University of CINCINNATI

Principles of Parallel Computing
(Mastering Cloud Computing: Chapter#2)

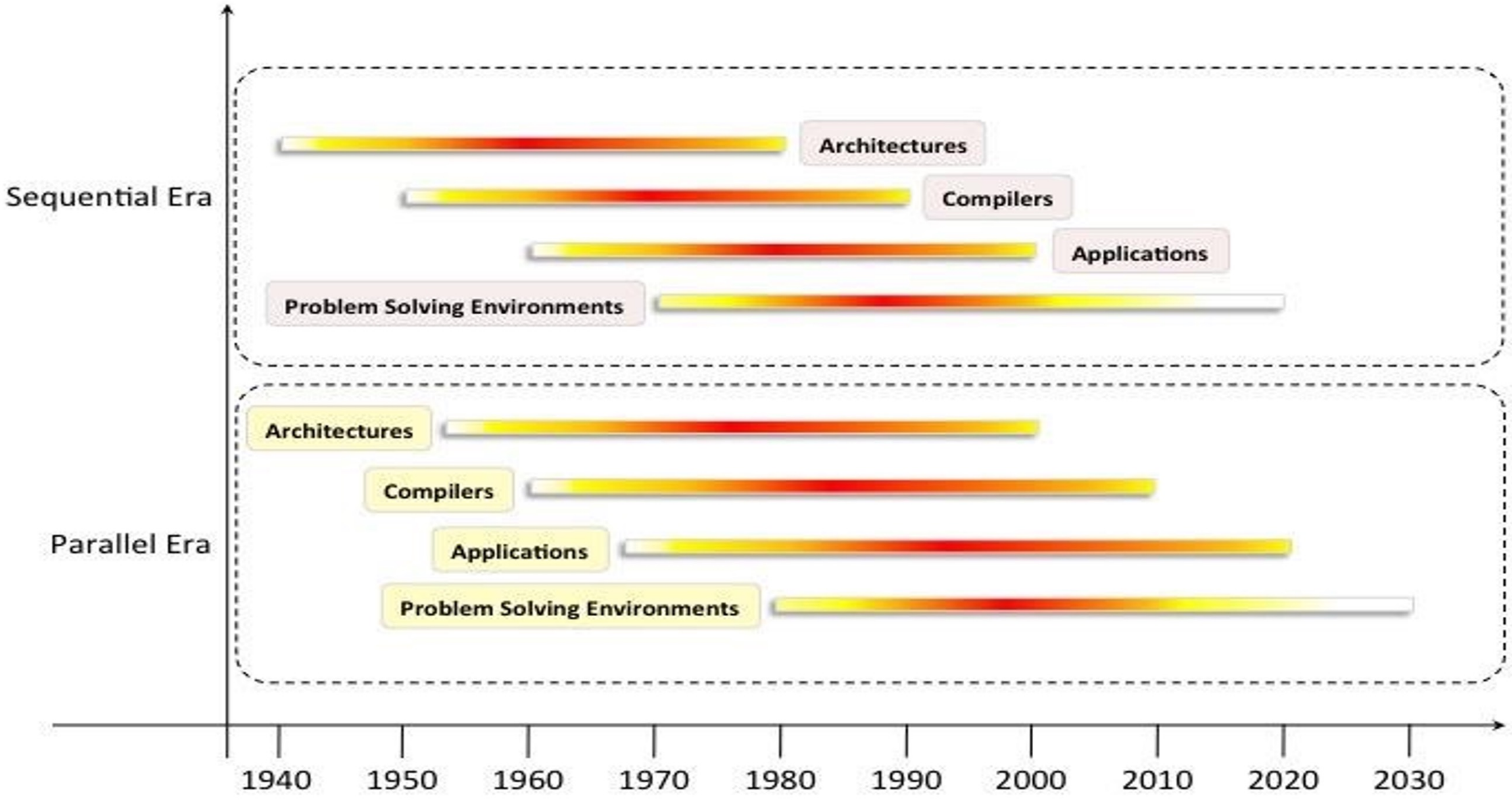
Rashmi Kansakar

Computing Eras

- ➢ Sequential - 1940s+
- ➢ Parallel and Distributed - 1960s+

- ➔ But… CS typically teaches sequential
- ➔ Parallel programming is hard!
- ➔ Moore’s Law (modern CPUs) now require us into write parallel programs.

Computing Eras…



Von Neumann Architecture

Named after the Hungarian mathematician John von Neumann who first authored the general requirements for an electronic computer in his 1945 papers

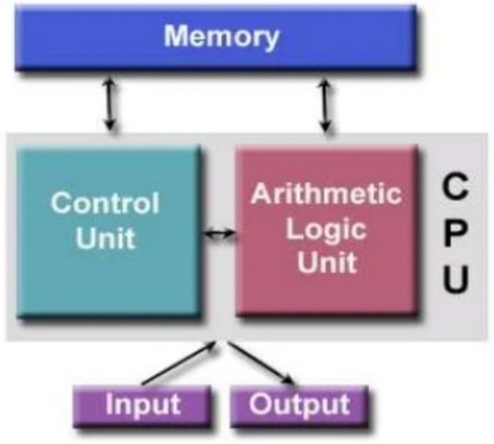
Since then, virtually all computers have followed this basic design

Comprises of four main components:

- **RAM** is used to store both program instructions and data
- **Control unit** fetches instructions or data from memory,

  Decodes instructions and then ***sequentially*** coordinates operations

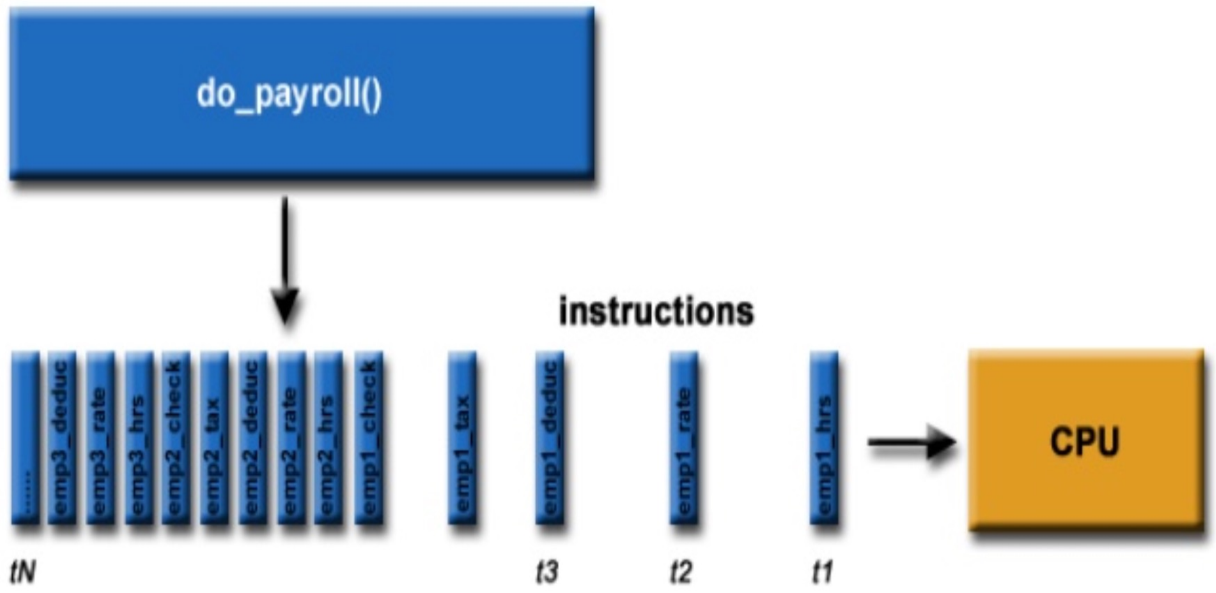
  to accomplish the programmed task.
- **Arithmetic Unit** performs basic arithmetic operations
- **Input-Output** is the interface to the human operator



What is Serial Computing?

Pass it on... Knowledge is power

Traditionally, software written for serial computation architecture

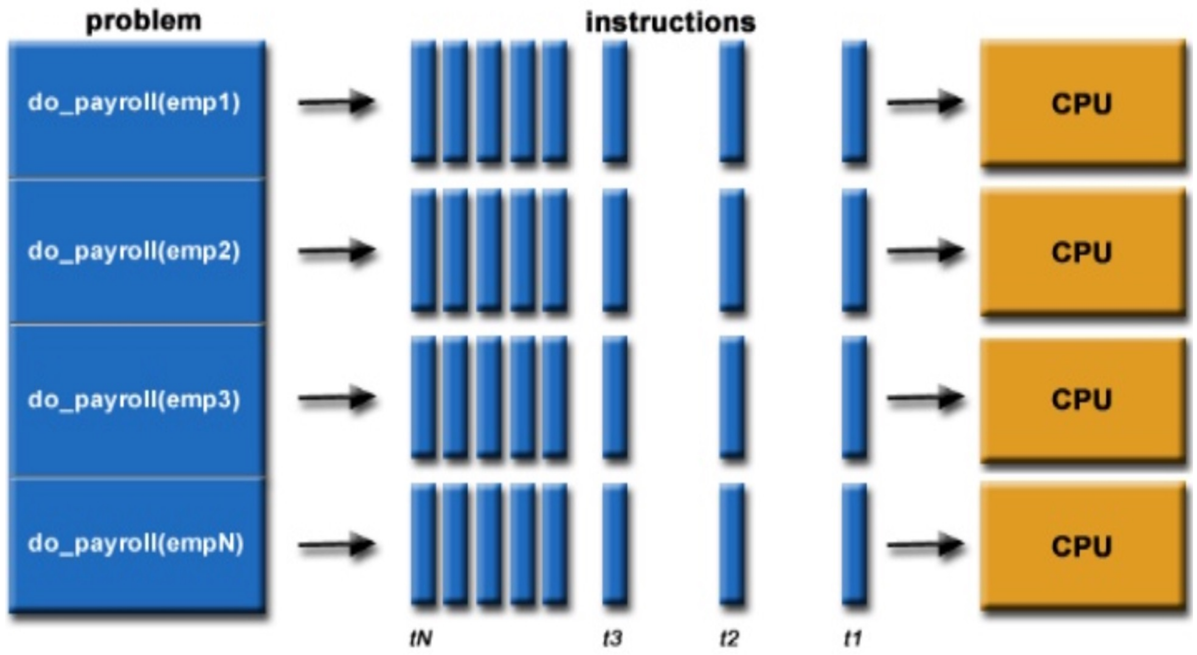


- ✓ Run on a single computer having a single Central Processing Unit (CPU)
- ✓ A problem is broken into a discrete series of instructions
- ✓ Instructions are executed one after another
- ✓ Only one instruction may execute at any moment in time

What is Parallel Computing?

Pass it on... Knowledge is power

Parallel computing is the simultaneous use of multiple compute resources to solve a computational problem



- ✓ Breaking the problem into independent parts
- ✓ Each processing element execute its part of algorithm simultaneously with the others
- ✓ Computational problem solved in less time with multiple compute resources than with a single compute resource
- ✓ Compute resources can be a single computer with multiple processors or Several networked computers

Parallel *Vs.* Distributed

- Parallel - tightly coupled system
  - Computation divided among processors sharing common memory
  - Homogeneous components. Each processor same type & capacity
  - Defined loosening with InfiniBand & distributed memory

- Distributed – Architecture that allows computation is broken down into units and executed concurrently
  - Parallel a subtype - distributed more general
  - Different nodes, processors, or cores
  - Heterogeneous components
  - E.g GRID computing, Internet computing systems

Parallel Computing

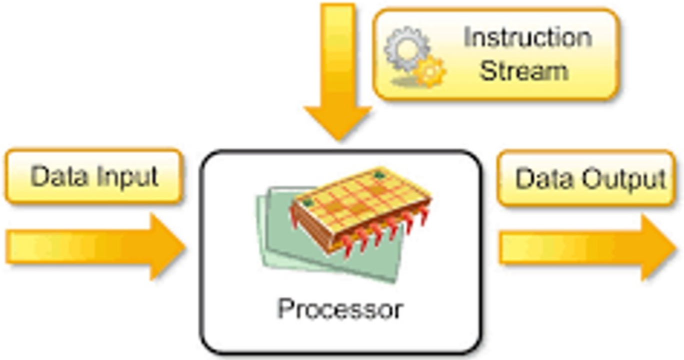
Renewed interest…

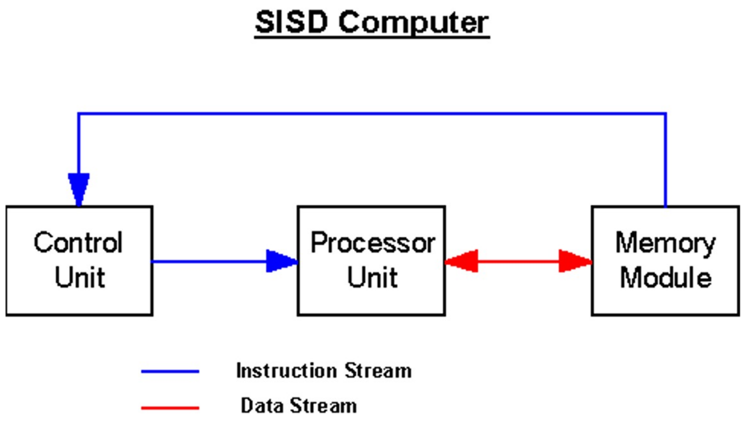
- ➢ Larger computation tasks
- ➢ CPU have reached physical limits
- ➢ Hardware features (pipelining, superscalar, etc) require complex compilers - reached limits
- ➢ Vector processing effective, but applicability isolated
- ➢ Networking technology mature

Hardware Architectures

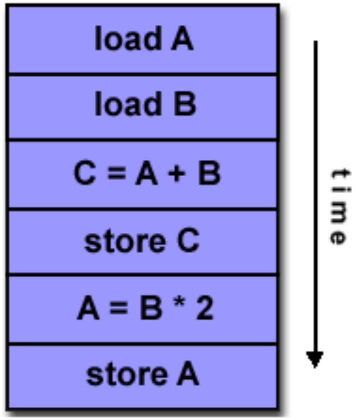
- Single instruction, single data (SISD)
- Single instruction, multiple data (SIMD)
- Multiple instruction, single data (MISD)
- Multiple instruction, multiple data (MIMD)

Single Instruction Single Data (SISD)

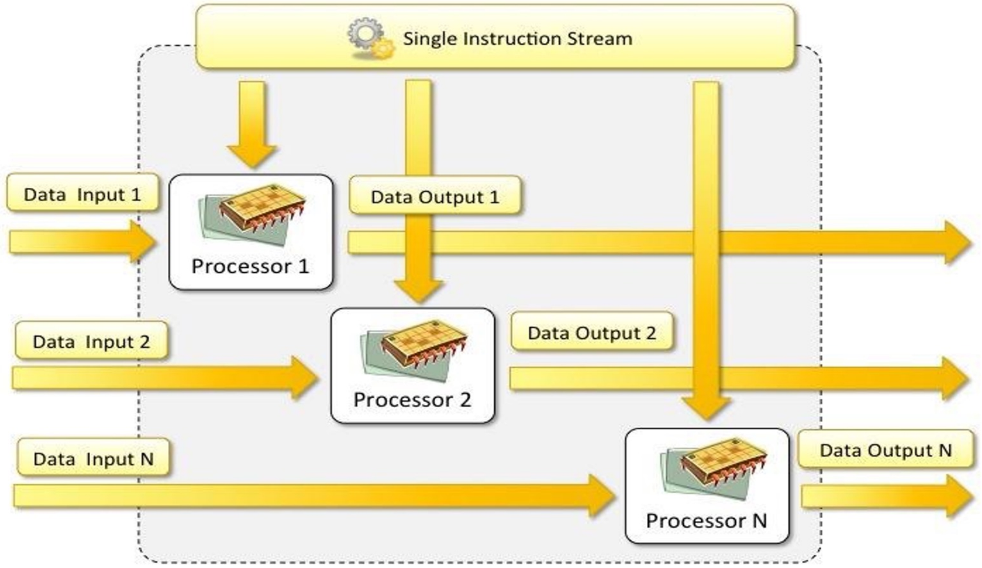


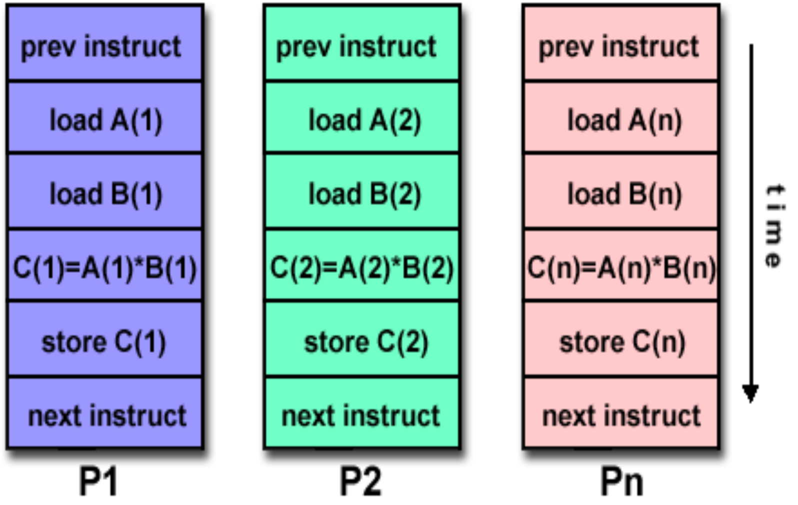


- **Sequential computers** (no parallel instruction/data streams)
- **Single Instruction** is being acted by the CPU in 1 clock cycle
- **Single Data** being used as input during any 1 click cycle
- Older generation mainframes, minicomputers
- ‘Normal’ computers, modern day PCs, Macs
- CS1, CS2, DS - typically programming



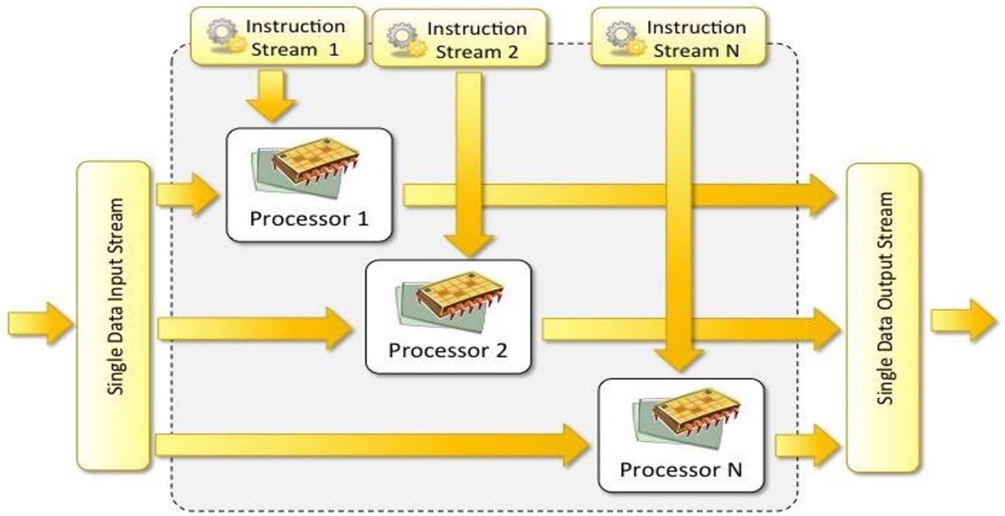
Single Instruction Multiple Data (SIMD)

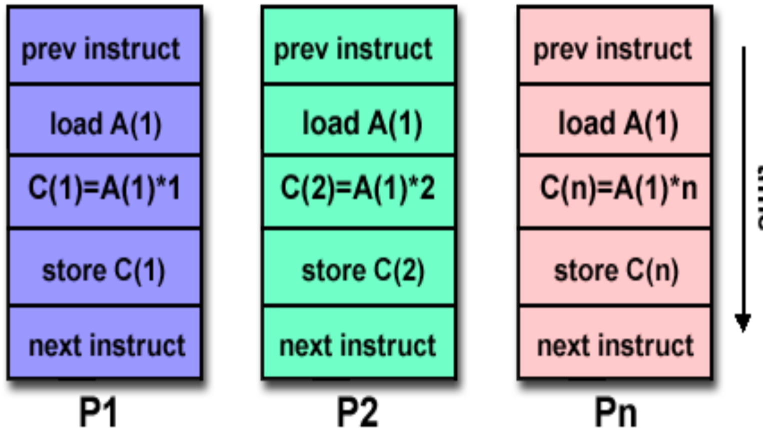




- Multiple data streams against a single instruction stream to perform operations
- Multiple Data: Each processing unit can operate on a different data element
- Vector Processor 1 instruction operate on 1D arrays of data called vectors
- Scientific workloads, vector and matrix operations
- GPUs (CUDA), Sony PS3 Cell processor (1, 2), Cray's vector processor

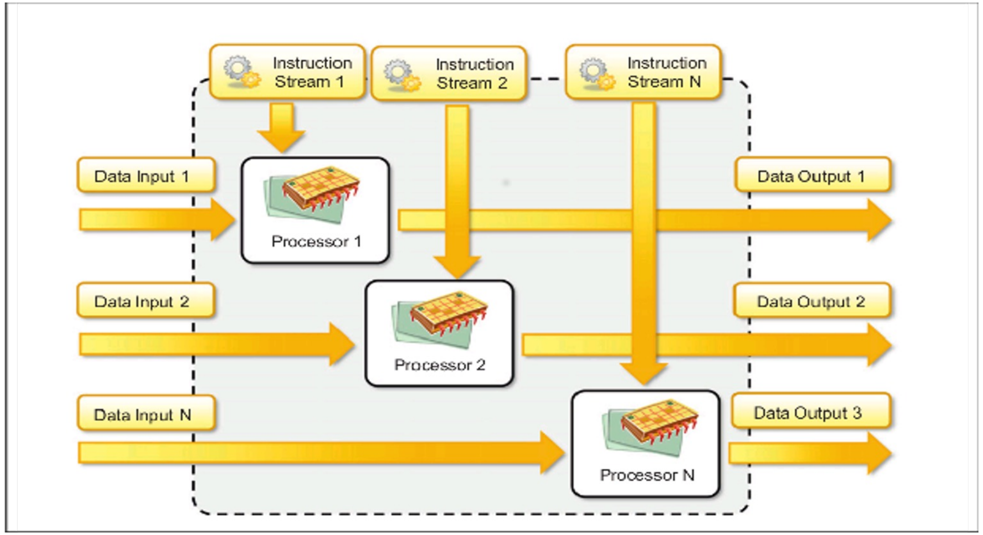
Multiple Instruction Single Data (MISD)

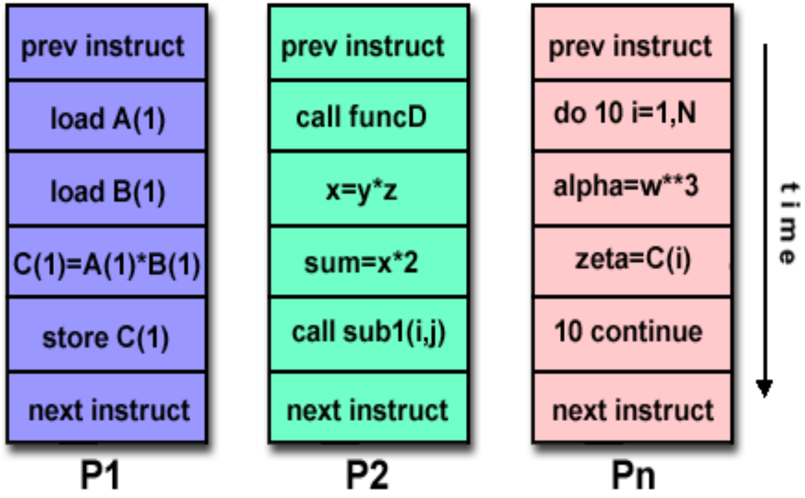




- Each processing unit operates on data independently via separate instruction streams. Example y = sin(x) + cos(x) + tan(x)
- A single data stream is fed into multiple processing units
- No commercial machines exist, though CPU superscalar and pipelining have a similar feel

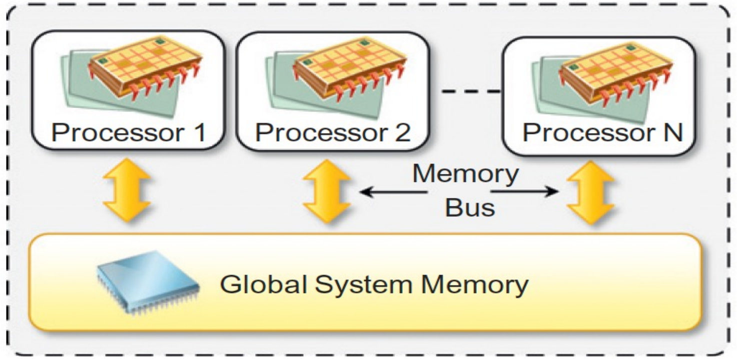
Multiple Instruction Multiple Data (MIMD)

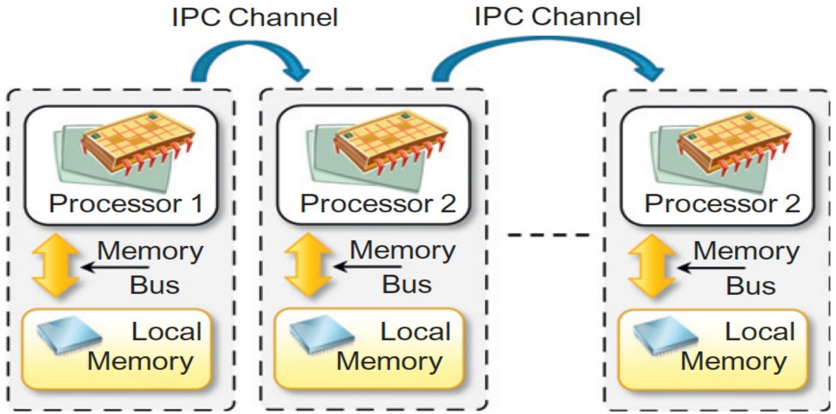




- Multiple autonomous processors simultaneously executing different instructions on different data
- Multiple Instruction: Every processor may be executing a different instruction stream
- Multiple Data: Every processor may be working with a different data stream
- Asynchronous transfers are generally faster than synchronous transfers
- Most supercomputers, networked parallel clusters, grids, clouds, multi-core PCs

Memory Architectures in MIMD





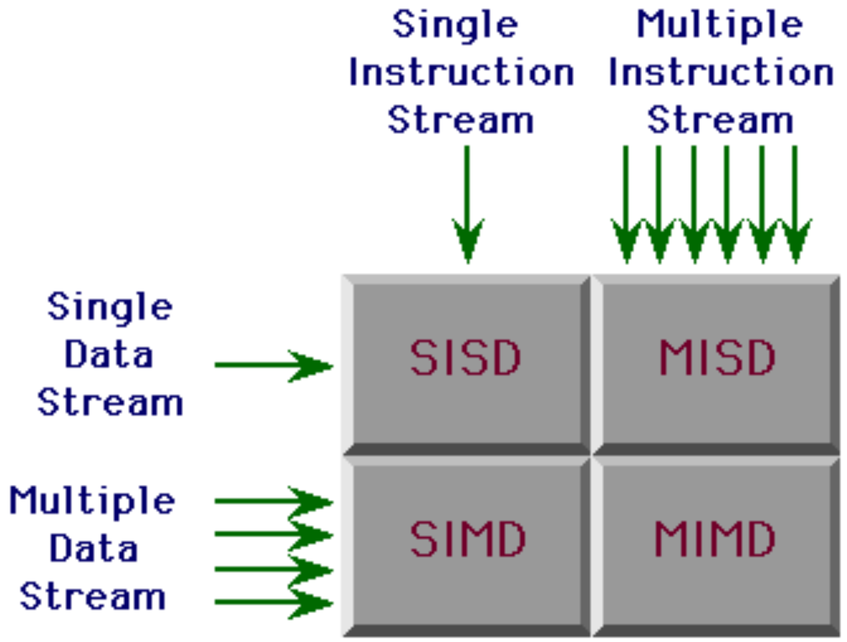
Shared Memory
- All PEs are connected to a single global memory and have access to it
- Tightly-coupled multiprocessor systems
- Communication thru shared memory
- Shared MIMD easier to program, less tolerant to failure, and harder to scale
- Failure affect entire entire system

Distributed-memory
- All PEs have local memory. Loosely-coupled multiprocessor systems
- Cost effectiveness: can use commodity, off-the-shelf processors
- Failure can be isolated. Popular
- Each processor can access its own memory without interference

Hardware Architectures

University of CINCINNATI

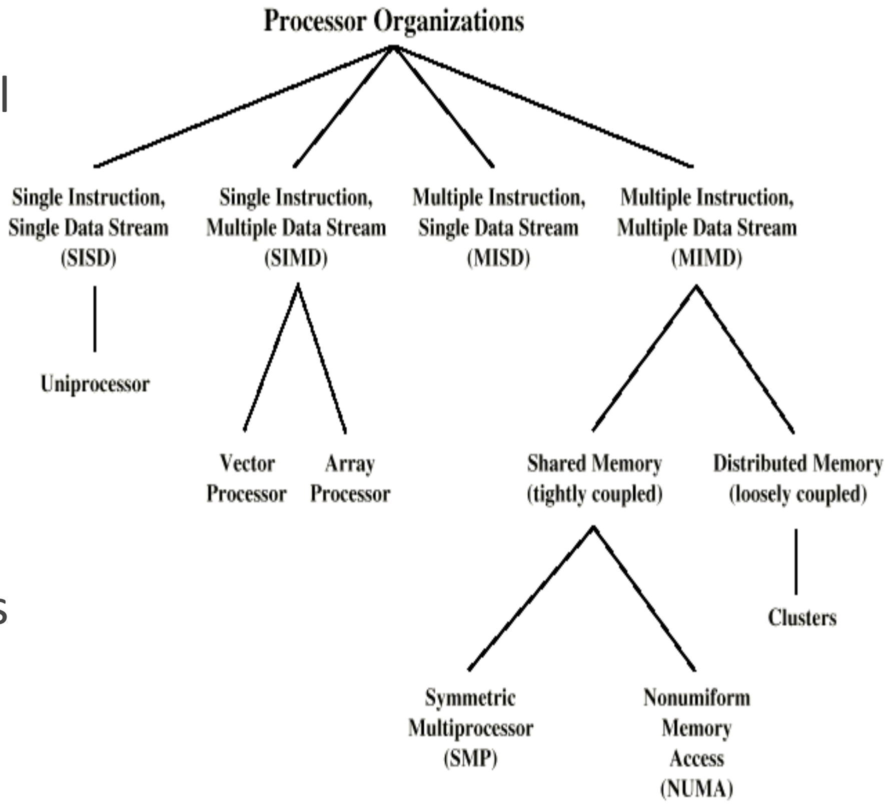
Flynn's taxonomy: Based upon the number of concurrent instruction and data streams available in the architecture



- SISD
  Uniprocessors
- SIMD
  Vector Processors
  Parallel Processing
- MISD
  Maybe Pipelined Computers
- MIMD
  Multi-Computers
  Multi-Processors

Parallel Processor Architectures

- The computers of today, and tomorrow, have tremendous processing power that require parallel programming to fully utilize.
- There are significant differences between sequential and parallel programming, that can be challenging.
- With early exposure to these differences, students are capable of achieving performance improvements with multicore programming.


Processor Organizations
Single Instruction, Single Data Stream (SISD)
Single Instruction, Multiple Data Stream (SIMD)
Multiple Instruction, Single Data Stream (MISD)
Multiple Instruction, Multiple Data Stream (MIMD)
Uniprocessor
Vector Processor
Array Processor
Shared Memory (tightly coupled)
Distributed Memory (loosely coupled)
Clusters
Symmetric Multiprocessor (SMP)
Nonuniform Memory Access (NUMA)

How to Program in Parallel?

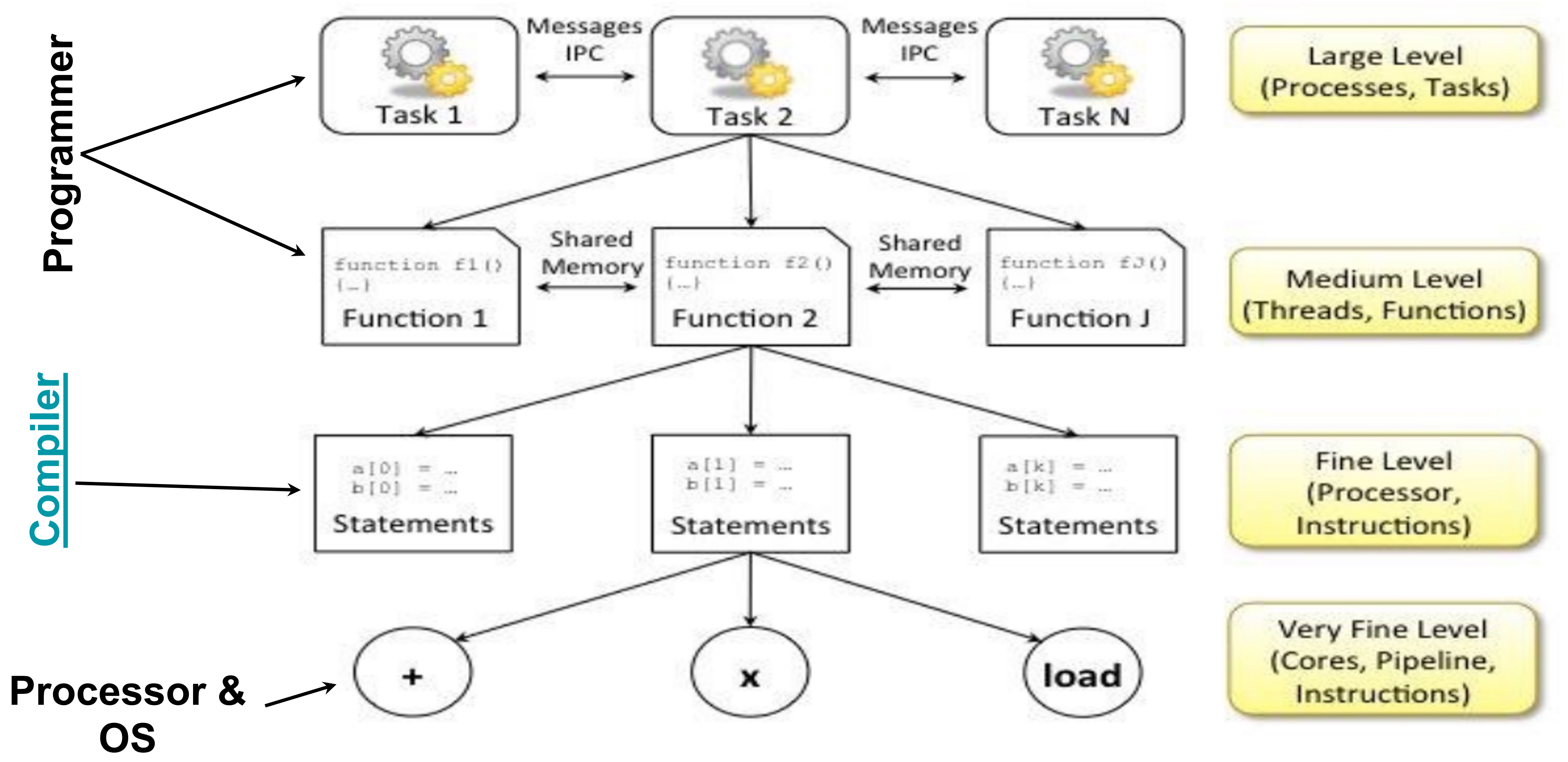
- Problem specific!
- Approaches:
  - ***Data parallelism***
    - MapReduce (data based)
  - ***Process parallelism***
    - Game/Cell Processor (code based)
  - ***Farmer-and-worker model***
    - Web serving (Apache) (thread based)

Level of Parallelism

- Goal?
  - Never have a processor idle!
  - ‘Grain size’ important
    - How you break up the problem.

Grain Size	Code Item	Parallelized By
Large (task)	Separate (heavyweight process)	Programmer
Medium (control)	Function or procedure (thread)	Programmer
Fine	Loop or instruction block	Compiler
Very Fine	Instruction	Processor or OS

Level of Parallelism…



Limit to Parallelism (*yet*)

Linear speedup not possible

- Doubling # cores doesn't double speed
  - Communication overhead
- General guidelines:
  - Computation speed = sqrt(system cost) or faster a system becomes the more expensive it is to make it faster
  - Speed of parallel computer increases as the log(n) of processors (*n = number of processors*)

Parallel Overhead

- Sequential program runs in a single processor, and has single line of control
- Parallel programming is making many processors collectively work on a single program

- **Parallel Overhead:**
    - The amount of time required to coordinate parallel tasks, as opposed to doing useful work.

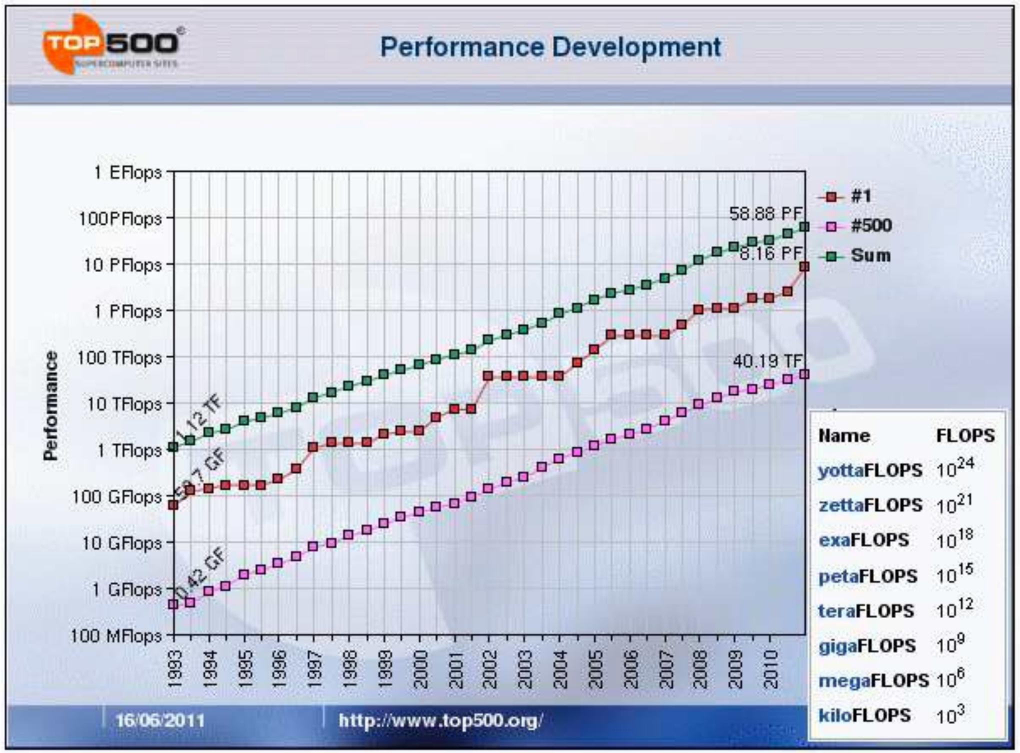
- **Parallel overhead can include factors such as:**
    - Task start-up time
    - Synchronizations
    - Data communications
    - Software overhead imposed by parallel languages, libraries,
    - operating system, etc.
    - Task termination time

Why Use Parallel Computing ?

- **Save time and money**
  - In theory, throwing more resources at a task will shorten its time to completion
  - Parallel computers can be built from cheap, commodity Components
- **Solve larger problems**
  - Many problems are so large and complex that it is impractical or impossible to solve them on a single computer, especially given limited computer memory
- **Provide concurrency**
  - A single compute resource can only do one thing at a time.
  - Multiple computing resources can be doing many things simultaneously
- **Limits to serial computing**
  - Increasingly expensive to make a single processor faster
  - Using a larger number of moderately fast commodity processors to achieve the same or better performance is less expensive

The Future

Trends indicated by ever faster networks, distributed systems, and multi-processor computer architectures clearly show that **PARALLELISM** is the future of



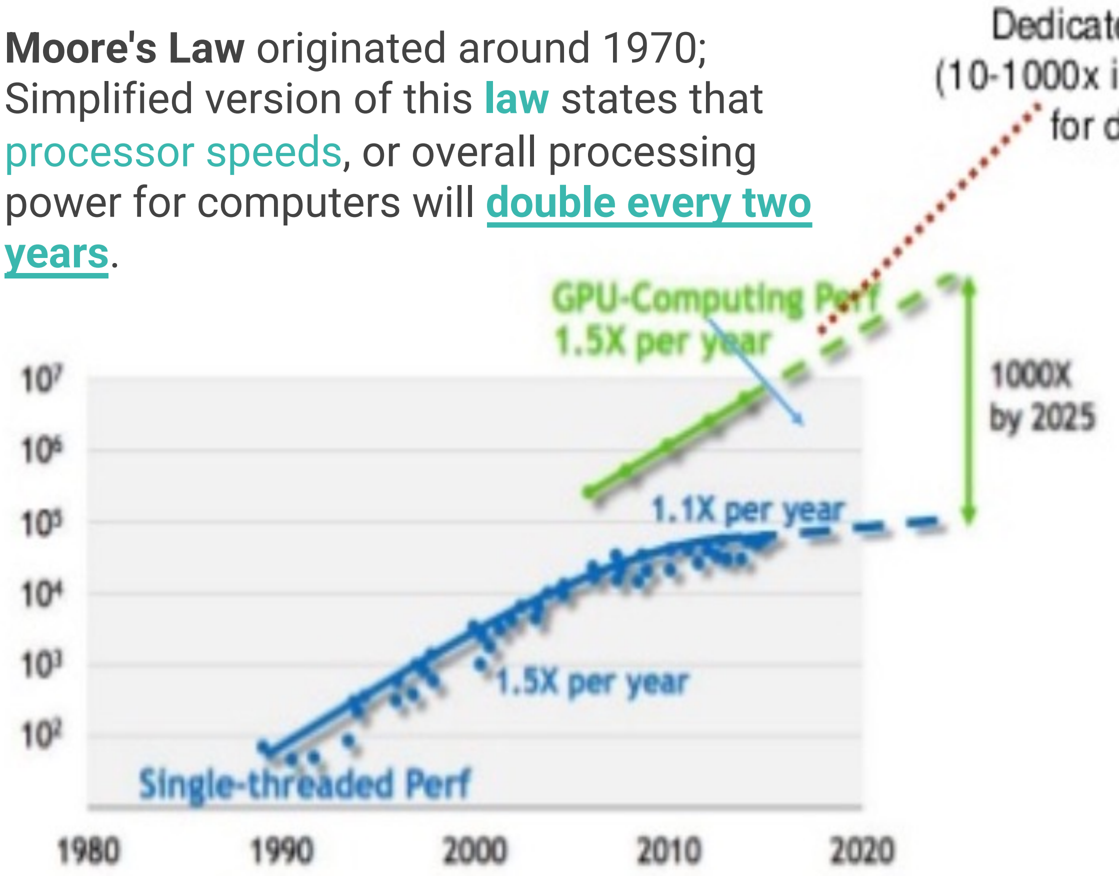
Name	FLOPS
yottaFLOPS	10^{24}
zettaFLOPS	10^{21}
exaFLOPS	10^{18}
petaFLOPS	10^{15}
teraFLOPS	10^{12}
gigaFLOPS	10^{9}
megaFLOPS	10^{6}
kiloFLOPS	10^{3}

- There has been a greater than 1000x increase in supercomputer performance, with no end currently in sight
- The race is already on for Exascale Computing!

1 exaFlops = 10^18,
FLOATING POINT OPERATIONS PER SECOND
FLOP machine will do one "operation" in a second.

Moore's Law and beyond

Moore's Law originated around 1970; Simplified version of this **law** states that processor speeds, or overall processing power for computers will **double every two years**.



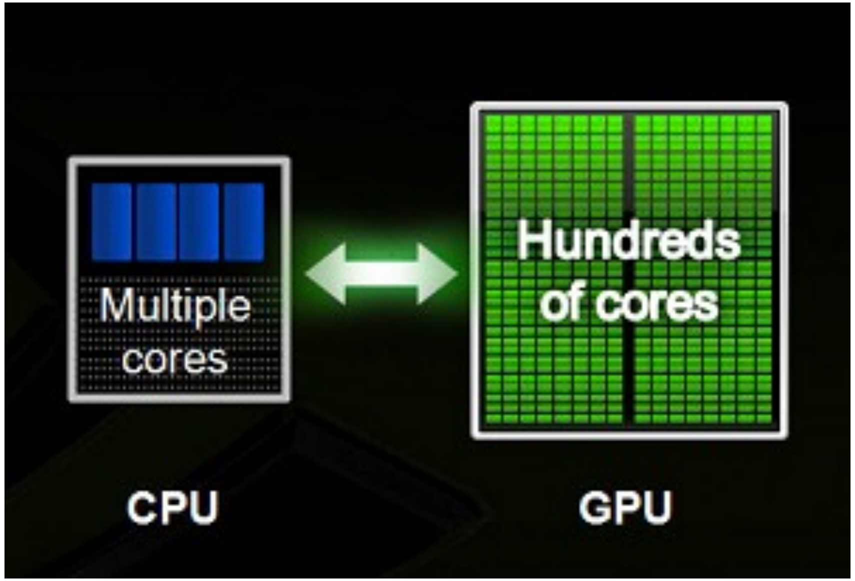
Moore's Law is no longer relevant, **GPUs** advancing faster pace than CPUs

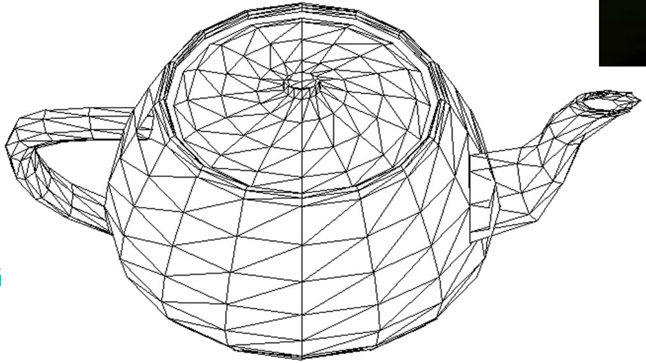
Driven by
massive computer demand
for artificial intelligence
which is the software glue
binding services together

Source: Nvidia Investor Day Presentation, 2017

Improving parallel processing CPU vs. GPU

- **CPU** is sometimes called the brains of a computer while a **GPU** acts as a specialized microprocessor.
- **CPU** is good at handling multiple tasks but a **GPU** can handle a few specific tasks very fast.
- **GPU (**graphical processing unit) is a programmable processor designed to quickly render high resolution images and video.
- **CUDA cores** are an Nvidia GPU's equivalent of CPU **cores**. They are optimized for running a large number of calculations simultaneously
- GPU render Billions of triangles per second
- **PARALLEL PROCESSING** in steroid!





Intel XEON PLATINUM 9282	
CPU Cores:	56 cores
Retail:	$50,000 +
Transistors :	8 Billion

NVIDIA TITAN V	
CUDA Cores:	5,120
Tensor Cores:	640
Retail:	$2,999
Transistors :	21 Billion

Beyond GPU to dedicated ML silicon

	CPU	GPU
1.	CPU stands for Central Processing Unit.	While GPU stands for Graphics Processing Unit.
2.	CPU consumes or needs more memory than GPU.	While it consumes or requires less memory than CPU.
3.	The speed of CPU is less than GPU's speed.	While GPU is faster than CPU's speed.
4.	CPU contain minute powerful cores.	While it contain more weak cores.
5.	CPU is suitable for serial instruction processing.	While GPU is not suitable for serial instruction processing.
6.	CPU is not suitable for parallel instruction processing.	While GPU is suitable for parallel instruction processing.
7.	CPU emphasis on low latency.	While GPU emphasis on high throughput.

Comparing GPU & CPU

MythBusters hosts: Adam and Jamie

Paint the Mona Lisa in 80 Milliseconds!

https://www.youtube.com/watch?v=WmW6SD-EHVY

Why Deep Learning uses GPUs

- Artificial intelligence with PyTorch and CUDA.
- CUDA cores are an Nvidia GPU's equivalent of CPU cores. They are optimized for running large number of calculations simultaneously
- Discuss how CUDA fits in with PyTorch, and more importantly, why we use GPUs in neural network programming.

https://www.youtube.com/watch?v=6stDhEA0wFQ