*
- ➢ Has been a standard for native GPU code for years
- ➢ Extensive set of optimized libraries available
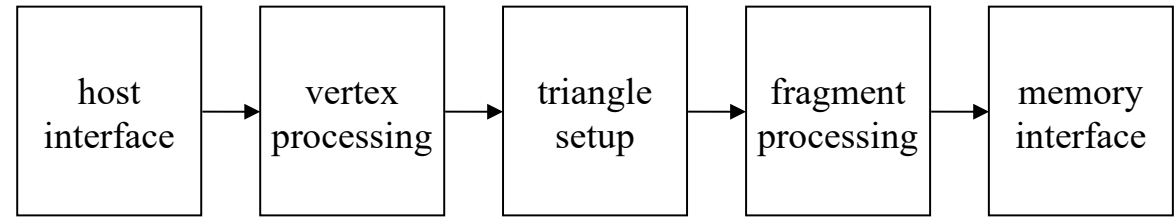- ➢ Custom syntax (extension of C++) supported only by CUDA compilers
- ➢ Support only f or NVIDIA devices

HIP *(Heterogeneous-computing Interface for Portability) from AMD*
- ➢ AMD effort to offer a common programming interface that works on both CUDA and ROCm devices
- ➢ Standard C++ syntax, uses nvcc/hcc compiler in the background
- ➢ Almost a one-on-one clone of CUDA from the user perspective
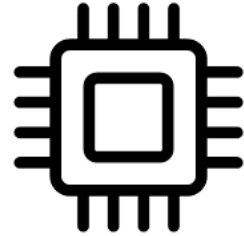- ➢ Ecosystem is new and developing fast

AMD and **NVIDIA** offers also a wide set of optimized libraries and tools

GPGPU

General **P**urpose computation

on **G**raphics **P**rocessing **U**nits.

GPGPU (General-Purpose Graphics Processing Unit)

- **GPGPU:** Using graphic hardware for **non-graphic** computations

- Prefect for **massive parallel processing** on data paralleled applications

- **GPU** acts as an "**accelerator**" to the CPU (heterogeneous system)
    - Most lines of code are executed on the CPU (serial computing)
    - Key computational kernels are executed on the GPU (stream computing)
        - Taking advantage of the large number of cores and high graphics memory bandwidth

- **GPUs** now firmly established in HPC industry
    - Can augment each node of parallel system with GPUs

GPGPU: Programming Considerations

- Standard (CPU) code *will not run on* a GPU unless it is adapted

- Programmer must
  - decompose problem onto the hardware in a specific way (e.g. using a hierarchical thread/grid model in CUDA)
  - Manage data transfers between the separate CPU and GPU memory spaces.
  - Traditional language (C, C++, Fortran etc) not enough, need extensions, directives, or new language.

- Once code is ported to GPU, optimization work is usually required to tailor it to the hardware and achieve good performance

- Many researchers are now successfully exploiting GPUs
  - Across a wide range of application areas

GPGPU (General-Purpose Graphics Processing Unit)

Architecture
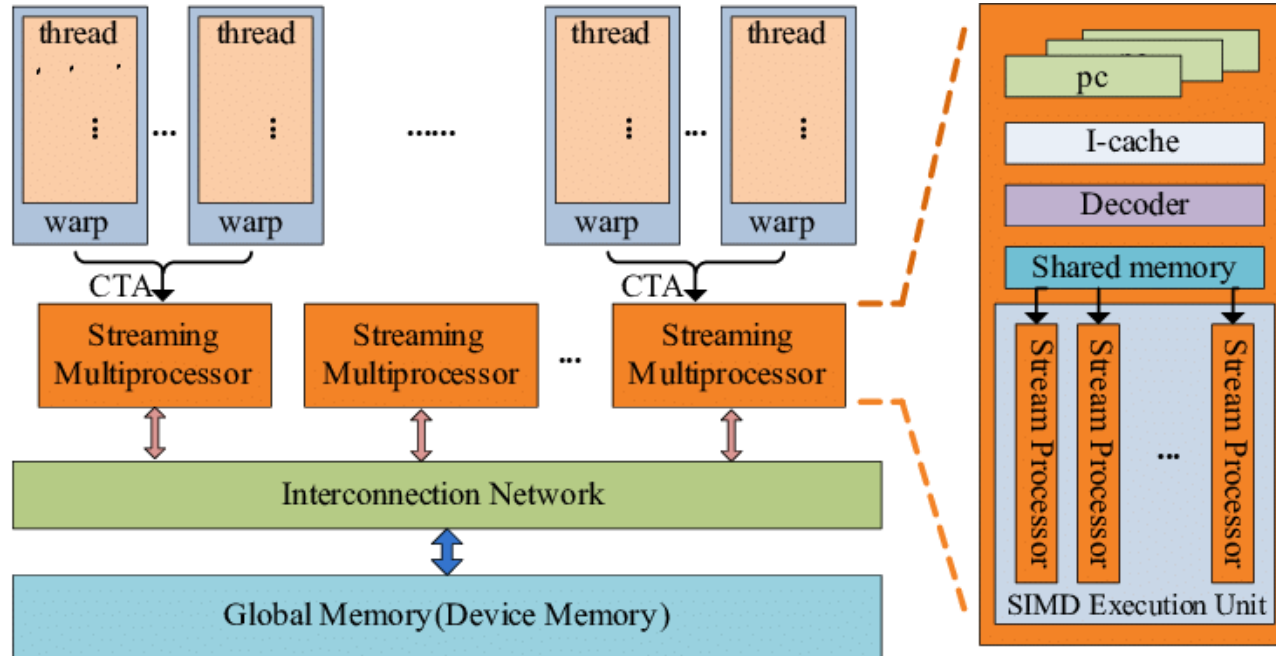
A GPGPU is a GPU that is programmed for purposes beyond graphics processing,

such as performing computations typically conducted by a Central Processing Unit (CPU).

GPGPU (General-Purpose Graphics Processing Unit)

Advantages of GPGPU

➢ GPUs have many *more cores* than CPUs, which allows them to process large amounts of data in parallel. This can result in **significant speedups** for some problems, especially those involving matrices, vectors, images or graphics.

➢ GPUs **can also handle floating point operations** more efficiently than CPUs, which is important for scientific computing and machine learning applications.

➢ GPUs **can be used to accelerate various domains** such as computer vision, natural language processing, cryptography, bioinformatics, physics simulation, etc.

GPGPU (General-Purpose Graphics Processing Unit)

Disadvantages of GPGPU

➢ GPUs are not suitable for all kinds of problems, especially those that require sequential or branching logic, complex data structures, or synchronization among threads.

➢ GPUs have **limited memory** and **bandwidth** compared to CPUs, which can limit the amount of data that can be transferred or processed at once.

➢ GPUs require **specialized programming languages** and APIs to access their features, which can increase the complexity and learning curve for developers.

GPGPU Programming Languages and APIs

There are several options for programming GPGPU applications:

CUDA: A proprietary platform developed by Nvidia that allows programmers to write C/C++ code that runs directly on Nvidia GPUs. It also provides libraries and tools for various domains such as linear algebra, image processing, deep learning, etc.

OpenCL: An open standard developed by the Khronos Group that supports multiple platforms and devices, including CPUs, GPUs, FPGAs, etc. It defines a C-like language and a runtime API for executing kernels on heterogeneous devices.

DirectCompute: A Microsoft API that is part of DirectX 11 and 12 that enables GPGPU programming on Windows platforms. It supports HLSL shaders and C++ AMP extensions for writing compute kernels.

Metal: An Apple API that provides low-level access to the GPU on iOS and macOS platforms. It supports Swift and Objective-C languages for writing compute shaders.

Vulkan: A cross-platform API developed by the Khronos Group that provides low-level access to the GPU and other devices. It supports SPIR-V as an intermediate language for writing compute shaders.

ROCm: is an Advanced Micro Devices (AMD) software stack for graphics processing unit (GPU) programming and spans several domains: GPGPU, HPC ,heterogeneous computing

Programming **i**nterface for **p**arallel **c**omputing

병렬 컴퓨팅을 위한 프로그래밍 인터페이스

Programming Interface

MPI (**M**essage **P**assing **I**nterface) and

OpenMP (**O**pen **M**ulti-**P**rocessing)

Programming interface MPI, OpenMP

MPI, OpenMP two complementary parallelization models.

MPI (Message Passing Interface) is a multi-process model whose mode of communication between the processes is **explicit** (communication management is the responsibility of the user).

Generally used on multiprocessor machines with distributed memory.

It is a library for passing messages between processes without sharing.
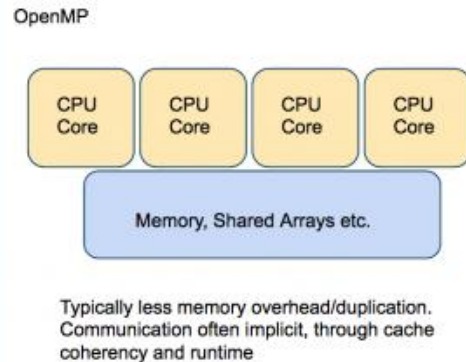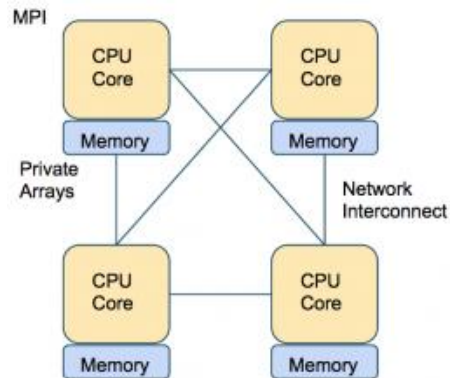
OpenMP (Open Multi-Processing) is a multitasking model whose mode of communication between tasks is implicit (the management of communications is the responsibility of the compiler).

OpenMP is used on shared-memory multiprocessor machines. It focuses on shared memory paradigms. It is a language extension for expressing data-parallel operations (usually parallelized arrays over loops).

MPI vs OpenMP

✅ **MPI**	**OpenMP**
Portable to a distributed and shared memory machine. Scale beyond a node No data placement issues	Easy to implement parallelism Implicit communications Low latency, high bandwidth Dynamic Load Balancing
❌ **MPI**	**OpenMP**
Explicit communication High latency, low bandwidth Difficult load balancing	Only on nodes or shared memory machines Scale on Node Data placement problem

Thank you for your attention !