Part 1: Hardware Systems

CSCI-4850/5850 High-Performance Computing

Summer 2021 at GIST

Tae-Hyuk (Ted) Ahn
Department of Computer Science
Program of Bioinformatics and Computational Biology
Saint Louis University

SAINT LOUIS
UNIVERSITY
EST. 1818

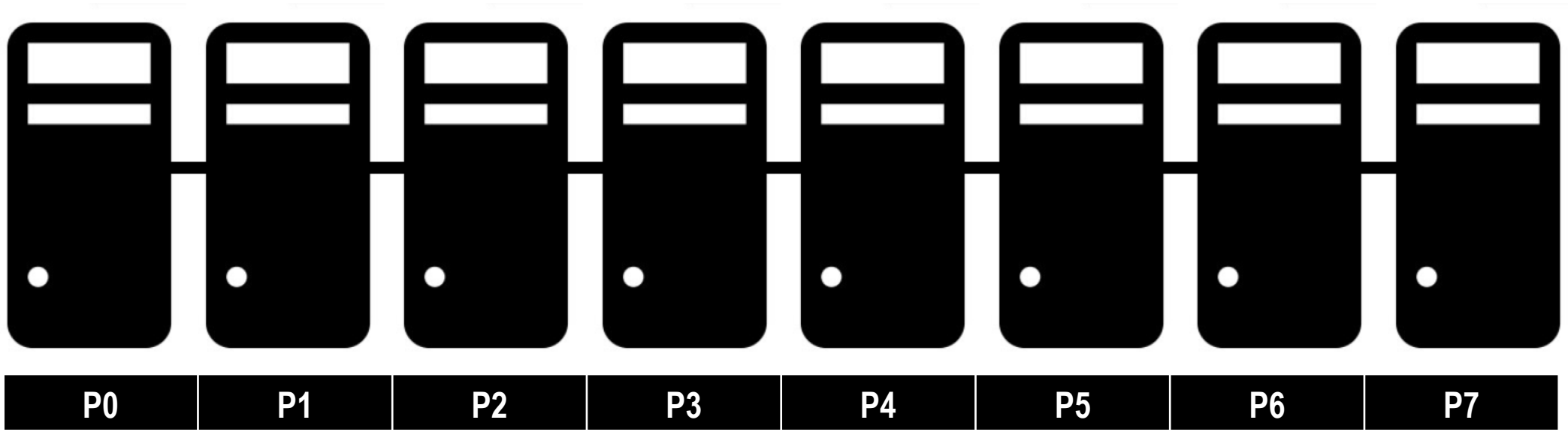
Traditional Parallel Computing Fundamentals

- In parallel computing, many computers, each with multiple processors, are interconnected.
- The individual processors can work simultaneously to solve a smaller part of a complex problem that is too large for a single computer.

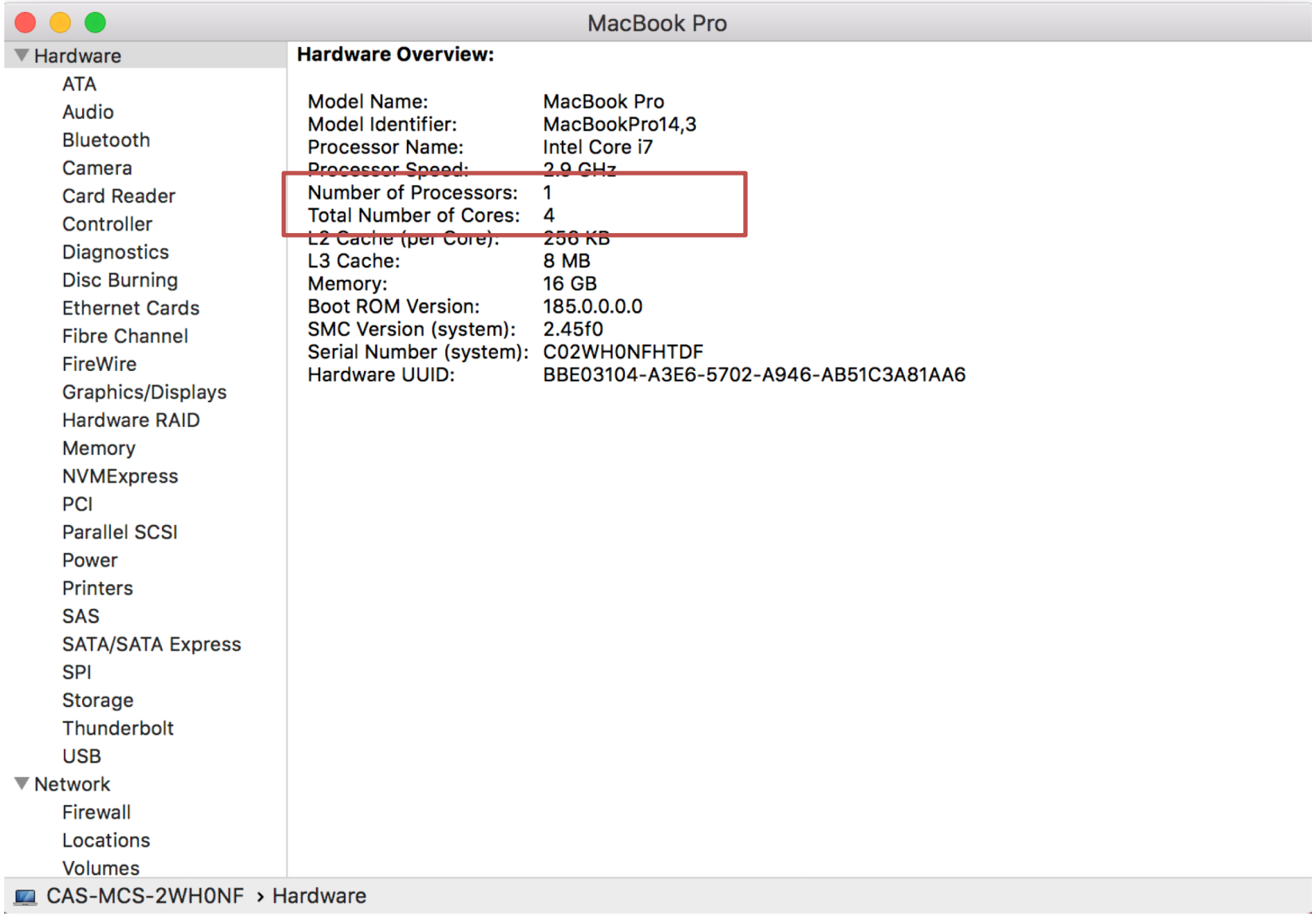
Generic Parallel Machine

- Good conceptual model is collection of processors

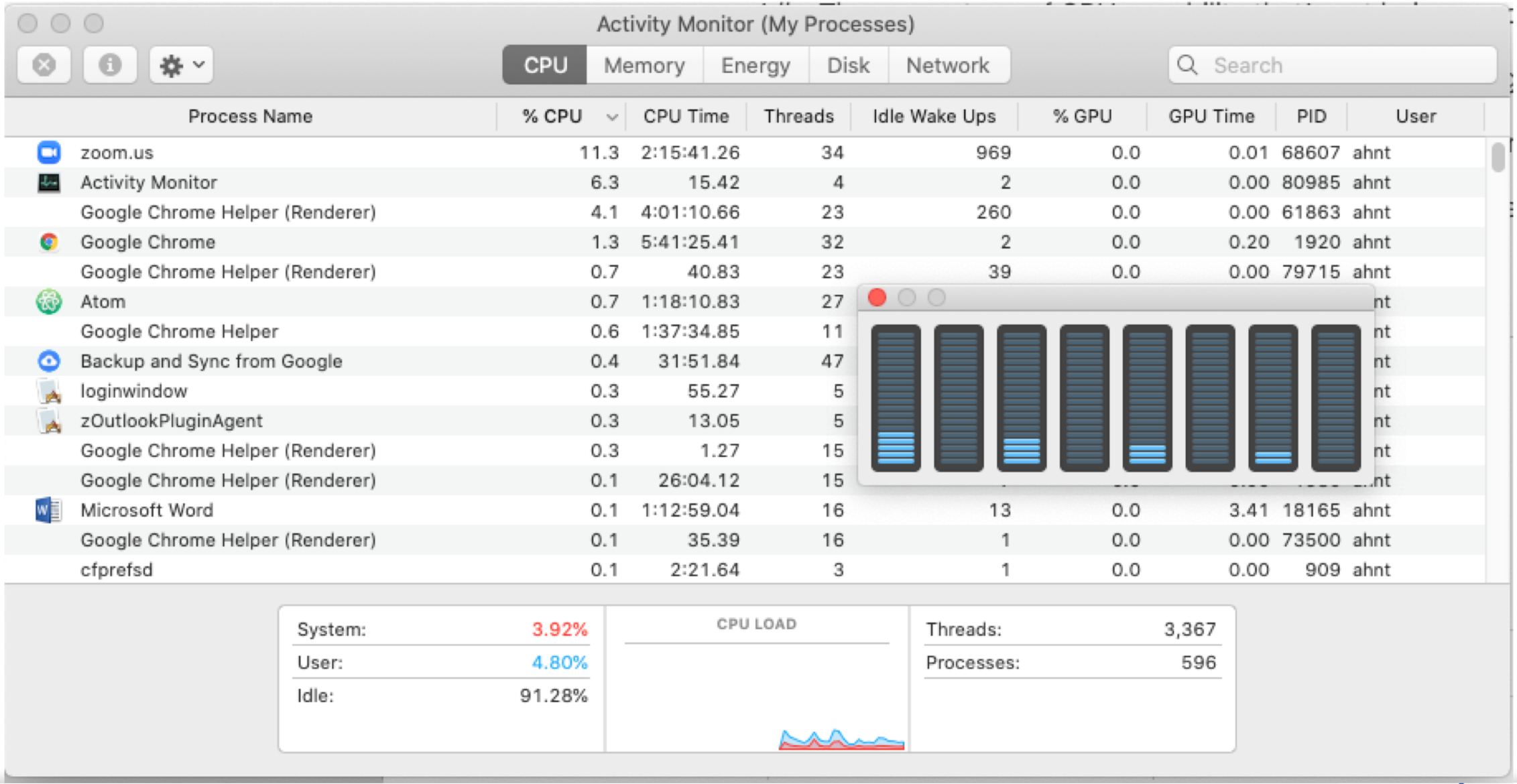
  => A processor is a generic term used to describe any sort of CPU, regardless of cores.



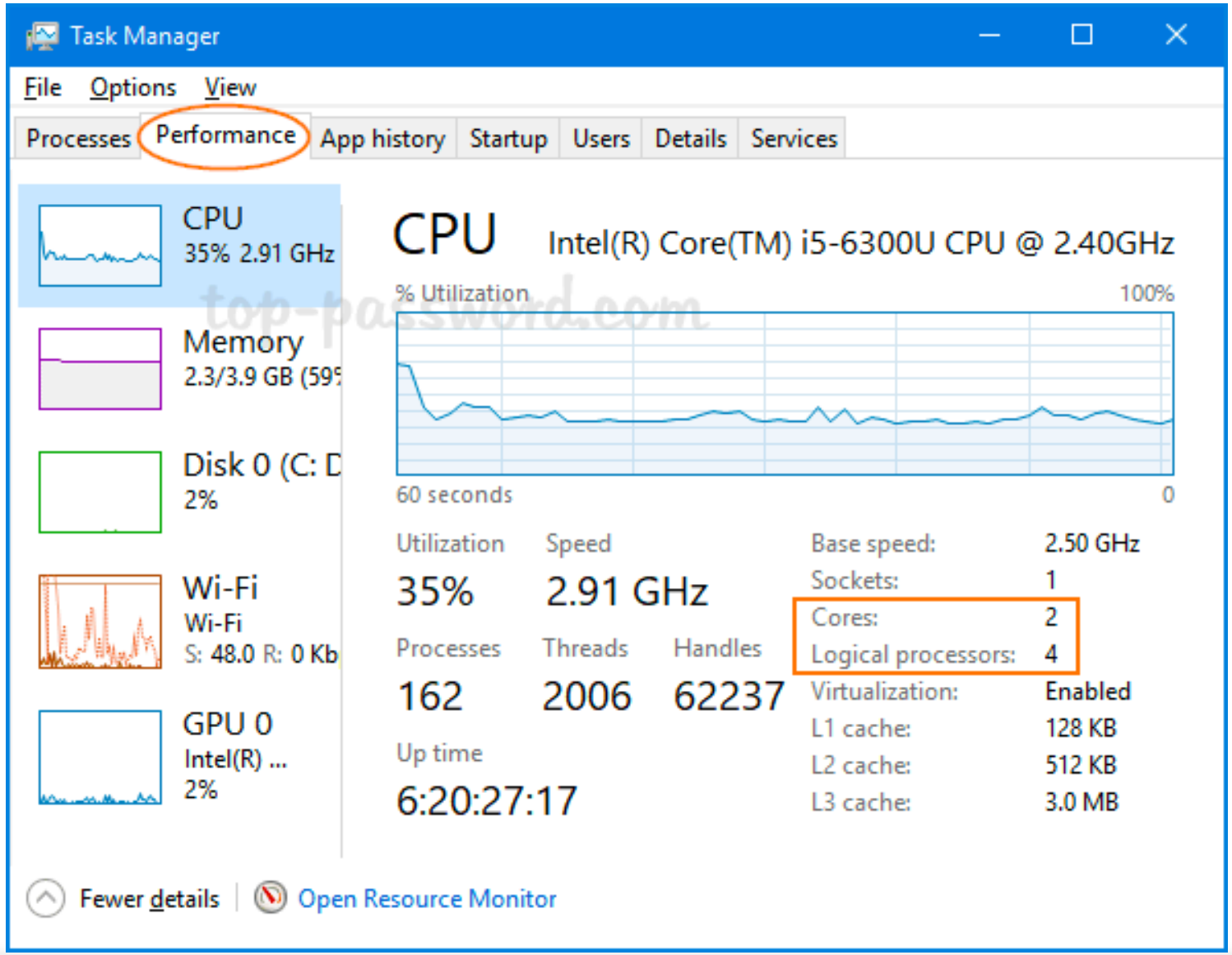
Find Number of Cores in your CPU



Find Number of Cores in your Mac and see CPU Usage



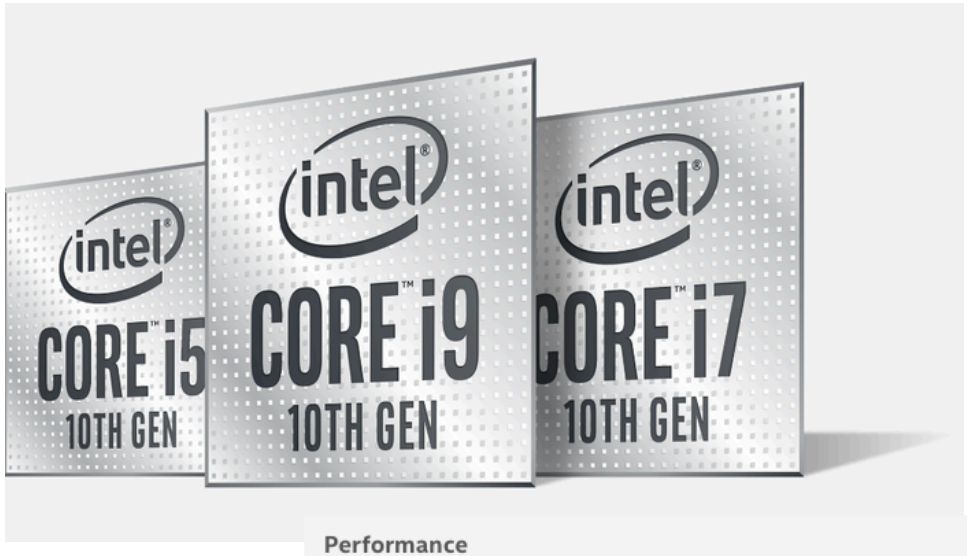
Find Number of Cores in your Windows and see CPU Usage



Core, Thread, Hyper Threading, Turbo Boost, K-Model

- Cores
  - A core is a single computing component with independent CPU.
  - A core is defined as a CPU which read and execute program instructions.
  - Number of cores directly affect processing speed of Device. The more number of cores you have the faster Processing you get.
- Process Thread
  - It is just a portion of a Process that CPU currently executing.
  - Larger programs are divided into threads (small sections) so that processor can execute them simultaneously to get fast execution.
- Hyper Threading
  - It means that the CPU has 2 physical cores but can process 4 threads simultaneously through hyper threading.
  - In reality, one physical core can only truly run one thread at a time, but using hyper threading, the CPU exploits the idle stages in the pipeline to process another thread.

Check Tech Specs

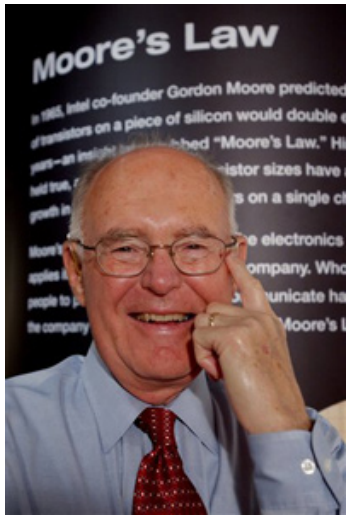


Credit: Intel

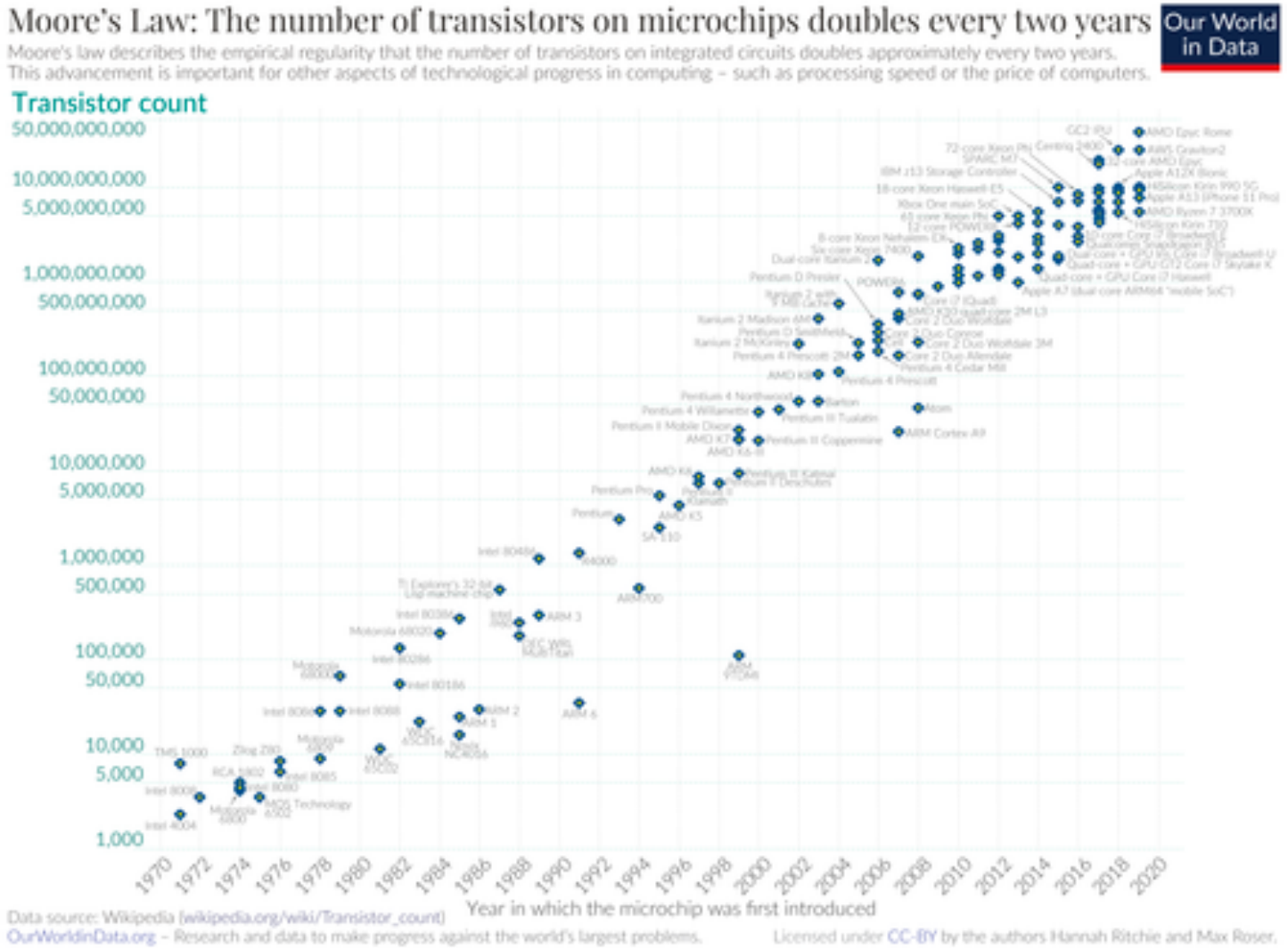
Performance

Intel® Thermal Velocity Boost Frequency	5.20 GHz	# of Cores	10
# of Threads	20	Processor Base Frequency	3.60 GHz
Max Turbo Frequency	5.20 GHz	Cache	20 MB Intel® Smart Cache
Bus Speed	8 GT/s	Intel® Turbo Boost Max Technology 3.0 Frequency ‡	5.10 GHz
TDP	125 W	Configurable TDP-down Frequency	3.30 GHz
Configurable TDP-down	95 W		

Moore's Law

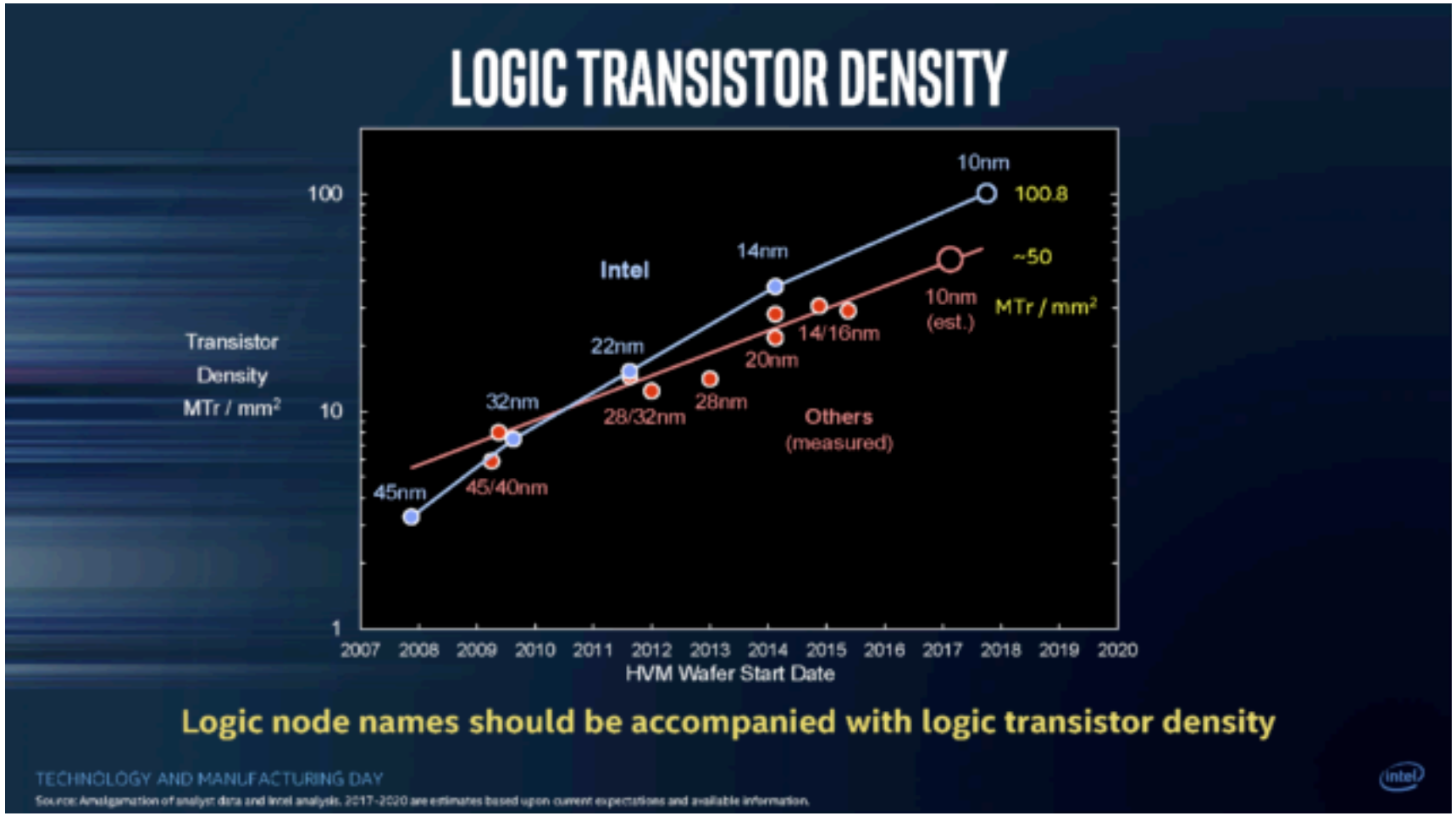


- An observation made in 1965 by Intel co-founder:
  - The number of transistors in a dense integrated circuit doubles approximately every two years.
  - The observation is named after Gordon Moore, the co-founder of Fairchild Semiconductor and CEO and co-founder of Intel.
  - It has been used as a standard in semiconductor industry (a self-fulfilling prophecy).



Intel is keeping Moore's Law alive

megatransistors per millimeter



5 nanometer

https://en.wikipedia.org/wiki/5_nanometer

Not logged in Talk Contributions Create account Log in

Article Talk | Read Edit View history | Search Wikipedia

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages
Català
Español
فارسی
Français
Italiano

5 nanometer

From Wikipedia, the free encyclopedia

In semiconductor manufacturing, the International Roadmap for Devices and Systems defines the **5 nanometer** (**5 nm**) node as the technology node following the 7 nm node.

Contents [hide]

Semiconductor manufacturing processes

10 μm – 1971
6 μm – 1974
3 μm – 1977
1.5 μm – 1982
1 μm – 1985
800 nm – 1989
600 nm – 1994
350 nm – 1995
250 nm – 1997
180 nm – 1999
130 nm – 2001
90 nm – 2004
65 nm – 2006
45 nm – 2008
32 nm – 2010
22 nm – 2012
14 nm – 2014
10 nm – 2017
7 nm – 2018
5 nm – *~2020*

Half-nodes

V • T • E

History [edit]

Background [edit]

The 5 nm node was once assumed by some experts to be the end of Moore's law.[1] Transistors smaller than 7 nm will experience quantum tunnelling through the gate oxide layer.[2] Due to the costs involved in development, 5 nm is predicted to take longer to reach market than the two years estimated by Moore's law.[1]

Beyond 7 nm, it was initially claimed that major technological advances would have to be made to produce chips at this small scale.[*citation needed*] In particular, it is believed that 5 nm may usher in the successor to the FinFET, such as a gate-all-around architecture.

Technology demos [edit]

Single transistor 7 nm scale devices were first produced by researchers in the early 2000s.

In 2002, IBM produced a 6 nm transistor.[3]

In 2003, NEC produced a 5 nm transistor.[4]

In 2015, IMEC and Cadence had fabricated 5 nm test chips. The fabricated test chips are not fully functional devices but rather are to evaluate patterning of interconnect layers.[5][6]

In 2015, Intel described a lateral nanowire (or gate-all-around) FET concept for the 5 nm node.[7]

In 2017, IBM revealed that they had created 5 nm silicon chips,[8] using silicon nanosheets in a *gate-all-around* configuration (GAAFET), a break from the usual FinFET design.[9]

What Is Clock Speed?

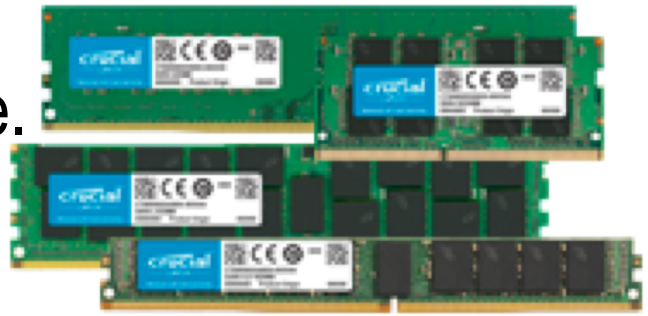
- In general, a higher clock speed means a faster CPU. However, many other factors come into play.
- Your CPU processes many instructions (low-level calculations like arithmetic) from different programs every second. The clock speed measures the number of cycles your CPU executes per second, measured in GHz (gigahertz).
- A "cycle" is technically a pulse synchronized by an internal oscillator, but for our purposes, they're a basic unit that helps understand a CPU's speed. During each cycle, billions of transistors within the processor open and close.
- A CPU with a clock speed of 3.2 GHz executes 3.2 billion cycles per second. (Older CPUs had speeds measured in megahertz, or millions of cycles per second.)

Why haven't CPU clock speed increased in the last few years?

- Why do we see so few Intel® processors over 5 GHz in the last few years?
- The reason is heat dissipation and power consumption.
- The faster we switch the transistors on and off, the more heat will be generated. Without proper cooling, they might fail and be destroyed.
- A CPU with a higher clock speed from five years ago might be outperformed by a new CPU with a lower clock speed, as the newer architecture deals with instructions more efficiently. An X-series Intel® processor might outperform a K-series processor with a higher clock speed, because it splits tasks between more cores and features a larger CPU cache.
- But within the same generation of CPUs, a processor with a higher clock speed will generally outperform a processor with a lower clock speed across many applications.

Memory

- Computer random access memory (RAM) is one of the most important components in determining your system's performance.
- RAM gives applications a place to store and access data on a short-term basis.



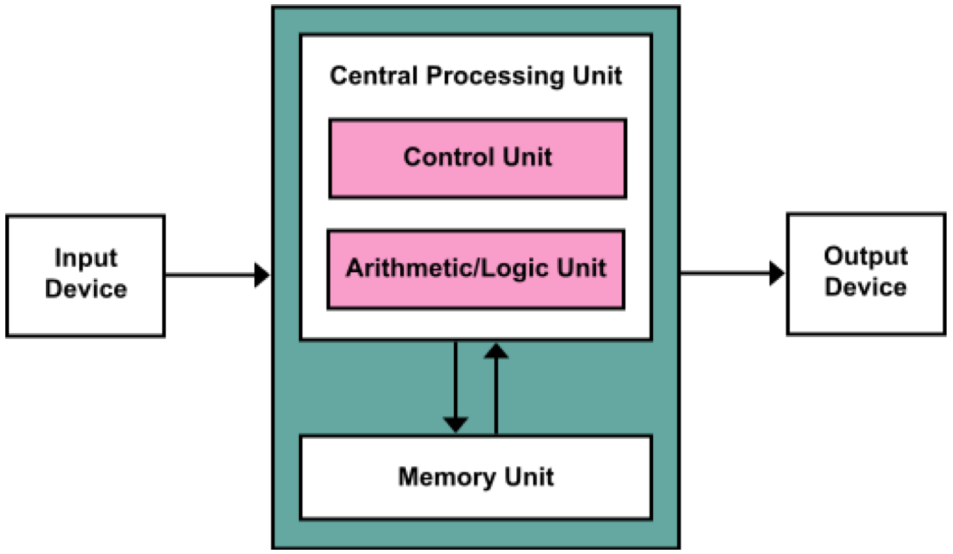
- It stores the information your computer is actively using so that it can be accessed quickly.
- The more programs your system is running, the more you'll need.
- SSDs (solid state drives) are also important components and will help your system reach its peak performance.

The Difference Between RAM Speed and CAS Latency

- Latency: At a basic level, latency refers to the time delay between when a command is entered and when the data is available.
- CAS latency, also known as "Column Access Strobe." This is the number of clock cycles that pass between when an instruction is given and when the information is made available.
- Read: https://www.crucial.com/articles/about-memory/difference-between-speed-and-latency

The von Neumann Architecture

- The “classical” von Neumann architecture consists of main memory, a central processing unit (CPU) and an interconnection between the memory and the CPU.



www.wikipedia.org/

- Arithmetic/Logic unit (ALU): executes the instruction.
- Main memory consists of a collection of locations, each of which is capable of storing both instructions and data.
- Every location consists of an address, which is used to access the location and the contents of the location—the instructions or data stored in the location.

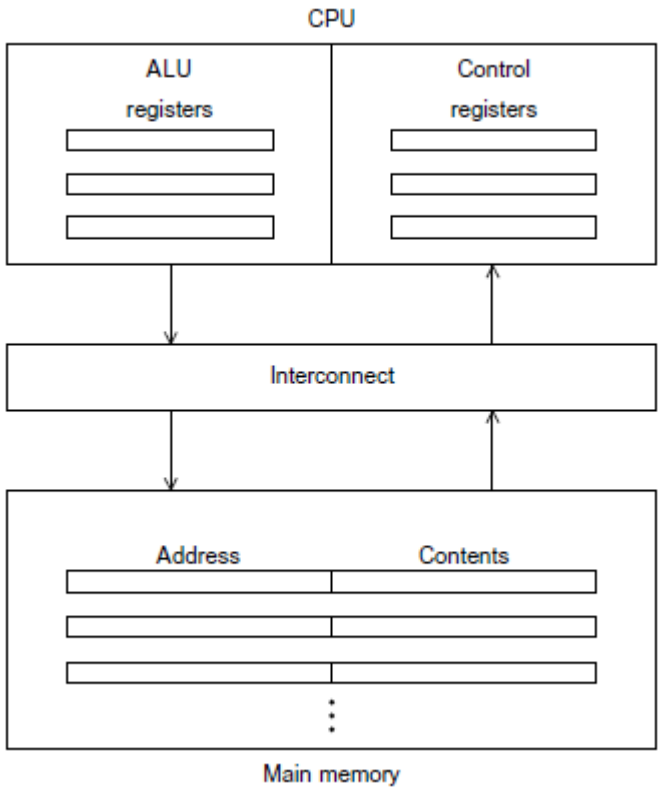
John von Neumann

- Hungarian-American mathematician (1903-1957).
- Contributions to mathematics, economics, computer science, and statistics.
- Member of Manhattan Project and Institute for Advanced Study.
- Proposed a design for a digital computer (EDVAC) in 1945 that later became the von Neumann model.
- Introduced cellular automata.
- Designed merge sort algorithm.

www.wikipedia.org

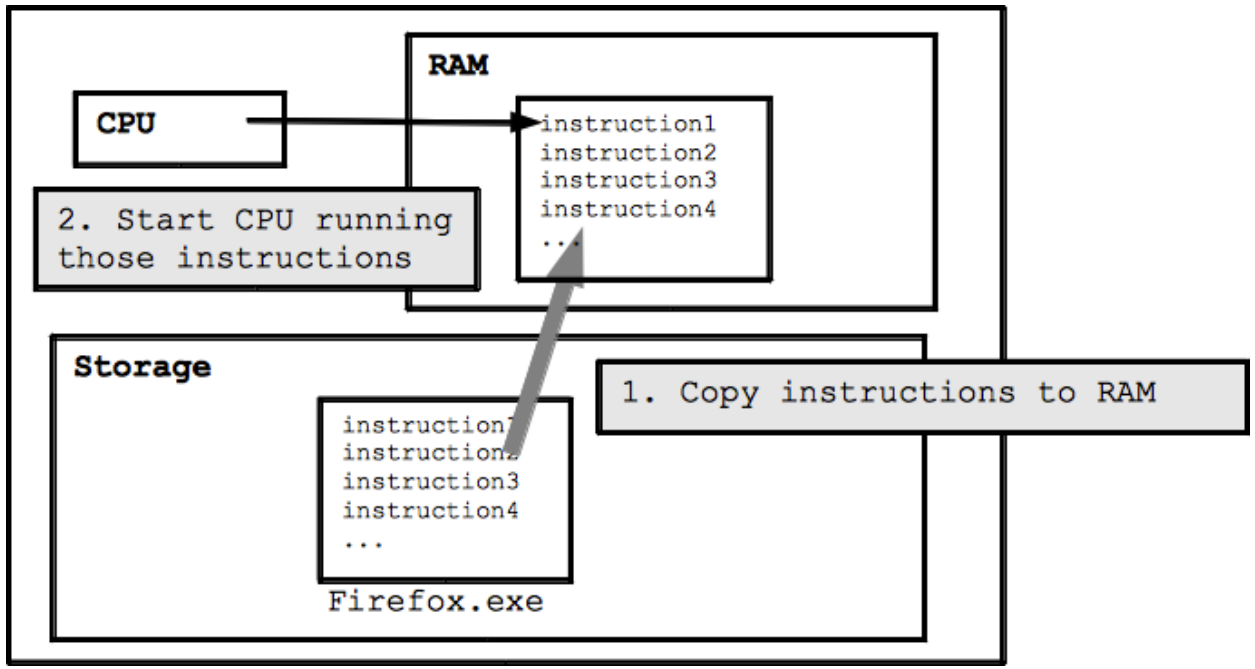
The von Neumann Architecture

- The control unit is responsible for deciding which instructions in a program should be executed.
- The **ALU** is responsible for executing the actual instructions.
- Data in the CPU and information about the state of an executing program are stored in special, very fast storage called **registers.**
- Instructions and data are transferred between the CPU and memory via the interconnect, the bus.



- When data or instructions are transferred from memory to the CPU, we sometimes say the data or instructions are **fetched** or **read** from memory

Instruction

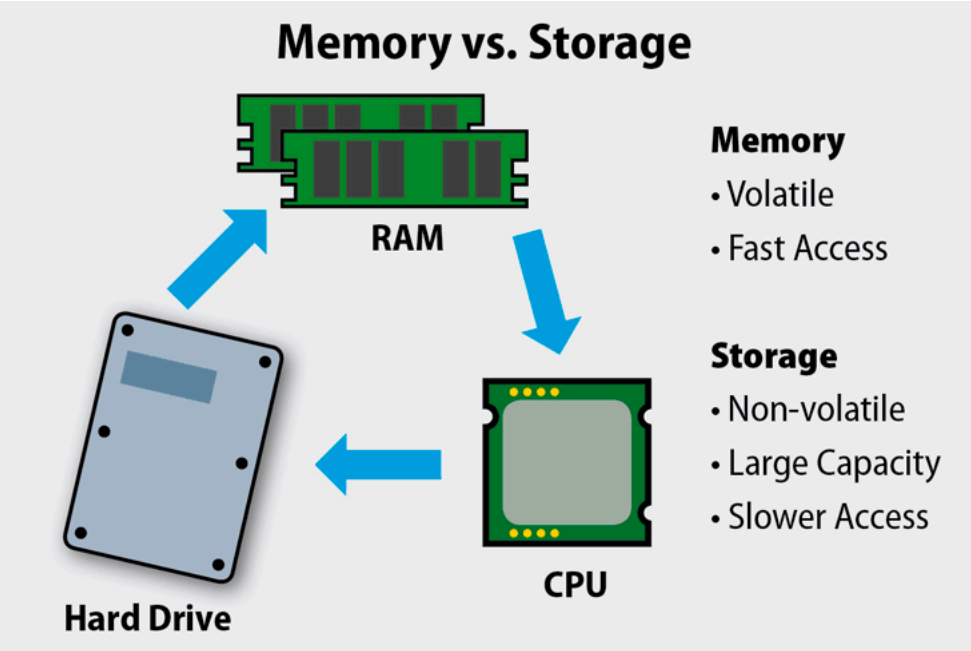


- An instruction has go through the following stages:
  - Instruction Fetch (IF)
  - Instruction Decode (ID)
  - Instruction Execution (EX)
  - Memory Read/Write (MEM)
  - Result Writeback (WB)

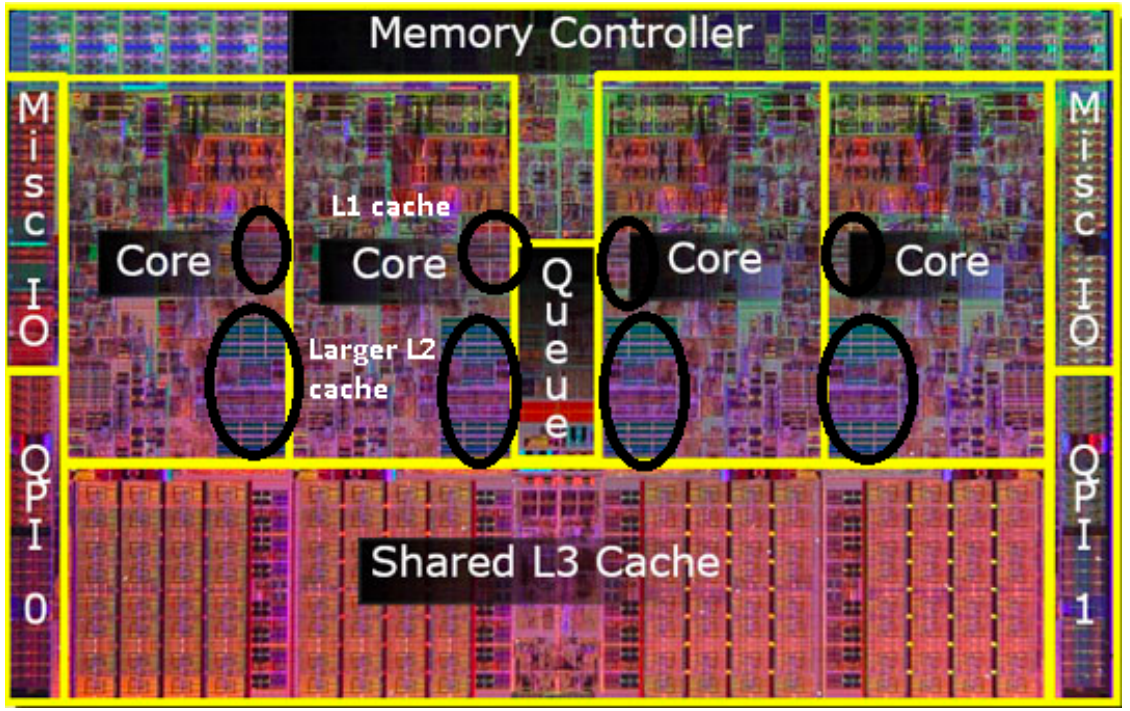
- Let us assume each stage takes one clock cycle.

- CPU instructions are numbers stored in memory.
- The instructions are specific to the CPU architectures.
- Basic operation: read instruction from memory, decode/crack/crunch it, execute, write-back, advance.
- OS and Computer Architecture cover this topic

Cache



https://www.enterprisestorageforum.com/storage-hardware/memory-vs-storage.html



https://superuser.com/questions/196143/where-exactly-l1-l2-and-l3-caches-located-in-computer

- Cache is a smaller and fast memory component in the computer which is inserted between the CPU and the main memory.

Caching

- A faster and smaller memory that only stores the most popular values.

- Principle of locality:
  - Spatial: if value X is accessed, its neighboring values will be likely to be accessed in the near future.
  - Temporal: if value X is accessed, it will be likely to be accessed in the near future.

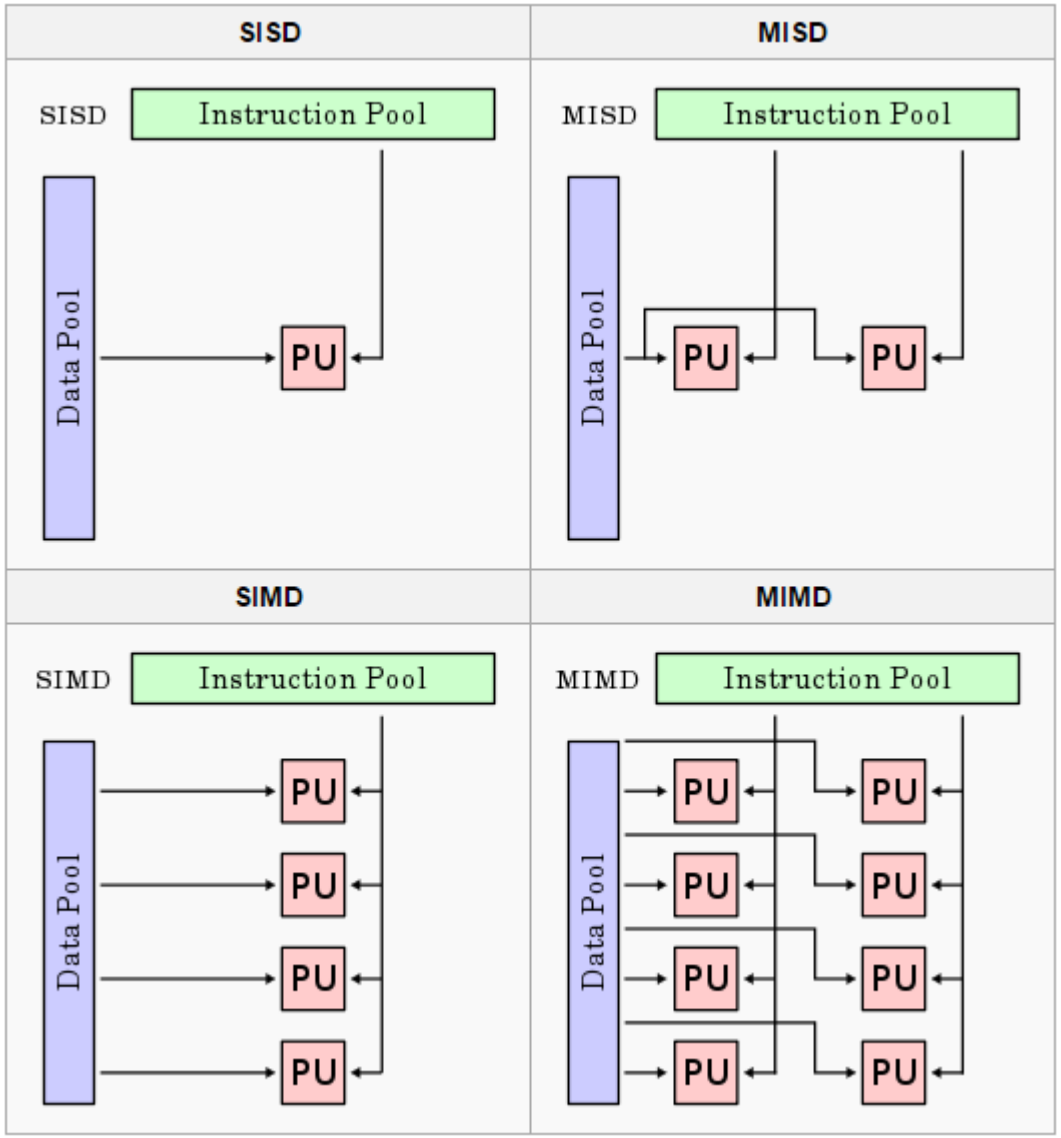
Flynn's Taxonomy

- First proposed by Michael J. Flynn in 1966, Flynn's taxonomy is a specific classification of parallel computer architectures that are based on the number of concurrent instruction (single or multiple) and data streams (single or multiple) available in the architecture. The four categories in Flynn's taxonomy are the following:
  - (SISD) single instruction, single data
  - (SIMD) single instruction, multiple data
  - (MISD) multiple instruction, single data
  - (MIMD) multiple instruction, multiple data

New Additional Concept:

- (SIMT) Single instruction, multiple threads

Flynn's Taxonomy



- SISD
  - Typical thread (Single-core processor)
- SIMD
  - Vector processors
  - GPUs
- MISD
  - For fault tolerance
- MIMD
  - Multi-core processor
  - Cluster of computers
  - Supercomputers

en.wikipedia.org

SIMD Architecture

SIMD Architecture

- SIMD architecture consists of a single CPU used for control; and a separate set of computational units (microprocessor chips), each with its own memory.
- To initiate a computation, the control processor broadcasts a single instruction to each computational unit that either executes the instruction or remains idle.
- Which computational units do or do not execute a given instruction will often depend on one or more conditional statements within the program.
- Consequently, one primary disadvantage of SIMD architectures is that many of the computational units can remain idle for extensive periods of time if a program has a large number of conditional branches or contains large sections of code whose execution depends on one or more conditional statements.

MIMD Architectures

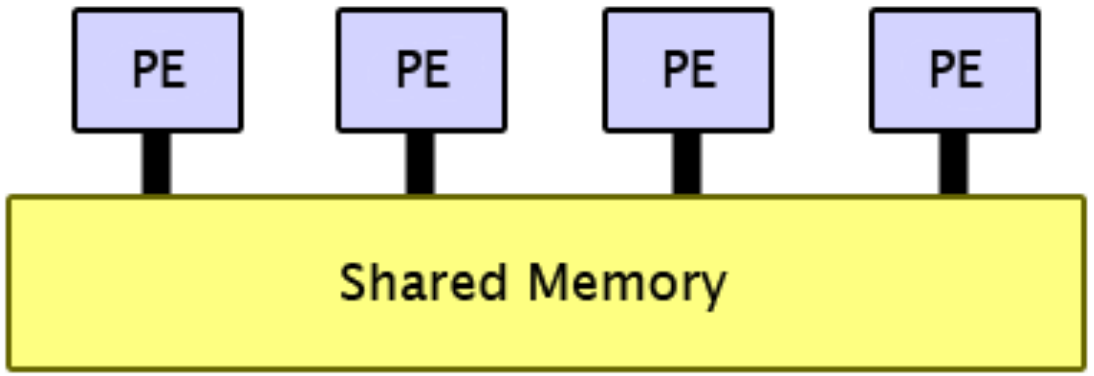
- In contrast to SIMD, each processor in a MIMD architecture has both a control unit and a computational unit. Consequently, each processor executes its own subprogram at a rate independent of all other processors within the system. That is, MIMD architectures are asynchronous.
- There are two main types of MIMD architecture – shared-memory and distributed-memory. Also, there is an architecture called shared memory processing (SMP) that is a combination of shared-memory and distributed-memory MIMD architectures.

Shared Memory System vs Distributed Memory System

- Shared Memory System vs Distributed Memory System

Conceptual Shared Memory System

- All processors share a single address space.
- Communication is implicit: write and read operations on shared variables.
- Simple programming model: no data distribution among processors.
- Limited scalability (memory contention).



Shared Memory System: From Desktop to Supercomputer

NERSC Cori ExVivo (https://docs.nersc.gov/science-partners/jgi/cori-exvivo/)

- 2 Intel® Xeon® Gold 6140 (Skylake) processors, 36 cores total
- 1.5 TB RAM
- 5.1 TB available local disk, solid state drive, mounted as /tmp

Numascale, Supermicro, and AMD Announce the World's Largest Shared Memory System to Date

HPC Today | Wire | November 20, 2014

A A A

Note: the wired news below has been filtered but not edited by HPC Today.

Numascale, Supermicro, and AMD Announce the World's Largest Shared Memory System to Date

System features 5184 cores and 20.7 TBytes of shared memory, and handles massive amounts of data for Big Data Analytics.

HPE unveils 'world's largest' single memory computer

16 May 2017 Share

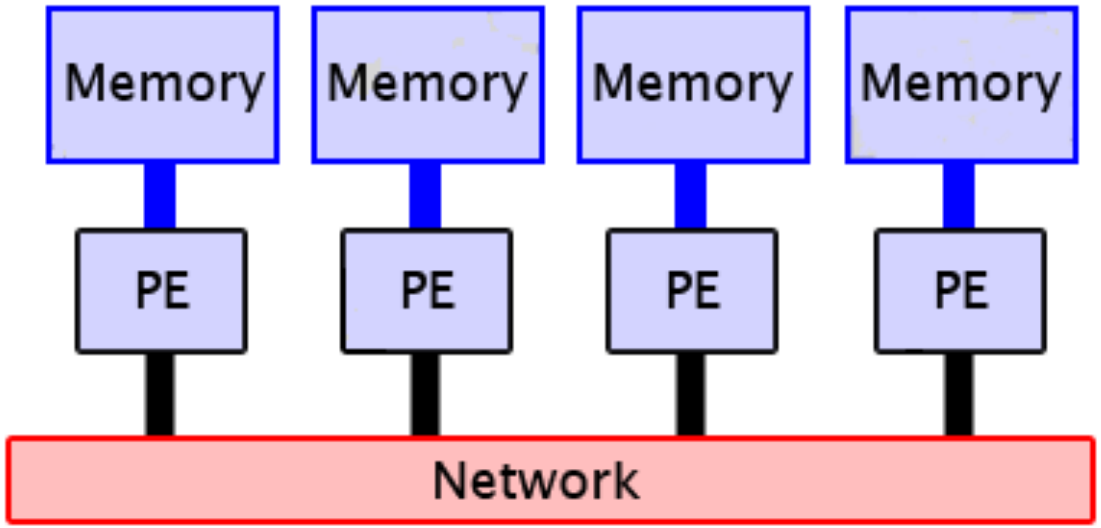
A prototype computer with 160TB of memory has been unveiled by Hewlett Packard Enterprises.

Designed to work on big data, it could analyse the equivalent of 160 million books at the same time, HPE said.

The device, called The Machine, had a Linux-based operating system and prioritised memory rather than processing power, the company said.

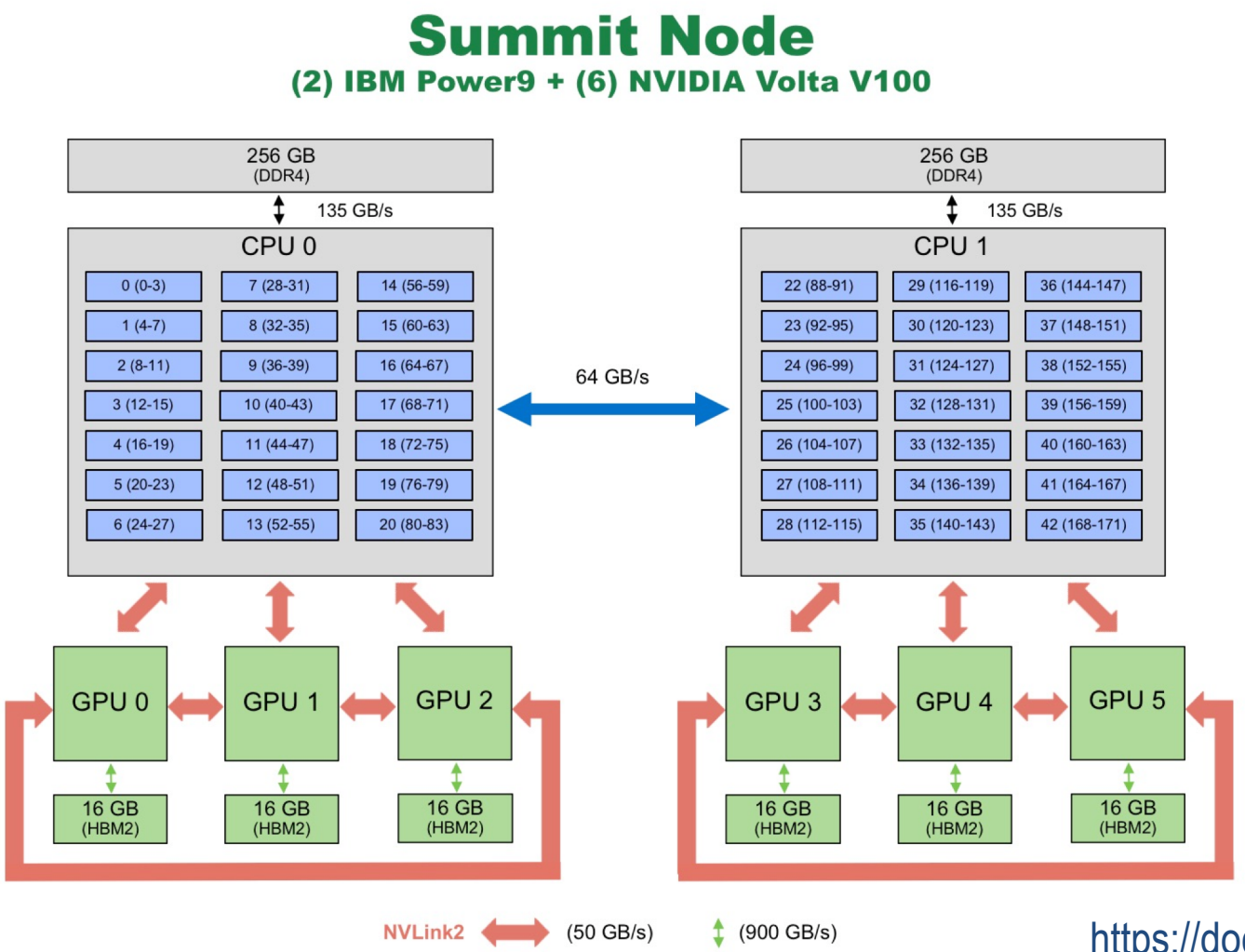
Conceptual Distributed Memory System

- Each processor has its own private memory.
- Communication is explicit through message passing.
- Involved programming model: data distribution.
- Good scalability.



ORNL Summit Node

https://www.olcf.ornl.gov/calendar/programming-methods-for-summits-multi-gpu-nodes/



https://docs.olcf.ornl.gov/systems/summit_user_guide.html

Summit Overview

OpenPOWER™

Compute System

10.2 PB Total Memory
256 compute racks
4,608 compute nodes
Mellanox EDR IB fabric
200 PFLOPS
~13 MW

Compute Rack

18 Compute Servers
Warm water (70°F direct-cooled components)
RDHX for air-cooled components

39.7 TB Memory/rack
55 KW max power/rack

Compute Node

2 x POWER9
6 x NVIDIA GV100
NVMe-compatible PCIe 1600 GB SSD

25 GB/s EDR IB- (2 ports)
512 GB DRAM- (DDR4)
96 GB HBM- (3D Stacked)
Coherent Shared Memory

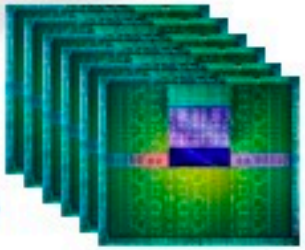
Components

IBM POWER9
- 22 Cores
- 4 Threads/core
- NVLink

GPFS File System

250 PB storage
2.5 TB/s read, 2.5 TB/s write

NVIDIA GV100
- 7 TF
- 16 GB @ 0.9 TB/s
- NVLink



https://www.olcf.ornl.gov/wp-content/uploads/2018/02/SummitJobLaunch.pdf

Summit Node

- Each node is Shared? Distributed?

➔ Distributed/shared memory hybrids

Summit one node has 2 Nonuniform Memory Access (NUMA) nodes.

V100 GPU

- Each Summit Compute node has 6 NVIDIA V100 GPUs.
- The NVIDIA Tesla V100 accelerator has a peak performance of 7.8 TFLOP/s (double-precision) and contributes to a majority of the computational work performed on Summit.
- Each V100 contains 80 streaming multiprocessors (SMs), 16 GB (32 GB on high-memory nodes) of high-bandwidth memory (HBM2), and a 6 MB L2 cache that is available to the SMs.
- The GigaThread Engine is responsible for distributing work among the SMs and (8) 512-bit memory controllers control access to the 16 GB (32 GB on high-memory nodes) of HBM2 memory.
- The V100 uses NVIDIA's NVLink interconnect to pass data between GPUs as well as from CPU-to-GPU. We provide a more in-depth look into the NVIDIA Tesla V100 later in the Summit Guide.

https://docs.olcf.ornl.gov/systems/summit_user_guide.html#nvidia-tesla-v100

V100 GPU


PCI Express 3.0 Host Interface
GigaThread Engine
GPC
TPC
SM
Memory Controller
HBM2
L2 Cache
High-Speed Hub
NVLink

V100 GPU SM

- Each SM on the V100 contains 32 FP64 (double-precision) cores, 64 FP32 (single-precision) cores, 64 INT32 cores, and 8 tensor cores.
- A 128-KB combined memory block for shared memory and L1 cache can be configured to allow up to 96 KB of shared memory.
- In addition, each SM has 4 texture units which use the (configured size of the) L1 cache.

Parallel Systems

- Concurrent programming languages, libraries, APIs, and parallel programming models (such as algorithmic skeletons) have been created for programming parallel computers.
- These can generally be divided into classes based on the assumptions they make about the underlying memory architecture - shared memory, distributed memory, or shared distributed memory.
  - Shared memory programming languages communicate by manipulating shared memory variables.
  - Distributed memory uses message passing.
  - POSIX Threads and OpenMP are two of most widely used shared memory APIs, whereas Message Passing Interface (MPI) is the most widely used message-passing system API.
  - CUDA and OpenACC for NVIDA accelerators.

Part 2: Parallel Performance Analysis

CSCI-4850/5850 High-Performance Computing

Summer 2021 at GIST

Tae-Hyuk (Ted) Ahn
Department of Computer Science
Program of Bioinformatics and Computational Biology
Saint Louis University

SAINT LOUIS
UNIVERSITY
EST. 1818

Why HPC and Parallel Programming?

Leveraging large-scale compute resources to solve complex problems with large-scale datasets

- Terabytes to petabytes to zettabytes of data
- Results in minutes to hours instead of days or weeks

Single-core processors and small-size memory cannot be made that have enough resource for the simulations needed.

Solution: parallel computing – divide up the work among numerous linked systems

Array Sum

Array Elements

Array Sum

Array Elements

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

```
#include <iostream>
using namespace std;
int main (){
    int arr[] = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};
    int n = 16, sum = 0;
    for(int i = 0; i<n ; i++){
        sum+=arr[i];
    }
    cout<<"The array sum is "<<sum;
    return 0;
}
```

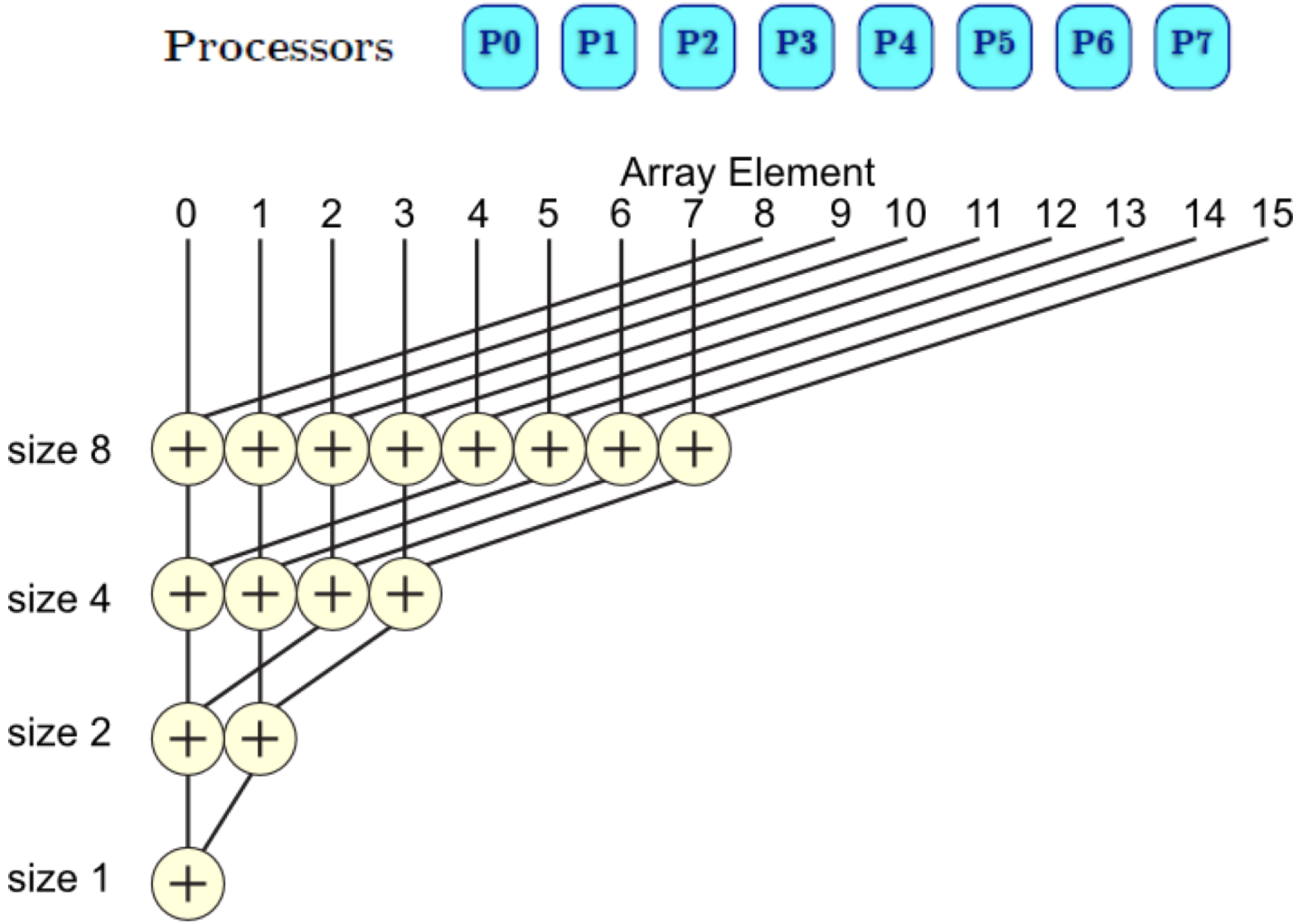
```
The array sum is 120
```

Parallel Array Sum

- A few assumptions
  - Only sum (add) operation takes 1 second and other operations (e.g., assignment) does not take any time.
  - Input/Output does not take any time.
  - Memory access does not take any time.
  - Therefore, the serial code took 16 seconds to get the output.
  - The computer has 8 processors (8 CPUs where each CPU has one core and no hyper-threading).
  - Data transfer among processors does not take any time.

  - How fast (seconds) can you achieve the sum (120) in previous example?

Parallel Sum



Parallel Systems

- Definition: A parallel system consists of an algorithm and the parallel architecture that the algorithm is implemented.
- Note that an algorithm may have different performance on different parallel architecture.
  - For example, an algorithm may perform differently on a linear array of processors and on a hypercube of processors.

Performance Metrices for Parallel Systems

- Run Time: The parallel run time (or wall-clock time) is defined as the time that elapses from the moment that a parallel computation starts to the moment that the last processor finishes execution.

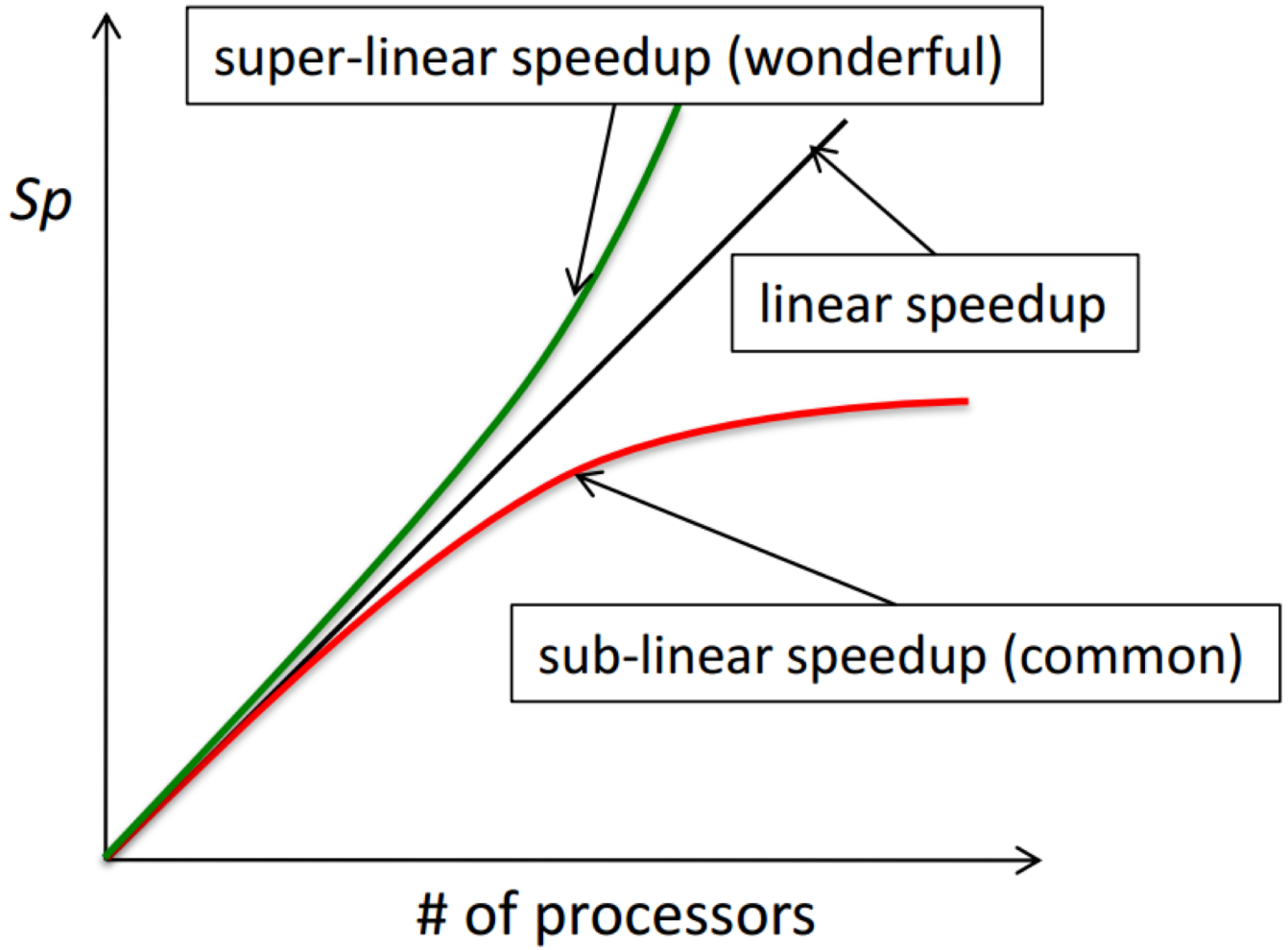
- Notation: Serial run time T_S , parallel run time T_P .

Speedup

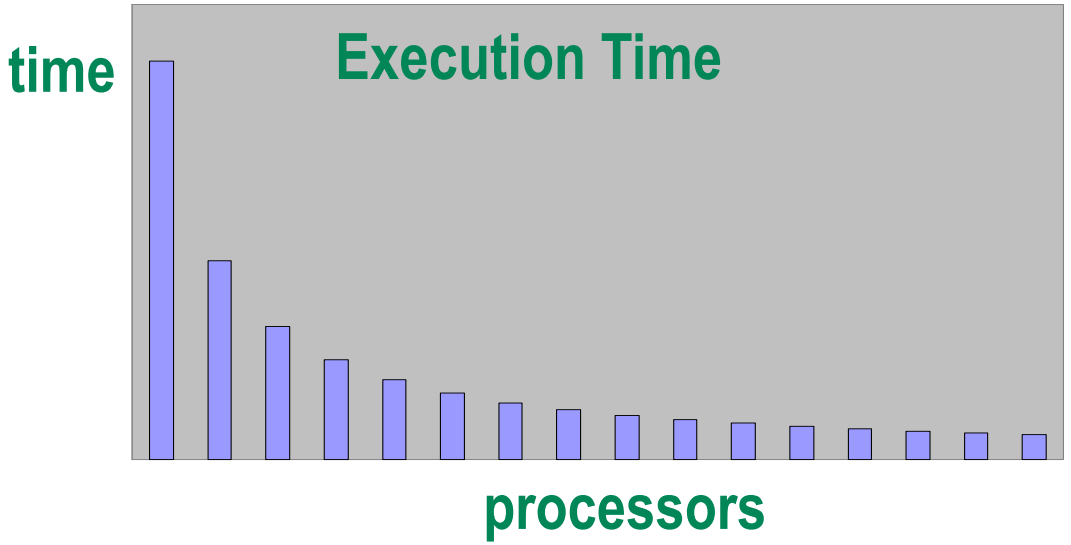
- The **speedup** is defined as the ratio of the serial runtime of the best sequential algorithm for solving a problem to the time taken by the parallel algorithm to solve the same problem on *p* processors.

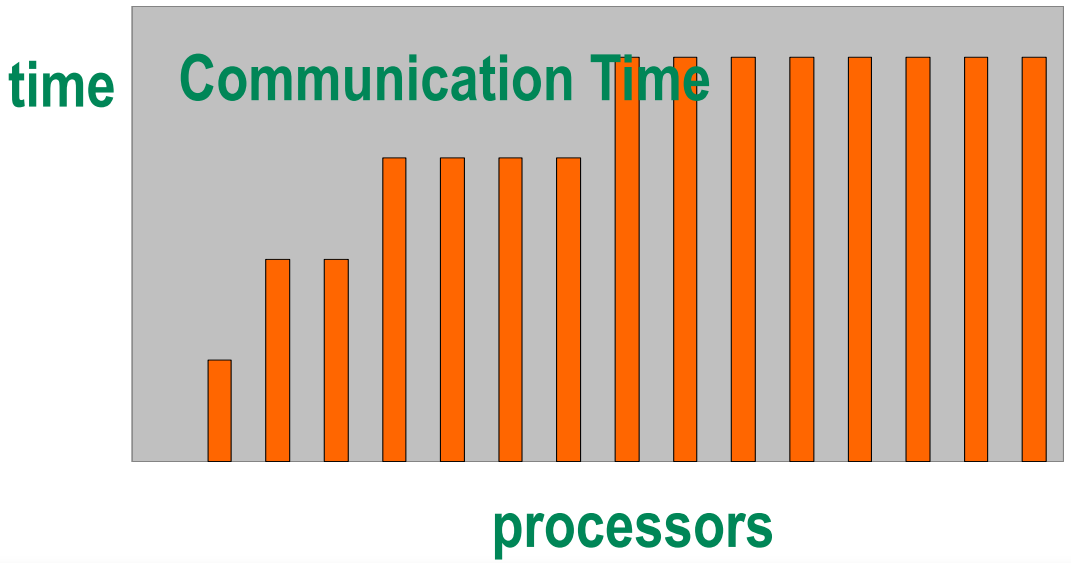
$$S = \frac{T_S}{T_P}$$

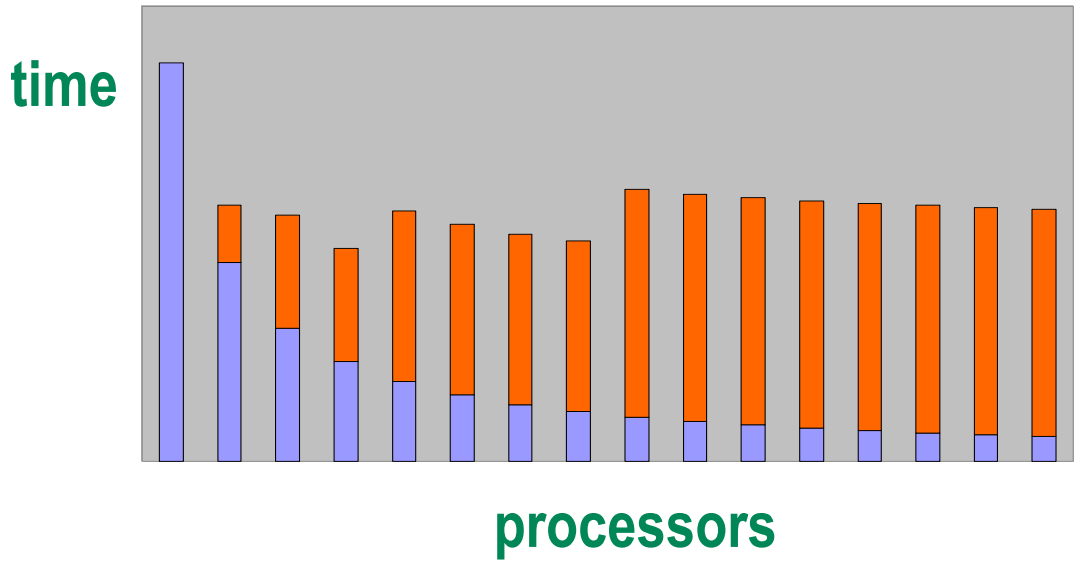
Speedup


super-linear speedup (wonderful)
linear speedup
sub-linear speedup (common)
Sp
# of processors

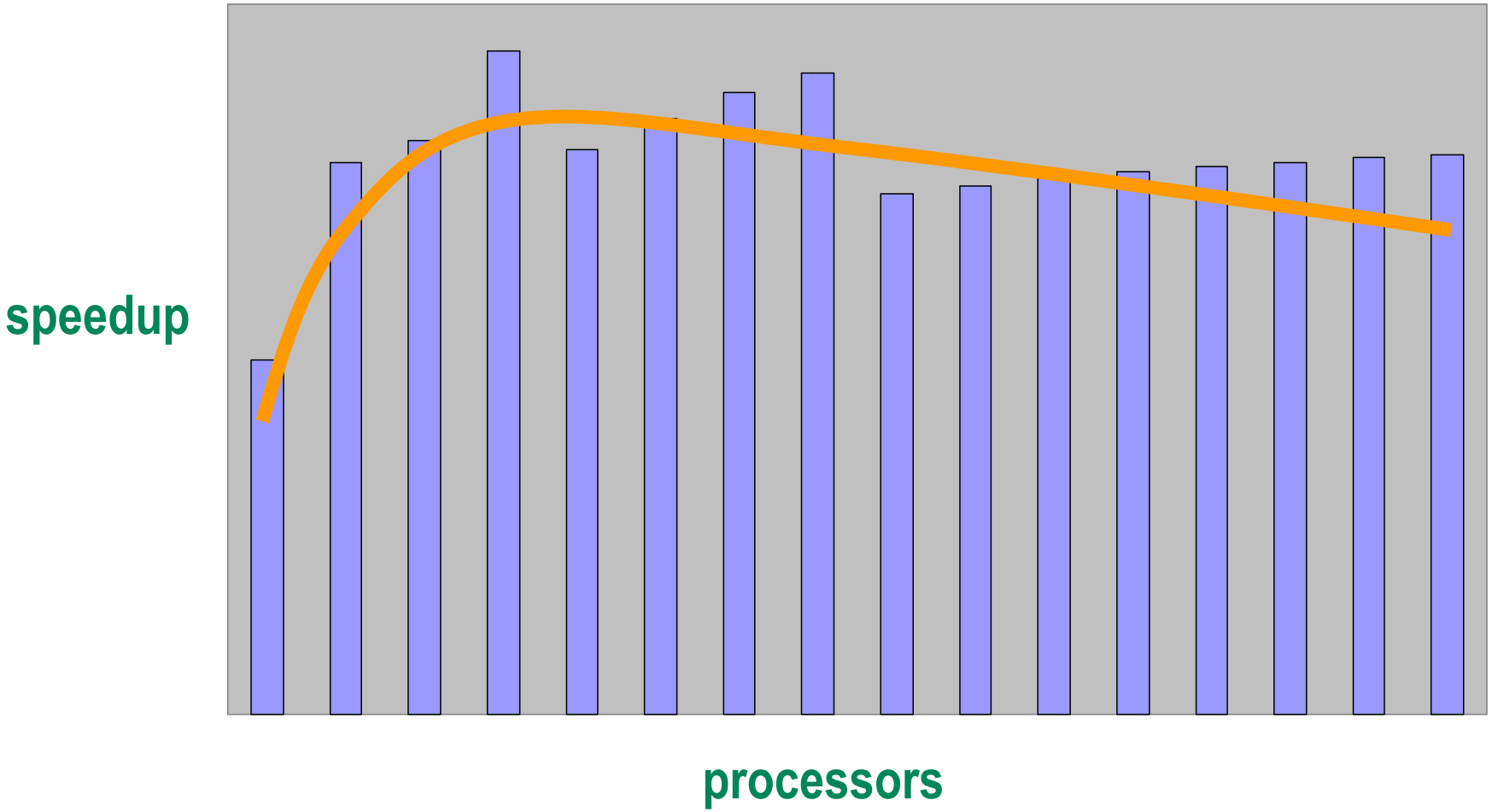
Real Case Scenario







Speedup Plot



Efficiency

- The **efficiency** is defined as the ratio of speedup to the number of processors. Efficiency measures the fraction of time for which a processor is usefully utilized.

$$E = \frac{S}{p} = \frac{T_S}{pT_P}$$

where

$$S(speedup) = \frac{T_S}{T_P}$$

- Ideal (or perfect) speedup means 100% efficiency $E = 1$
- Many difficult-to-parallelize algorithms have efficiency that approaches zero as P increases

Cost

- The **cost** of solving a problem on a parallel system is defined as the product of run time and the number of processors.
- A cost-optimal parallel system solves a problem with a cost proportional to the execution time of the fastest known sequential algorithm on a single processor.

Scalability

- Speedup describes how the parallel algorithm's performance changes with increasing *P*
- ***Scalability*** concerns the efficiency of the algorithm with changing problem size *N* by choosing *P* dependent on *N* so that the efficiency of the algorithm is bounded below
- Simply, Scalability is a measure of a parallel system's capacity to increase speedup in proportion to the number of processors.
- Well-known **Amdahl's law** dictates the achievable speedup and efficiency

Amdahl's Law (1967)

- The speedup of a program using multiple processors in parallel computing is limited by the time needed for the serial fraction of the problem.
- Why serial part?
  - Algorithm limitations: dependencies
  - Bottlenecks: shared resources
  - Startup overhead: starting a parallel program
  - Communication: interacting with others workers

Amdahl's Law (1967)

- If a problem of size W (T_S) has a serial component W_S (t_{ser}), then parallel time T_P can be achieved as

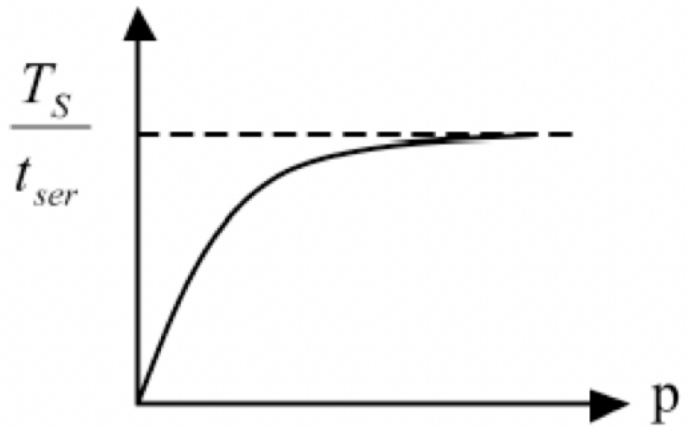
$$T_P = t_{ser} + \frac{T_S - t_{ser}}{p}$$

- Then, speedup is

$$S = \frac{T_S}{T_P} = \frac{T_S}{t_{ser} + \frac{T_S - t_{ser}}{p}}$$

- If p is increased a lot,

$$S = \frac{T_S}{T_P} \cong \frac{T_S}{t_{ser}}$$



Amdahl's Law (1967)

- Another view of Amdahl's Law

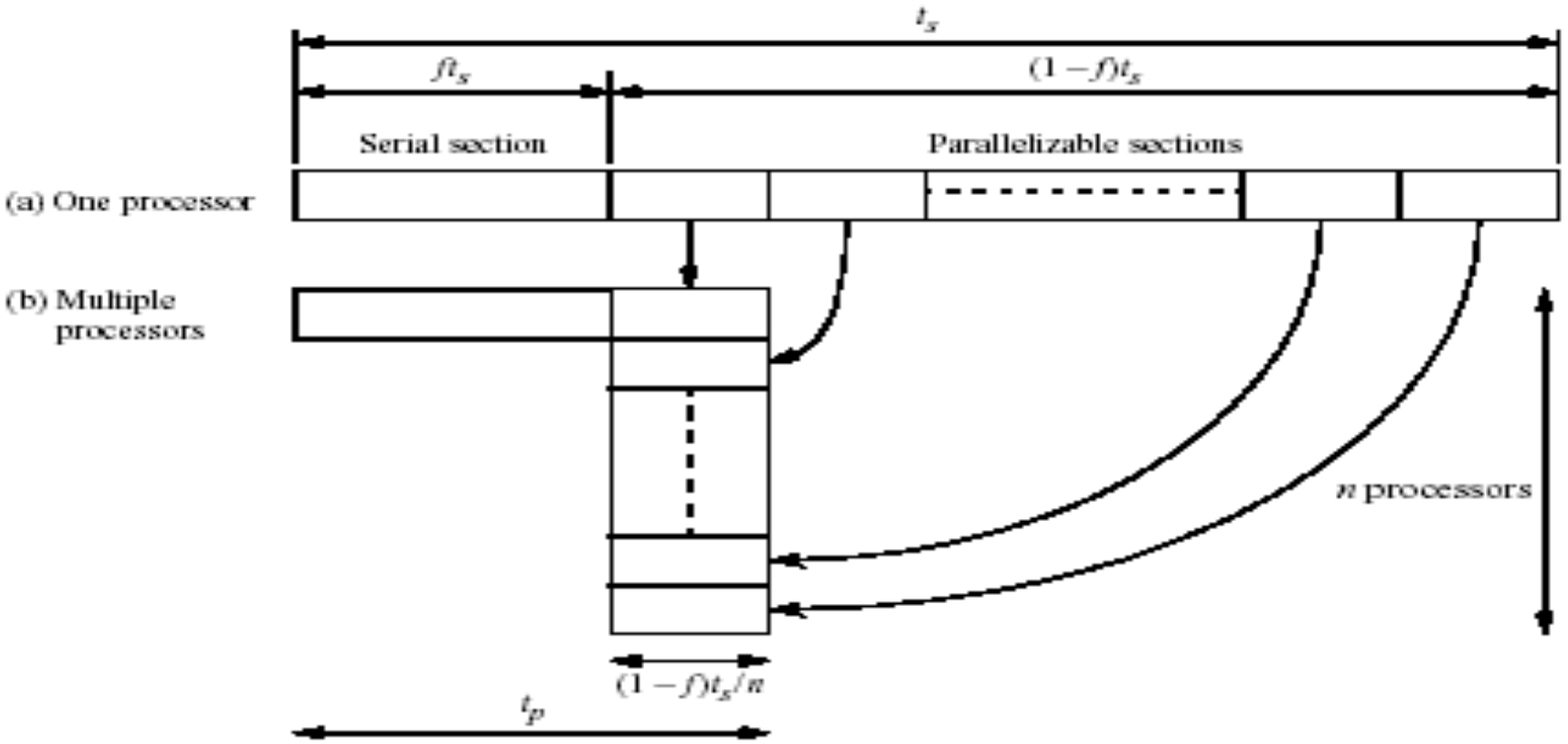


Figure 1.29 Parallelizing sequential problem — Amdahl's law.

Example 1:

- 95% of a program's execution time occurs inside a loop that can be executed in parallel. What is the maximum speedup we should expect from a parallel version of the program executing on 8 CPUs?

$$S(speedup) = \frac{T_S}{T_P} = \frac{1}{0.05 + \frac{(1 - 0.05)}{8}} \cong 5.926$$

Example 2:

- Serial part is 5%. Even with p=1024 processors the speedup is less than or equal to S = 20. True or False?
  - Answer: True

$$S(speedup) = \frac{T_S}{T_P} = \frac{1}{0.05 + \frac{(1 - 0.05)}{1024}} \cong 19.635$$

Quiz Example:

- Your collaborator (pharmaceutical modeler) gives you a serial pharmacokinetic MATLAB program. Then, asks you how much faster it might run on 8 processors.
- You can only find one function amenable to a parallel solution.
- Benchmarking on a single processor reveals 80% of the execution time is spent inside this function.
- The serial code took 10 hours. How much will it take with your parallel program? What is the best speedup a parallel version is likely to achieve on 8 processors?

Gustafson's Law

- Also known as the Gustafson-Barsis's Law
- Any sufficiently large problem can be efficiently parallelized with a speedup

$$S = p - \alpha(p - 1)$$

where p is the number of processors, and α is the serial portion of the problem

- Gustafson proposed a fixed time concept which leads to scaled speedup for larger problem sizes.
- Amdahl's point of view is focused on a fixed computation problem size as it deals with a code taking a fixed amount of sequential calculation time. Gustafson's objection is that massively parallel machines allow computations previously unfeasible since they enable computations on very large data sets in fixed amount of time. In other words, a parallel platform does more than speeding up the execution of a code: it enables dealing with larger problems.

Homework Assignment

Suppose the run-time of a serial program is given by $T_{\text{serial}} = n^2$, where the units of the run-time are in microseconds. Suppose that a parallelization of this program has run-time $T_{\text{parallel}} = n^2/p + \log_2(p)$. Write a program that finds the speedups and efficiencies of this program for various values of n and p. Run your program with $n = 10, 20, 40, \ldots, 320$, and $p = 1, 2, 4, \ldots, 128$. What happens to the speedups and efficiencies as p is increased and n is held fixed? What happens when p is fixed and n is increased?

C++ Online

- https://replit.com/
- https://www.w3schools.com/cpp/cpp_compiler.asp