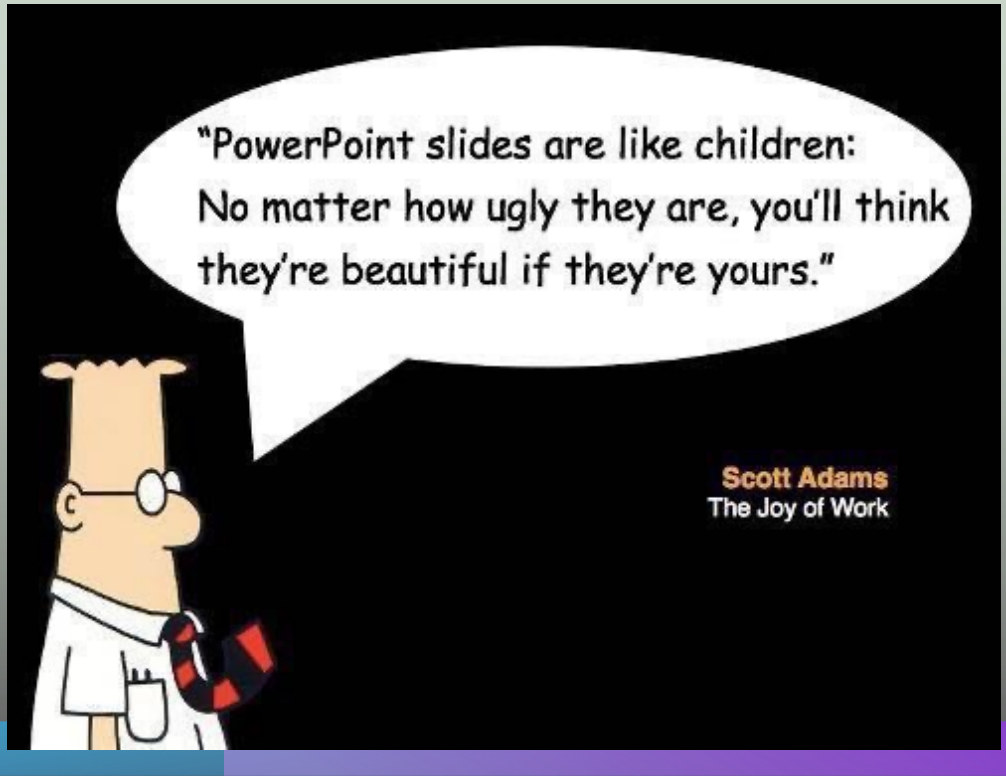
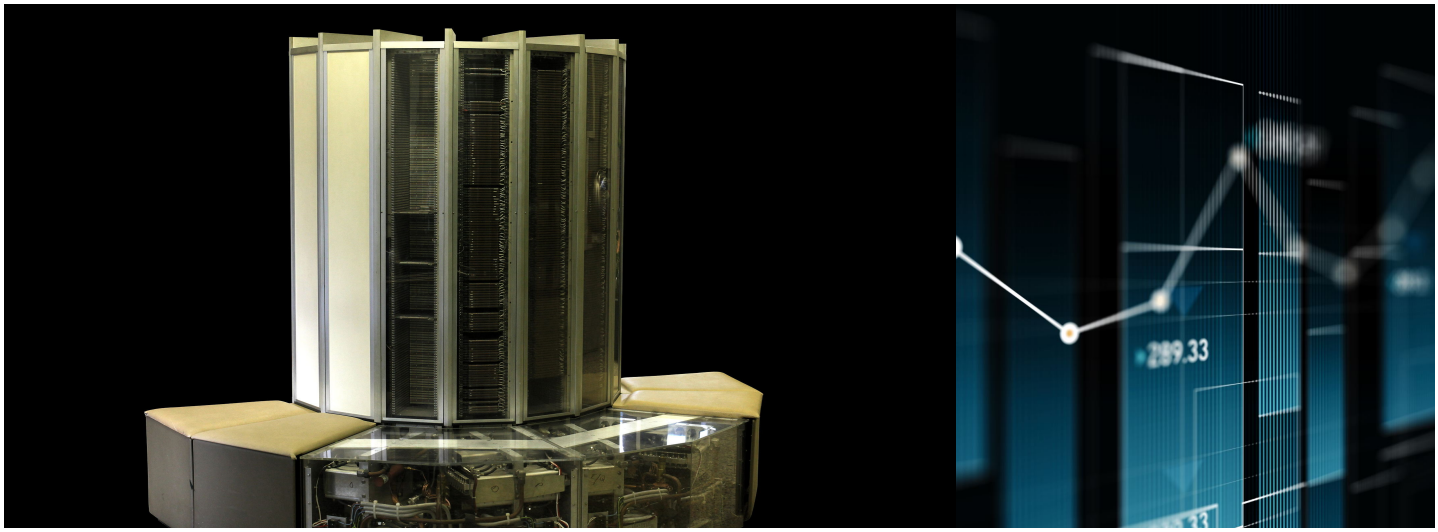




WHAT IS MASSIVE COMPUTING/HIGH PERFORMANCE COMPUTING AND HISTORY

WHAT IS MASSIVE COMPUTING

- We have to analyse 2 aspects:
  - Amount of data
  - Amount of calculae

	Small number of Data	High Number of Data
Small number of calculae by each data	Normal	M.C.
High number of calculae by each data	M.C.	M.C

OTHER PROBLEMS:

- High amount of data:
  - Fits in local RAM?
    - Yes: Good
    - No: What we can do?
      - Save in disk? Slow access
      - Split in several computers? Sharing Memory?
      - Transmits data between computers

MASSIVE COMPUTING

- Target:
  - Make all the calculae in a limited time
- Limits and restrictions
  - Fits all the data in memory (3rd and 4th level)
  - All the resources are finite

SMALL CONCEPTS

- **Multi-Task**: the ability to execute several tasks at "same time". The concept of "same time" vary depending on if it is simultaneously or switching between tasks
  - Human beings are multi-task switching between tasks (and always make mistakes)
  - Multi Tasking in computers depends on the Operating System (O.S):
    - Old O.S. and O.S. in critical systems are single tasks.
    - Modern O.S. and general-purpose O.S. are multi-task
  - Multi-Task O.S. does not imply its execute multiple tasks at the same time. It depends if the hardware has the resources.

- **Multi-processing**: Is the ability to execute multiple tasks at the same time simultaneously.
  - It requires multiple execution processor cores (CPU Cores).
- **CPU Core (**Computing Processor Unit Cor): Is the hardware that interprets and execute a set of instructions. Depending on the architecture, it could be more powerful or very simple.
- **CPU Processor**: Physical chip that contains one or several cores integrated. If it has several CPU cores, the processor includes the hardware it needs to coordinate all CPU Cores and their access to external resources.

HISTORY OF HIGH PERFORMANCE COMPUTATION

- In the beginning was the Mainframe
  - Mainframe: Big computers (size) with small computational capacity and memory
  - Very expensive
  - Scalars: only scalar and lineal operations
  - Multi-tasks
  - One CPU
  - Mutilple dummy terminals (just keyboard with printer or screen)

HISTORY OF HIGH PERFORMANCE COMPUTATION

This is the Cray-1.

Announced in 1975.

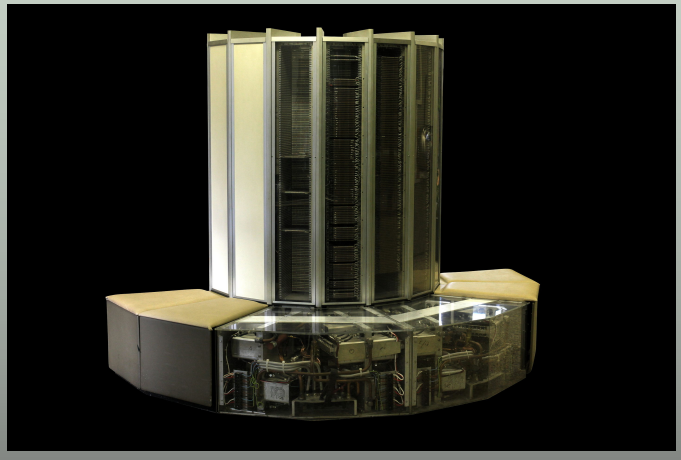
160 Mflops of capacity

Now: Nexust 10 or HTC performs 1GFlops (close to 8 times)

First Vector Processor, which can operate over one-dimensional arrays, called vectors.

The GPUs are the evolution of this Vector Processors

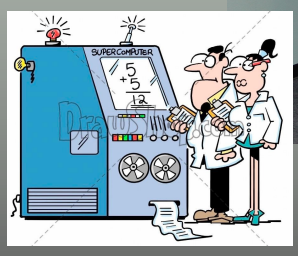
Costs close to 1 Million U$ (1976)



Harold Molina-Bulla DTSC-UC3M

HISTORY OF HIGH PERFORMANCE COMPUTATION

- Mainframes still exists.
- Very expensive supercomputers.
- Not scalable
- Medium life span



2022

Harold Molina-Bulla DTSC-UC3M

HISTORY OF HIGH PERFORMANCE COMPUTATION

After Mainframes, appears minicomputers:

- Shared memory/CPU computers with several terminals attached
- Not so expensive (100K U$ in 1982)
- High amout of memory: 1MByte of memory an HDD of 50MBytes removable

Harold Molina-Bulla DTSC-UC3M

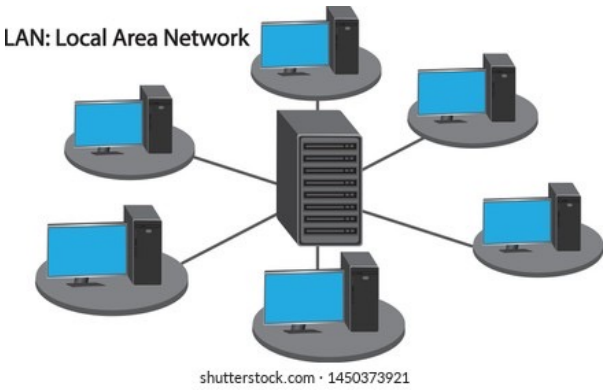
HISTORY OF HIGH PERFORMANCE COMPUTATION

- Workstations:
- Medim size computers
- High amout of memory (512MByte up to 1GByte of mem)
- High speed processors: 256Mhz
- Multitask Operating System
- Affordable Price: 10K U$

2022

HISTORY OF HIGH PERFORMANCE COMPUTATION

- Clusters of Workstations:
- We can buy up to 10 workstations by the price of une minicomputer,
- 10 times more memory (3rd and 4th level)
- Can execute separated segments of code and join at the end.
- Problems: Synchronization and communications



MODERN SUPER-COMPUTERS

Farm of small servers

- Each server can costs close to 5KU$
- High amount of memory (256GBytes ~ 1TByte) per node
- High number of computational cores (48 cores per CPU/ Multi CPU)
- High speed network communications
  - 48 Gbits per second
- Highly scalable: More computational power? Install more servers

MareNostrum Supercomputer in Barcelona
In top 10 fastest computer in the world
The best building: romanic church

Harold Molina-Bulla DTSC-UC3M

COMMON TERMS IN MASSIVE COMPUTATION

Harold Molina-Bulla DTSC-UC3M

COMMON TERMS

- Multitask:
  - The hability to execute several tasks at the same time, not necessary simultaneously
    - Example: Humans: Read, eats, and listen other people.
      - Sharing same resources,
      - Lost attention to ther tasks etc.
      - Jump from one task to other
- Multiprocess
  - The hability to execute several tasks at same time simultaneausly
  - This is because it have several processors, each one execute one task in parallel
  - All multiprocess systems are multitasks but not all multitasks are multiprocess

2022

COMMON TERMS

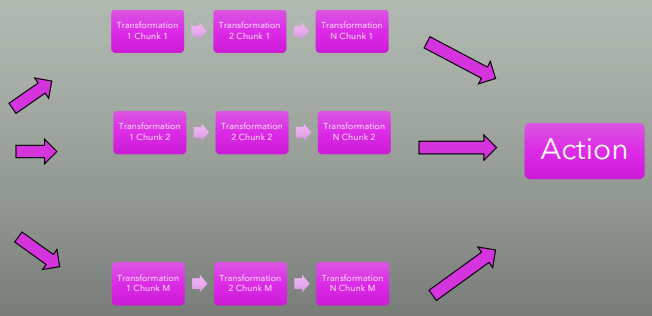
- Multithread
  - Able to execute in parallel parts of a process, but share the main core with other threads
    - Question: Which CPU have your computer?
    - How many cores and threads?
  - When ona thread have to wait for data, the other threads take the control of the CPU
  - But if una thread never have to wait for data??? THIS IS THE MASSIVE COMPUTING CASE

PROCESS LIMITED BY

- I/O BOUND
  - Process limited by I/O
  - E.gr: Word processing. Limited by the speed of click keys (human speed)
  - The CPU is sleeping all the time (except if its MS Word).
  - The process have to wait all the time to have data from I/O ports
  - If you use too many memory, it have to swap to disk -> performance is degraded
- CPU Bound
  - Processes which all the time uses the CPU, exept when they have to upload data from the memory (3rd or 4th level)
  - Which are the memory of 1st and 2nd level?
    - Inside the processors
      - 1st level: CPU registers. Fastest access memory, but very limited amount
      - 2nd level: L2 cache memory. More memory. Local memory, the the access is very, very high, but not like registers
      - If the data is not in the L2 cache, causes a hit-memory, and have to download the actual memory, and load from L3 level memory

PARALLEL EXECUTION

- In science, there are many calculae problems, which can be splited, separate different sub-calculae and after join to get the final solution



Harold Molina-Bulla DTSC-UC3M

PARALLEL EXECUTION

- Requeriments:
  - Each separated section MUST be independent of the other ones.
  - Sometimes we have to collect, centralize and execute in a linear way several calculae
  - Some tasks CAN NOT BE parallelized
    - Fibbonacci sequence: each data depends of the two previous:

$$fib(n) = fib(n-1) + fib(n-2)$$

  - Others are easy paralleizable:
    - Count then number of students in each classroom at same time.
  - Other are partially parallelizable:
    - Count the number of words in all the Shakespeare works
    - Calculate the dot product between 2 vectors

2022

PROBLEMS WITH PARALLEL EXECUTION

- Kinder math problem:
  - If one worker build a house in 1 year, how many time takes to build 30 houses?
  - Now, how many workers we need to build 30 houses in 1 year????

PROBLEM WITH PARALLEL EXECUTIONS

- Kinder Answer: 30 workers
- Real world: with 30 workers will take MORE tha one year to build 30 houses
  - Why????
    - The ammount of bricks are limited. If they ends, all will have to wat until the new comes
    - The officer who distribute the bricks only will attends one builder at time
    - You have a limited number of concrete machines
    - Etc…
  - The problem in this case is: You have a limited number of resources available and you will have to share with other processes. You will have to wait to access to shared resources

PROBLEMS WITH PARALLEL EXECUTION

- More kinder questions:
  - If you wants to finish your 30 houses in ½ year? How many workers will you have to hire?

Harold Molina-Bulla DTSC-UC3M

PROBLEMS WITH PARALLEL EXECUTION

- Kinder answer: 60 workers
- Cruel reality: more than 60 workers… Why?
  - Even you assign more than 2 workers per house, some task will need from previous one: Second floor must be builded after the first floor and after the basement.
  - Before walls you need to install pipes (electricity, water, gas, etc…)
  - Without walls, you cannot install electrical wires.
  - Only at end you paint walls.
- Those pre-conditions limits the optimal parallelization of tasks

HALF EMPTY GLASS

- The Amdahl's law is a formula which gives the theoretical speedup in latency of the execution of a task at fixed workload that can be expected of a system whose resources are improved.
- $S_{latency}(s) = \frac{1}{(1-p)+\frac{p}{s}}$
- Where:
  - $S_{latency}$ is the theoretical speedup of the execution of the whole task
  - s is the speedup of the part of the task thant benefits from parallel resources
  - p is the portion of execution time that the part benefiting from parallel resources originally occupied
- This law shows, even you have infinite resources, your task will never achive the best performance.

GUSTAFSON'S LAW

- This is the possitive point of view:
  - $S_{latency}(s) = 1 - p + sp$
- Where
  - $S_{latency}$ is the theoretical speedup of the execution of the whole task
  - s is the speedup of the part of the task thant benefits from parallel resources
  - p is the portion of execution time that the part benefiting from parallel resources originally occupied
- The Gustafson law propose that the programmers tend to set the size of problems to fully exploit the computing power.
- It shows always you will improve your task using parallel resources.

Harold Molina-Bulla DTSC-UC3M

HAZZARDS IN PARALLEL ENVIRONMENTS

- When you program to execute in parallel enviroment, you will have 3 well know hazards with your data:
  - READ AFTER WRITE: When one process neds to read a memory possition,and meanwhle other process write that possition, the data could be corrupted, and the results will be unknown
  - WRITE AFTER READ: IF a portion of program needs to read the updated version of a memory possition, and be read BEFORE the other process update them, the results will be unknown
  - WRITE AFTER WRITE: With two process write the same memory possition, one after the other, the results will be corrupted for one of them, and the final result will be unknown.