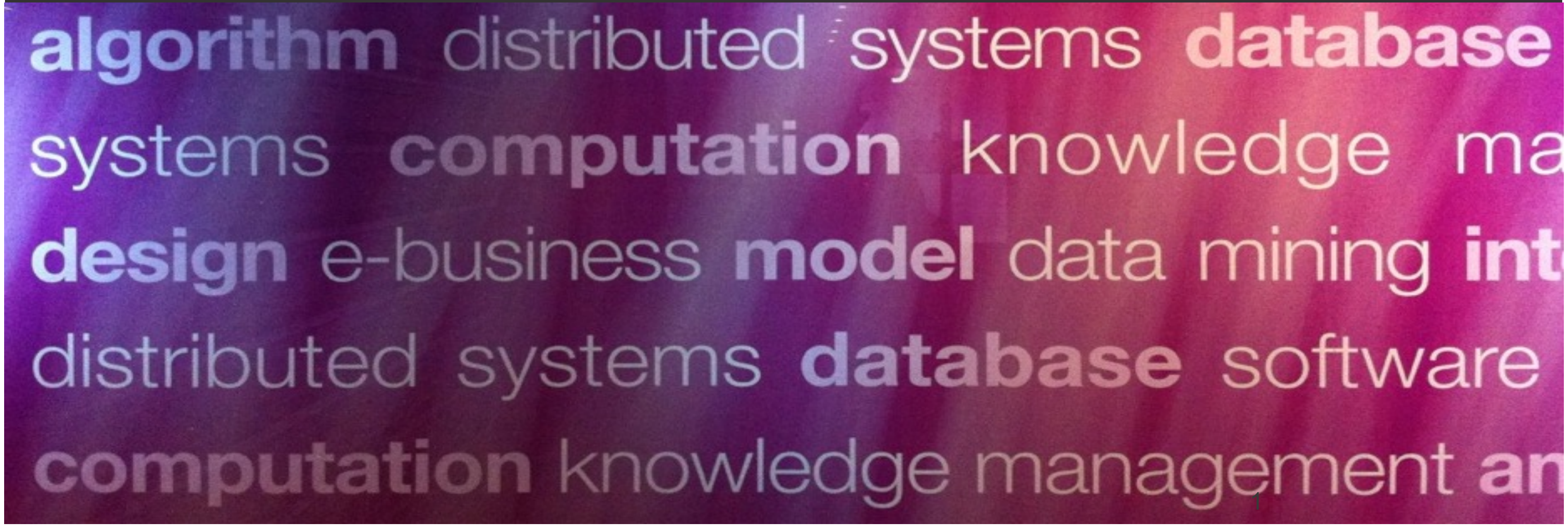
MONASH University

Information Technology

FIT3143 - LECTURE WEEK 1

INTRODUCTION TO PARALLEL COMPUTING


algorithm distributed systems database
systems computation knowledge ma
design e-business model data mining int
distributed systems database software
computation knowledge management an

Overview

1. Parallel computing concept and applications
2. Parallel computing models
3. Parallel computing performance

Associated learning outcomes

- Explain the fundamental principles of parallel computing architectures and algorithms (LO1)
- Analyze and evaluate the performance of parallel algorithms (LO4)

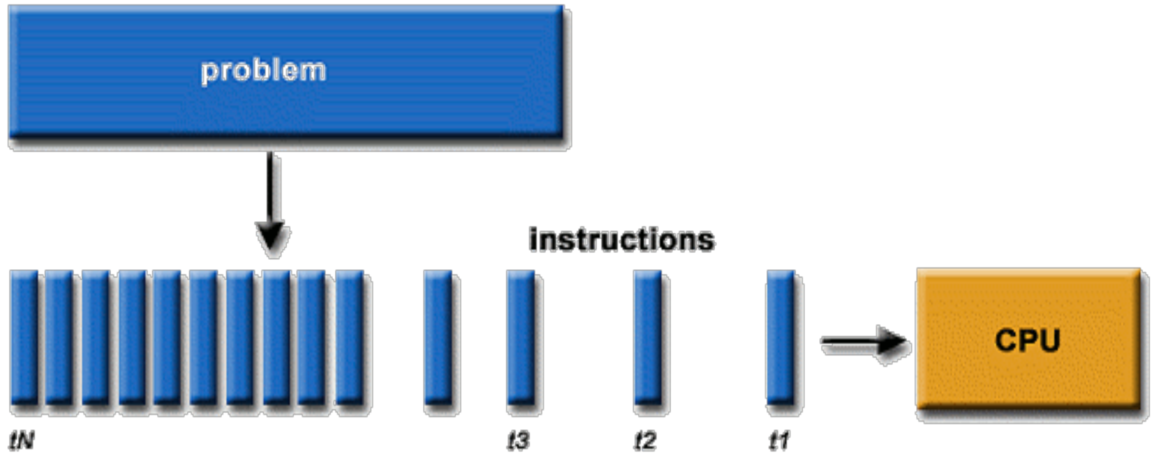
1. Parallel computing concept and applications

Definitions

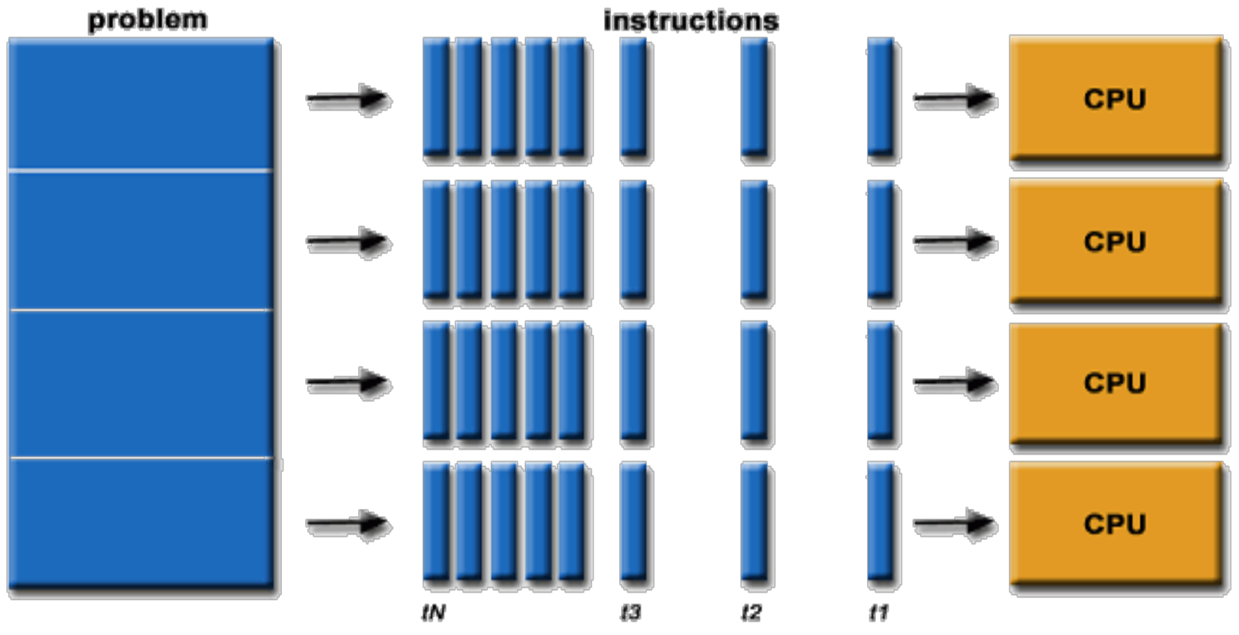
A parallel computer is simply a single computer with multiple internal processors, or a group of multiple computers interconnected to form a coherent high-performance computing platform.

Parallel programming is programming multiple computers, or computers with multiple internal processors, to solve a problem at a greater computational speed than is possible with a single computer.

4



Serial Computing



Parallel Computing

Adapted from https://computing.llnl.gov/tutorials/parallel_comp/

5

The demand for computational speed

There is a continual demand for greater computational speed from a computer system than is currently possible.

Areas requiring great computational speed include **numerical modeling** and **simulation of scientific and engineering problems**.

Computations must be completed within a "**reasonable**" time period.

Grand challenge problems

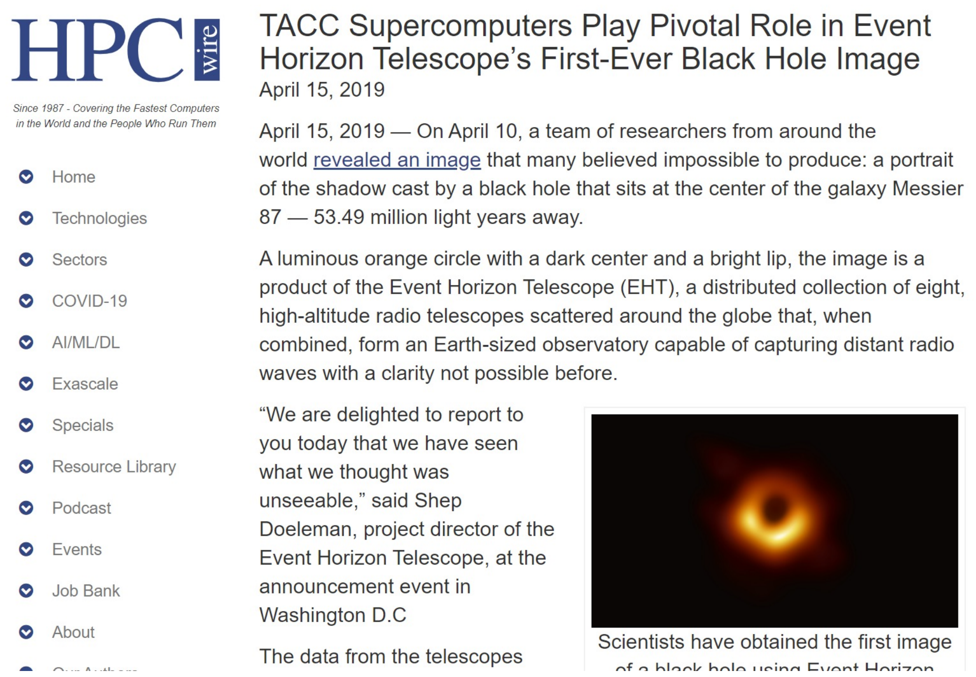
A **grand challenge problem** is one that cannot be solved in a reasonable amount of time with today’s computers.

For example, an execution time of 10 years is usually unacceptable.

Examples:

- Modeling large DNA structures
- Global weather forecasting
- Modeling motion of astronomical bodies

Computationally challenging problems


HPC wire

Since 1987 - Covering the Fastest Computers in the World and the People Who Run Them

- Home
- Technologies
- Sectors
- COVID-19
- AI/ML/DL
- Exascale
- Specials
- Resource Library
- Podcast
- Events
- Job Bank
- About

TACC Supercomputers Play Pivotal Role in Event Horizon Telescope's First-Ever Black Hole Image

April 15, 2019

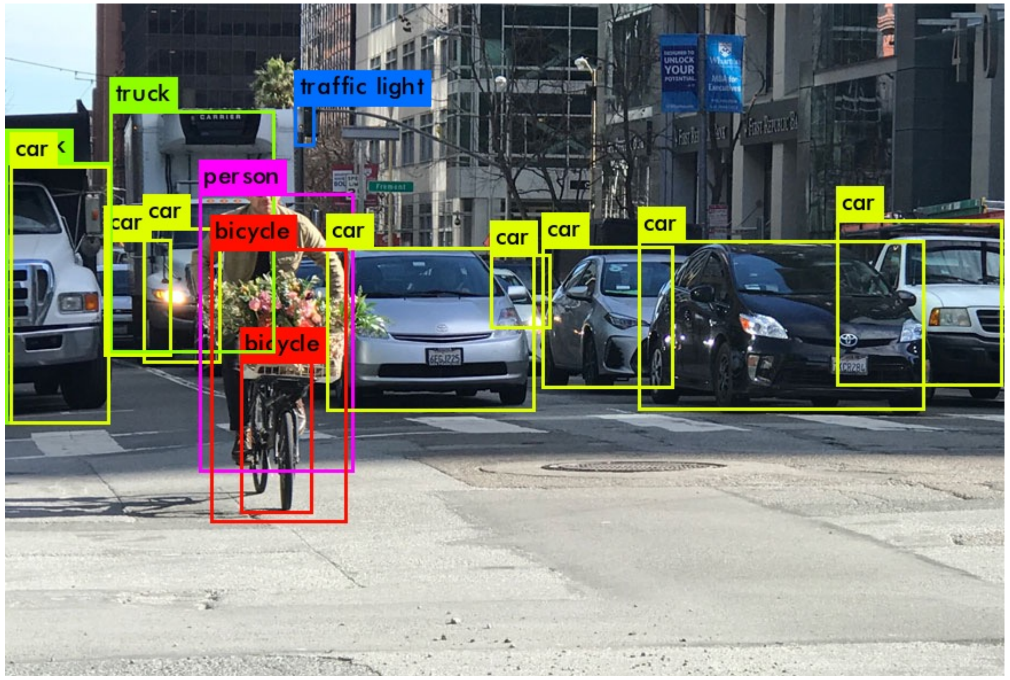
April 15, 2019 — On April 10, a team of researchers from around the world revealed an image that many believed impossible to produce: a portrait of the shadow cast by a black hole that sits at the center of the galaxy Messier 87 — 53.49 million light years away.

A luminous orange circle with a dark center and a bright lip, the image is a product of the Event Horizon Telescope (EHT), a distributed collection of eight, high-altitude radio telescopes scattered around the globe that, when combined, form an Earth-sized observatory capable of capturing distant radio waves with a clarity not possible before.

"We are delighted to report to you today that we have seen what we thought was unseeable," said Shep Doeleman, project director of the Event Horizon Telescope, at the announcement event in Washington D.C

The data from the telescopes

Scientists have obtained the first image



Reasons for not using existing serial machines

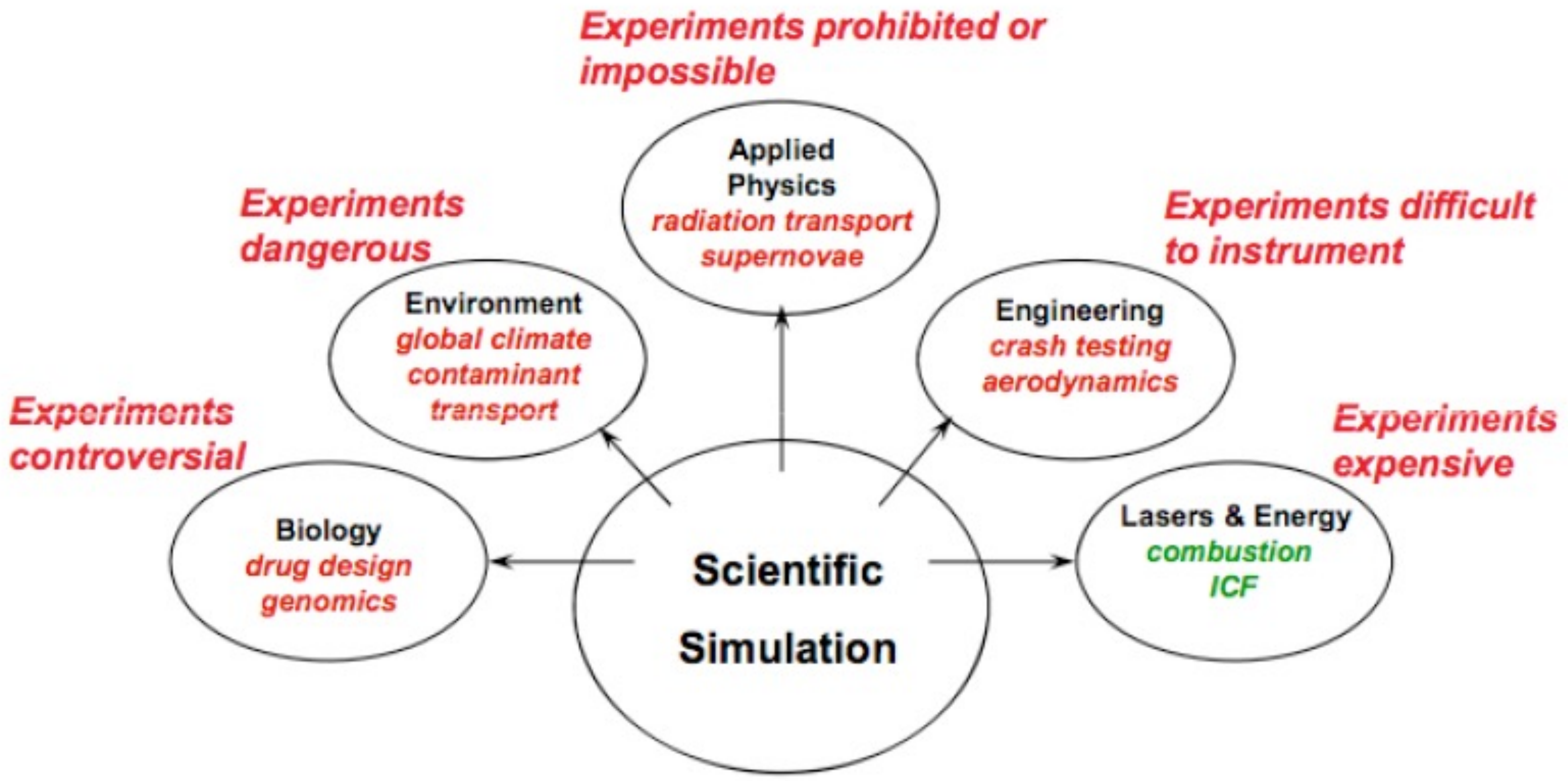
- May not be able to reach the solution **within a reasonable amount of time** if using a single Von Neumann machine.

- The **limits** imposed by both:
  - ➢ Physical constraints such as the speed of light
  - ➢ Cost of designing and manufacturing new chips.
    - - Current speed of chips - may reach a few billion Hz but will not grow enormously beyond that.

Reasons for using parallel machines

- With parallelism, much greater speeds can be achieved. The fastest computer in the world has a peak speed of about 415,530 Teraflops (https://top500.org/lists/top500/2020/06/). (flops: floating point instructions)
- The **enormous size** of the problems that are being investigated. These problems fall in many domains of **science, engineering**, and **economics**.
- To solve these problems requires carrying out huge numbers of computations, such as 10^{12}, 10^{14}, 10^{16} or even more. This number of computations cannot be done by any existing serial machine but is feasible to attack using parallel processing.

Why is Parallel Computing Important?



Source: Vivek Sarkar, Rice University, 2008

Some Particularly Challenging Computations

- **Science**
  - Global climate modelling
  - Biology: genomics; protein folding; drug design
  - Astrophysical modelling
  - Computational Chemistry
  - Computational Material Sciences and Nanosciences
- **Engineering**
  - Semiconductor design
  - Earthquake and structural modelling, remote sensing
  - Computation fluid dynamics (airplane design) & Combustion (engine design)
  - Simulation
  - Deep learning
  - Game design
  - Telecommunications (e.g. Network monitoring & optimization)
  - Autonomous systems
- **Business**
  - Financial derivatives and economic modelling
  - Transaction processing, web services and search engines
  - Analytics
- **Defense**
  - Nuclear weapons -- test by simulations
  - Cryptography

Source: Vivek Sarkar, Rice University, 2008

Why Parallelism is now necessary for Mainstream Computing?

- In single core processor, the performance of the processor will not gain much simply by the increasing operating frequency:
  - **Memory wall**

    The processor was incapable of optimizing the performance by increasing the operating frequency as the actual cause is the increasing gap between the CPU and memory throughput (data transfer rate).
  - **ILP (Instruction Level Parallelism) wall**

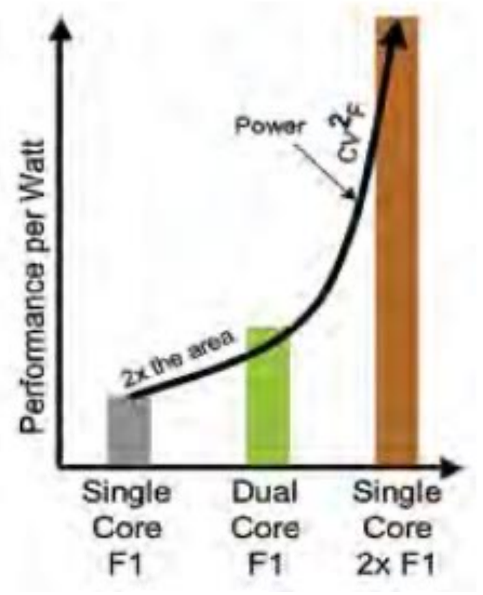
    Difficulty in full parallel instruction processing.
  - **Power wall**

    Power consumption doubled following the doubling of operating frequency.

    Faster clock speeds require higher input voltages.

    Every transistor leaks a small amount of current

    1% clock increase results in a 3% power increase.

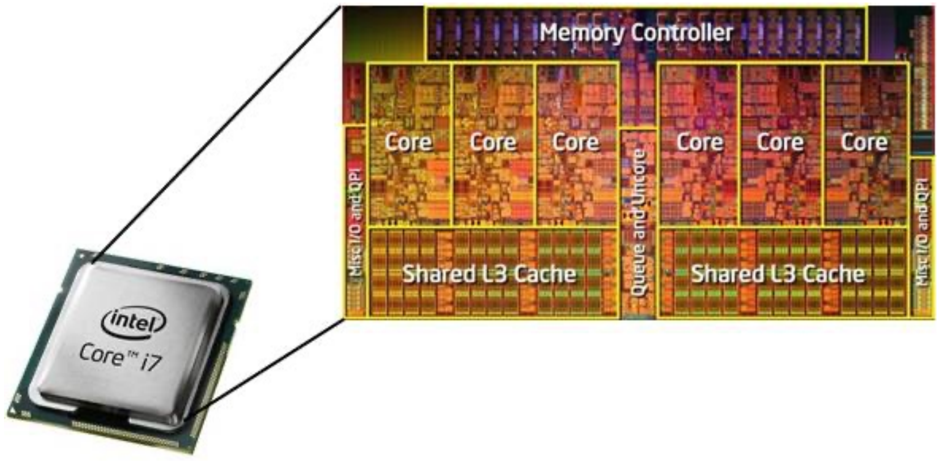


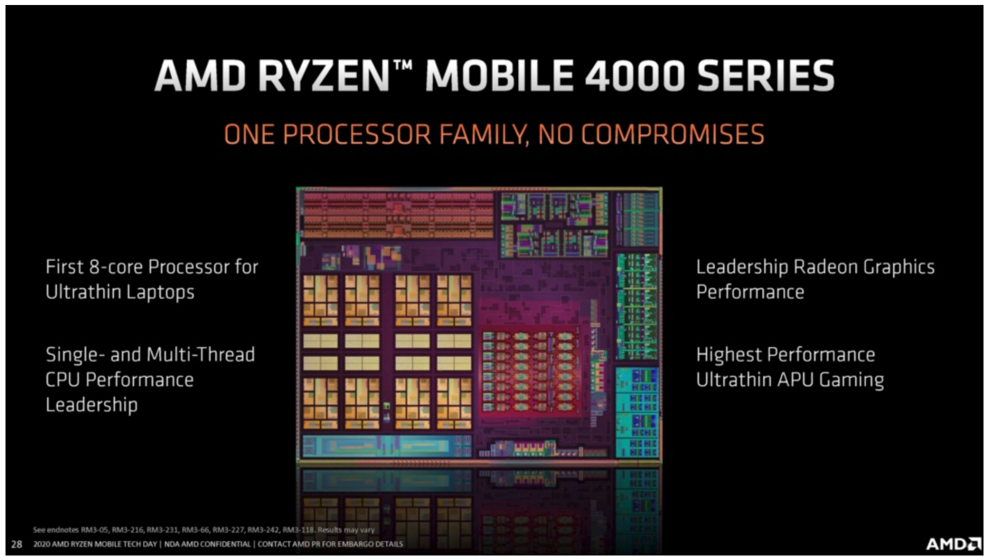
Types of parallelism

- **Bit level parallelism**
  - Parallelism achieved by doubling word size
- **Instruction level parallelism**
  - Superscalar processor with multiple executing units and out of order execution.
- **Data level parallelism**
  - Each processor performs the same task on different pieces of distributed data
- **Task level parallelism**
  - Each processor executes a different thread (or process) on the same or different data

Classes of parallel computers

Multi core processor




AMD RYZEN™ MOBILE 4000 SERIES

ONE PROCESSOR FAMILY, NO COMPROMISES

First 8-core Processor for Ultrathin Laptops

Single- and Multi-Thread CPU Performance Leadership

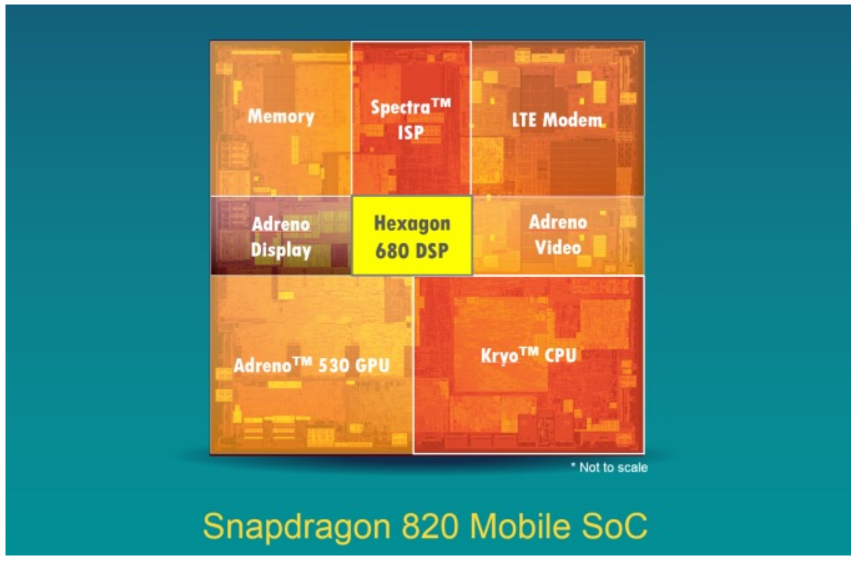
Leadership Radeon Graphics Performance

Highest Performance Ultrathin APU Gaming

See endnotes RM3-05, RM3-216, RM3-231, RM3-66, RM3-227, RM3-242, RM3-118. Results may vary

28 2020 AMD RYZEN MOBILE TECH DAY | NDA AMD CONFIDENTIAL | CONTACT AMD PR FOR EMBARGO DETAILS

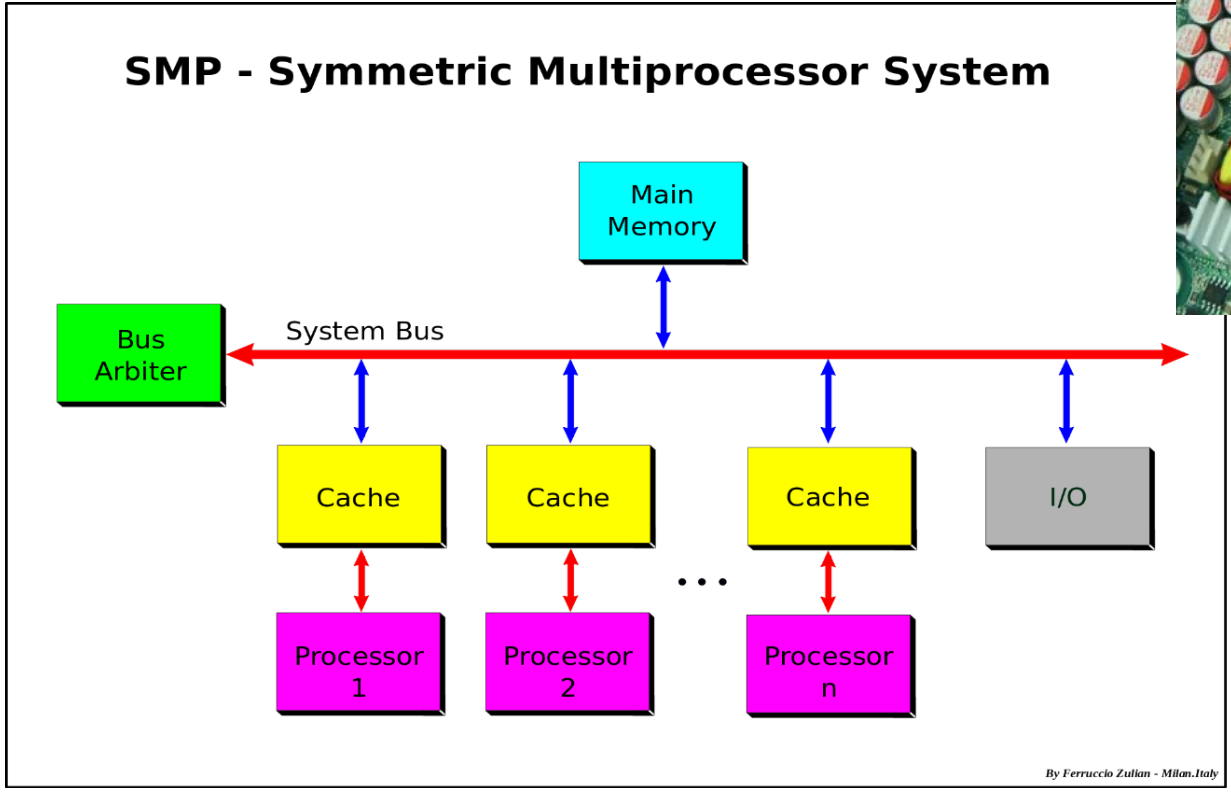
AMD



Snapdragon 820 Mobile SoC

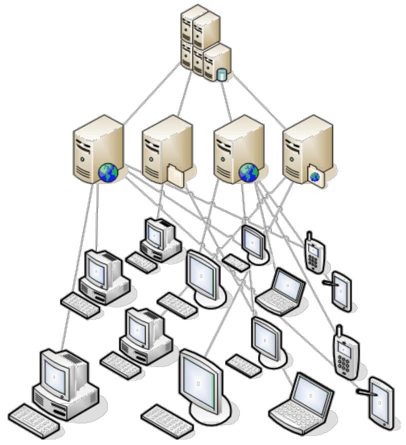
Classes of parallel computers

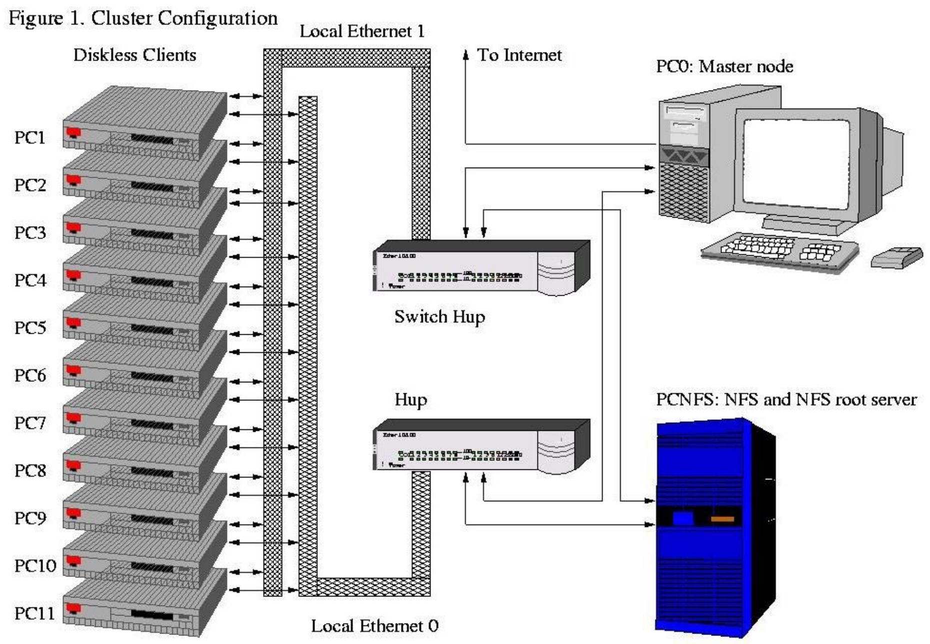
Symmetric multiprocessor



Classes of parallel computers

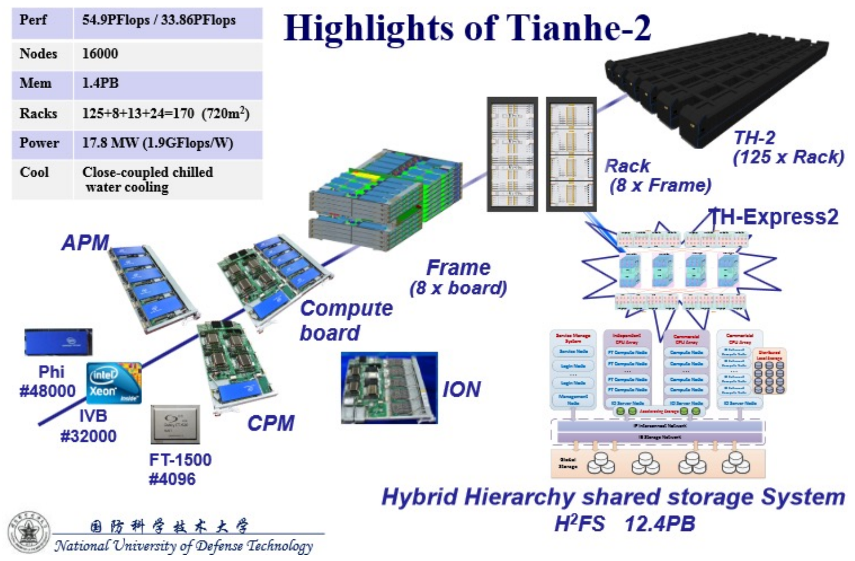
Cluster & Distributed/Grid computing





Classes of parallel computers

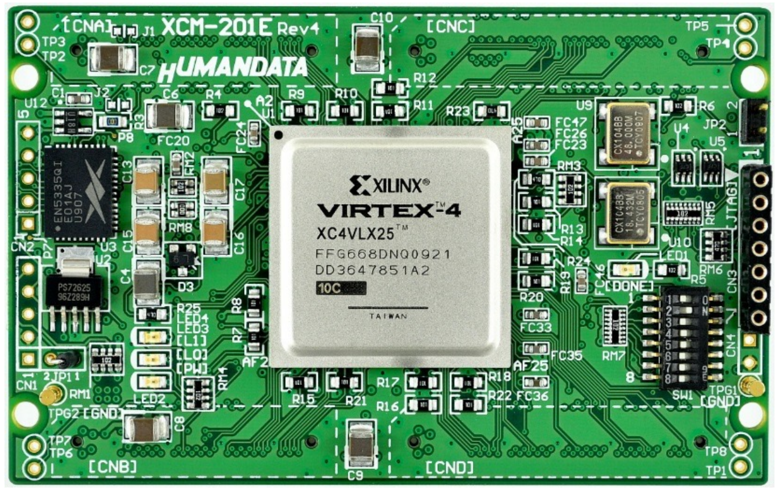
Massively parallel computing / supercomputing

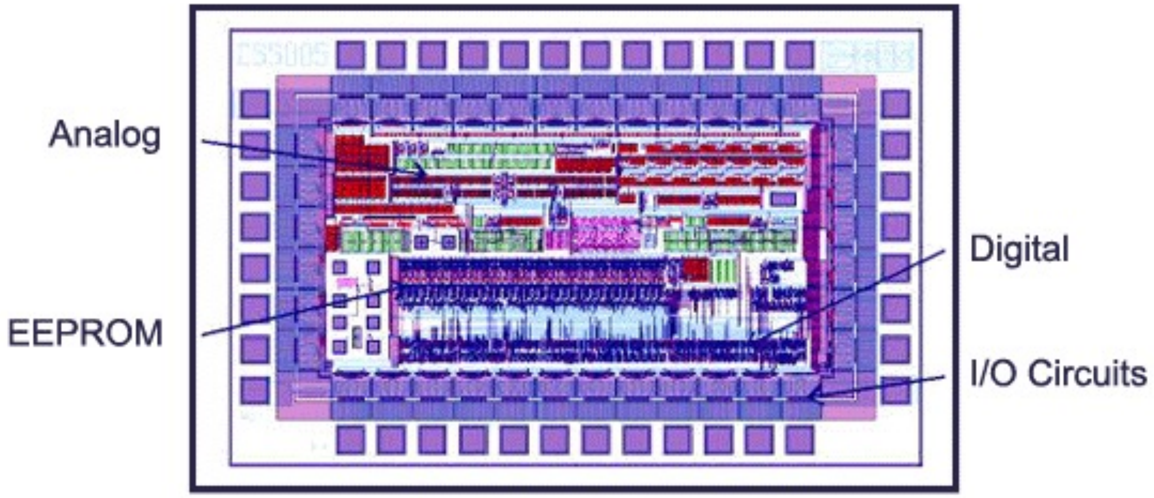


Perf	54.9PFlops / 33.86PFlops
Nodes	16000
Mem	1.4PB
Racks	125+8+13+24=170 (720m^2)
Power	17.8 MW (1.9GFlops/W)
Cool	Close-coupled chilled water cooling

Classes of parallel computers

Reconfigurable computing (FPGA) & ASIC

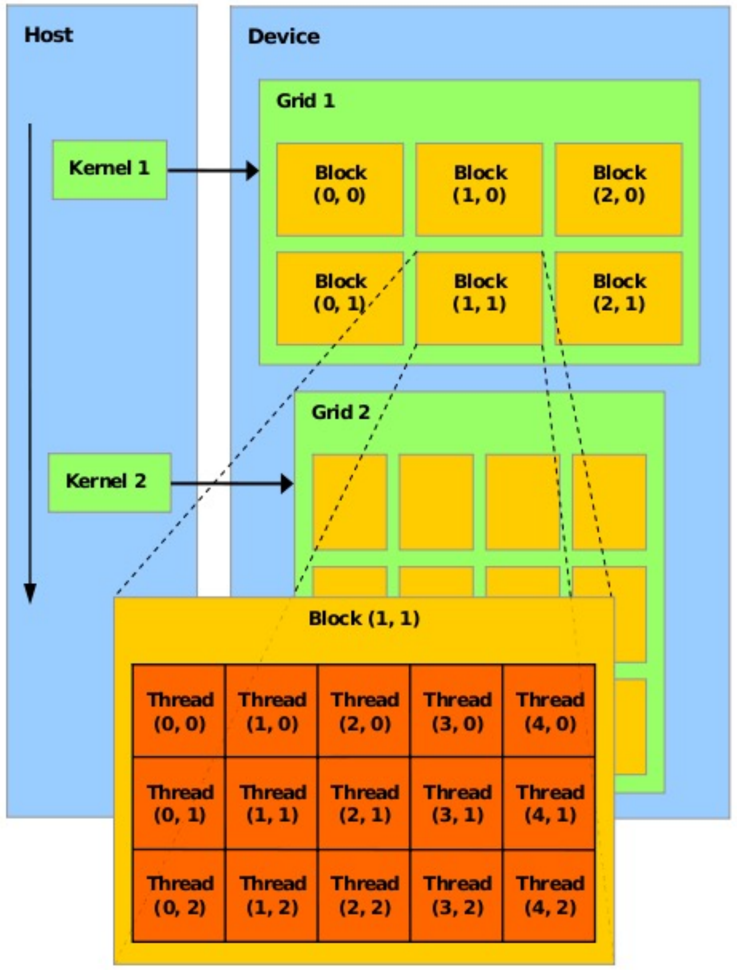




A Typical Mixed Signal ASIC

Classes of parallel computers

General-purpose computing on graphics processing units (GPGPU)



E.g. Embarrassingly parallel computation

An **ideal** parallel computation is one that can be immediately divided into completely <u>independent</u> parts that can be executed simultaneously – this is what is known as an *embarrassingly-parallel computation*, or *job-level parallelism*.

In this context, a network is a collection of integrated processors that are communicating and sharing information to speed up the solution to a single task.

Issues

1. How big of a collection?

- *A few very powerful, special purpose computers? (2-64)*
- *Many smaller, standard microprocessors? (100-10,000)*
- *A huge number of simple, one bit processors? (billions, trillions)*

2. What kind of processors?

- *Special purpose?*
- *Standard microprocessors or Simple?*
- *One-bit processors?*

3. How closely integrated is the collection of processors?

http://www.lsbu.ac.uk/oracle/oracle7/server/doc/SPS73/chap3.htm

- ***Tightly coupled****? Designed as a single architecture and a single system.*
- ***Loosely coupled****? Systems that can operate as a single logical system but can also function independently. Not designed as a single architecture.*

4. **How do the processors communicate?**
   - Shared memory?
   - External communications channel?
5. **What type of information is shared?**
   - Data values, inputs, results?
   - Synchronizing information?
6. **Focus on a single task**
   - We will focus more on ***task level parallelism*** rather than *job-level parallelism* or *embarrassingly parallel computation*. This means we're more interested in situations where all the processors work together to speed up the solution of a single task.

Job = Task1 + Task2 + Task3 + …

2. Parallel computing models

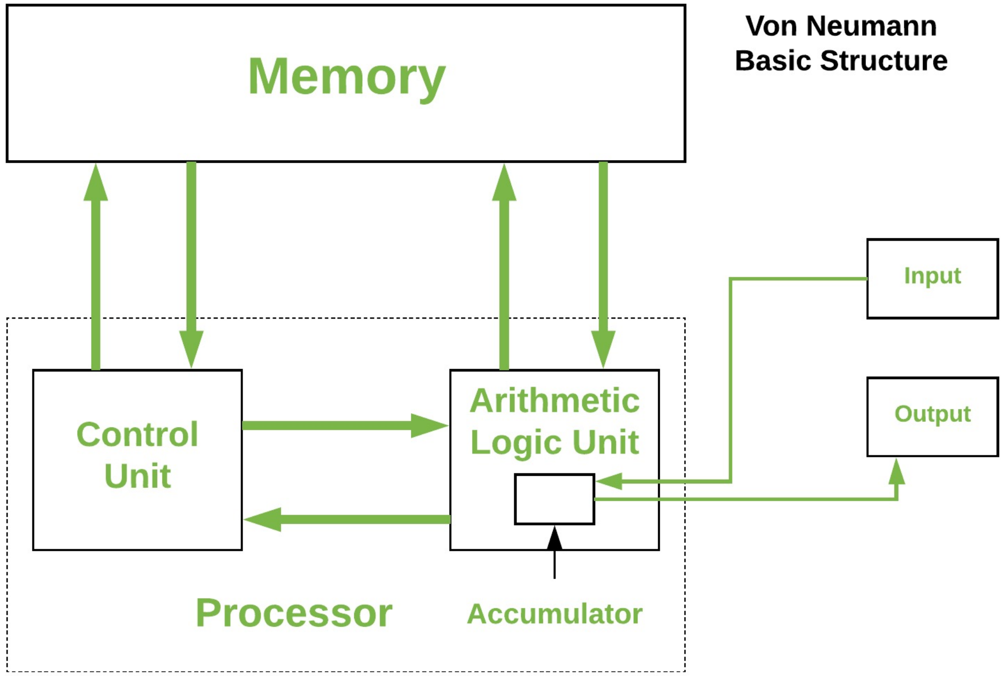
How to classify parallel computers?

Parallel computers can be **classified** by:

- Type and number of processors.
- Interconnection scheme of the processors.
- Communication scheme.
- Input/output operations.

Conventional computer

- Consists of a processor executing a program stored in a (main) memory.



- Each main memory location located by its *address.* Addresses start at 0 and extend to 2^n -1 when there are n bits (binary digits) in the address. (You learned this in COA)

Image link: https://media.geeksforgeeks.org/wp-content/uploads/basic_structure.png

Multi-processor computer architecture

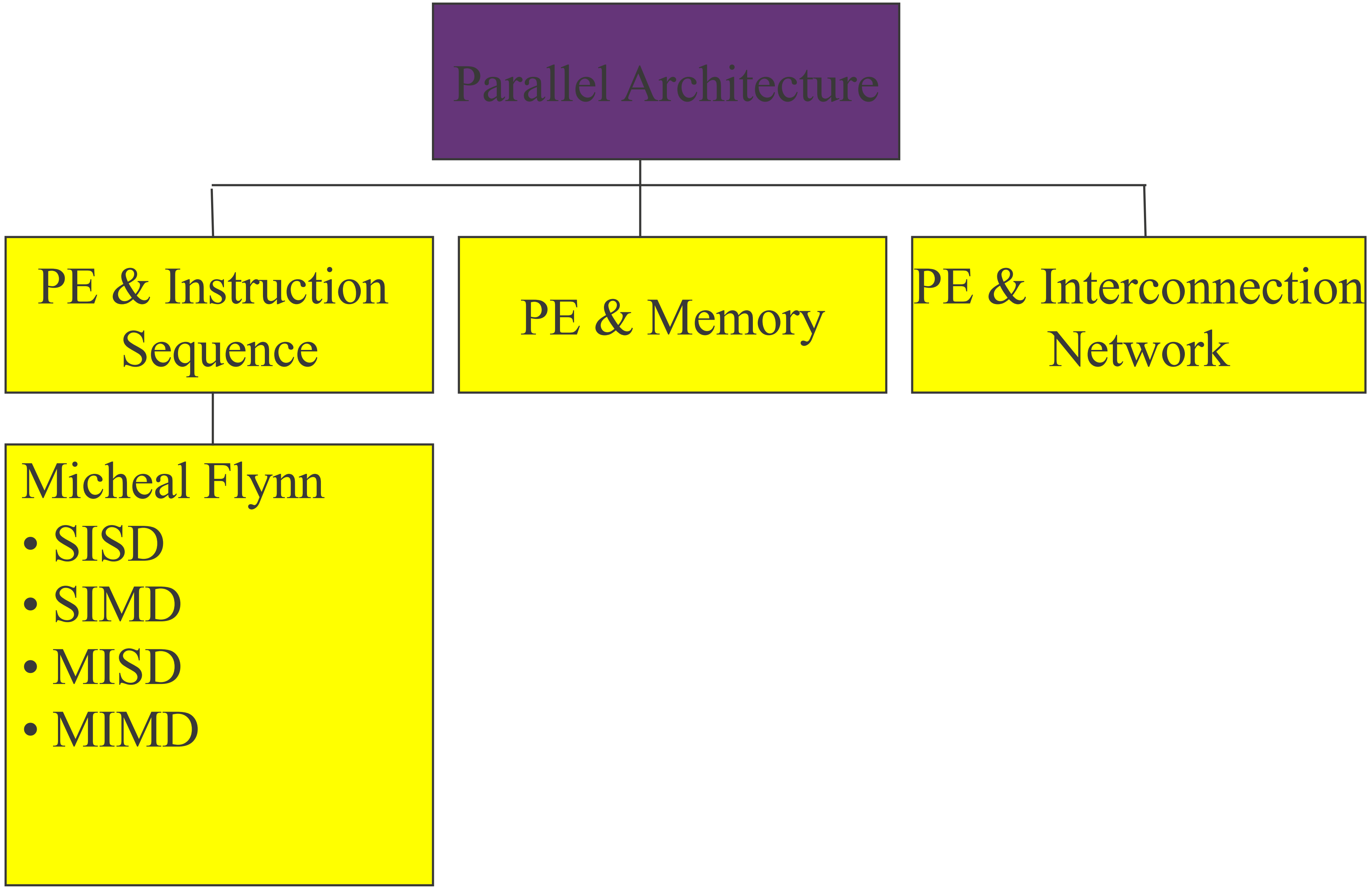
1. Flynn’s Classification
2. Shared memory model
3. Distributed memory model
4. Distributed shared memory model

Taxonomy (Classification)

A taxonomy of parallel architectures can be built based on three relationships:

1. **Relationship between PE and the instruction sequence executed**
2. **Relationship between PE and the memory**
3. **Relationship between PE and the interconnection network**

PE: Processing Element (CPU)

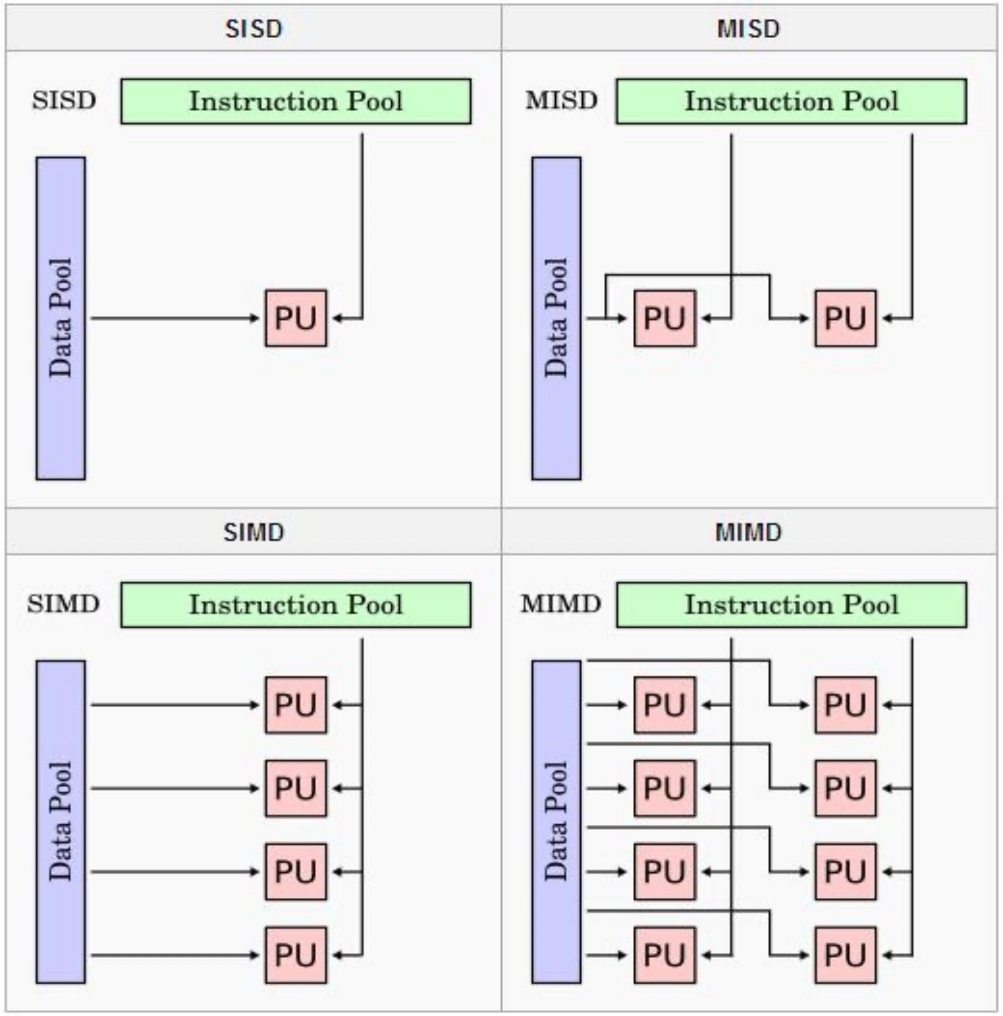

Parallel Architecture
PE & Instruction Sequence
PE & Memory
PE & Interconnection Network
Micheal Flynn
• SISD
• SIMD
• MISD
• MIMD

Flynn taxonomy

Michael Flynn (1966) developed a taxonomy of parallel systems based on the number of independent *instruction and data streams*.

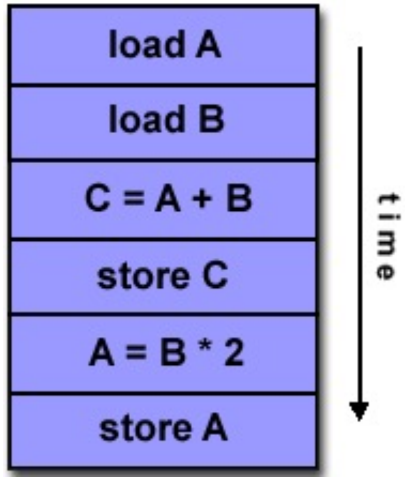
1. ***SISD:*** *Single Instruction Stream- Single Data Stream*
   This is a good conventional sequential Von Neumann machine.
2. ***MISD:*** *Multiple Instruction Stream- Single Data Stream*
3. ***SIMD:*** *Single Instruction Stream- Multiple Data Stream*
4. ***MIMD:*** *Multiple Instruction Stream- Multiple Data Stream*

Flynn’s Classification



Flynn’s Classification

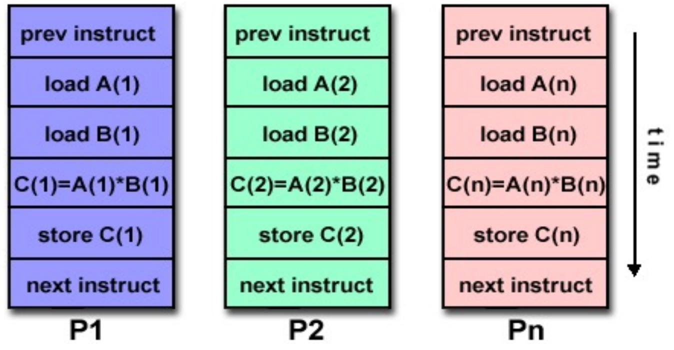
- **Single Instruction, Single Data (SISD):**
  - A serial (non-parallel) computer
  - Single instruction: only one instruction stream is being acted on by the CPU during any one clock cycle
  - Single data: only one data stream is being used as input during any one clock cycle
  - Deterministic execution
  - This is the oldest and until recently, the most prevalent form of computer
  - Examples: most PCs, single CPU workstations and mainframes



Flynn’s Classification

Single Instruction, Multiple Data (SIMD):

- A type of parallel computer
- Single instruction: All processing units execute the same instruction at any given clock cycle
- Multiple data: Each processing unit can operate on a different data element
- This type of machine typically has an instruction dispatcher, a very high-bandwidth internal network, and a very large array of very small-capacity instruction units.
- Best suited for specialized problems characterized by a high degree of regularity, such as image processing.
- Synchronous (lockstep) and deterministic execution
- Two varieties: Processor Arrays and Vector Pipelines



- Examples:
  - Processor Arrays: Connection Machine CM-2, Maspar MP-1, MP-2
  - Vector Pipelines: IBM 9000, Cray C90, Fujitsu VP, NEC SX-2, Hitachi S820

Flynn’s Classification

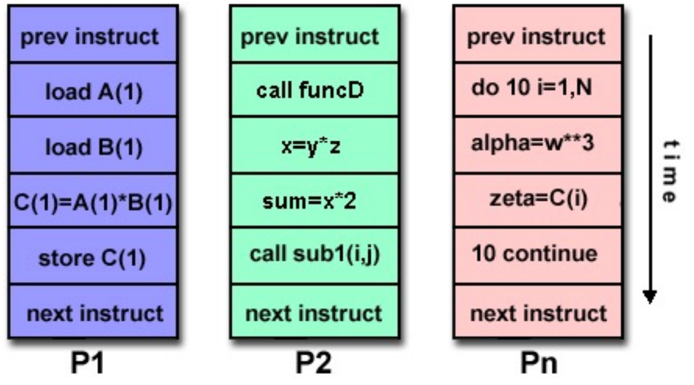
Multiple Instruction, Single Data (MISD):

- A single data stream is fed into multiple processing units.
- Each processing unit operates on data independently via independent instruction streams.
- Few actual examples of this class of parallel computer have ever existed. One is the experimental Carnegie-Mellon computer
- Some conceivable uses might be:
  - multiple frequency filters operating on a single signal stream
  - multiple cryptography algorithms attempting to crack a single coded message.

Flynn’s Classification

Multiple Instruction, Multiple Data (MIMD):

- Currently, the most common type of parallel computer. Most modern computers fall into this category.
- Multiple Instruction: every processor may be executing a different instruction stream
- Multiple Data: every processor may be working with a different data stream
- Execution can be synchronous or asynchronous, deterministic or non-deterministic
- Examples: most current supercomputers, networked parallel computer "grids" and multi-processor SMP computers - including some types of PCs.

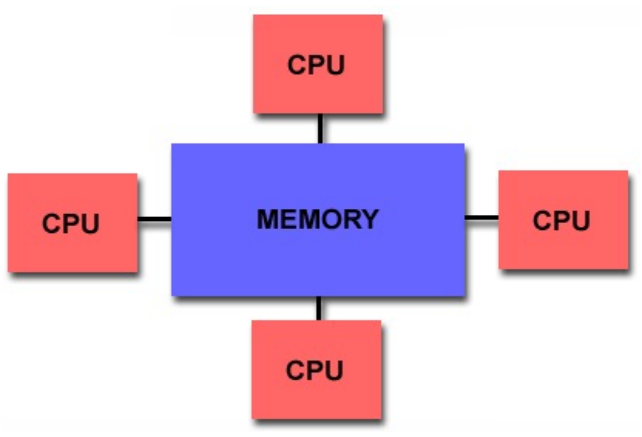


Parallel Computer Memory Architectures

- Broadly divided into three categories
  - Shared memory
  - Distributed memory
  - Hybrid

Shared Memory

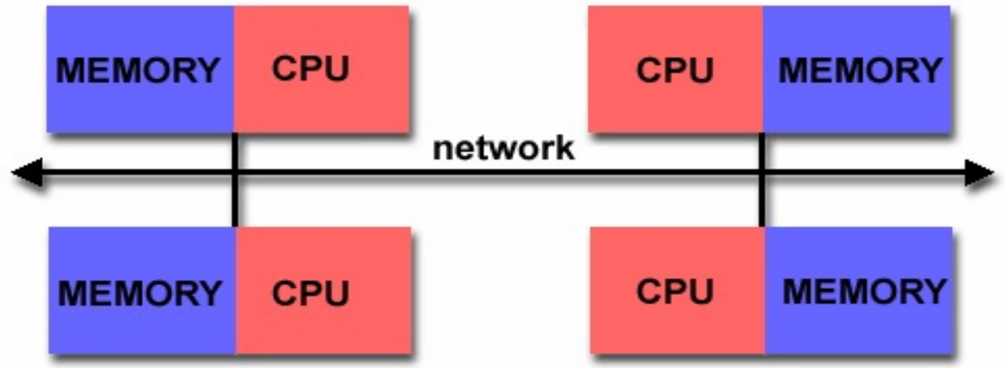
- Shared memory parallel computers vary widely, but generally have in common the ability for all processors to access all memory as global address space.
- Multiple processors can operate independently but share the same memory resources.
- Changes in a memory location effected by one processor are visible to all other processors.
- Shared memory machines can be divided into two main classes based upon memory access times: ***UMA*** and ***NUMA***.



Parallel Computer Memory Architectures

Distributed Memory

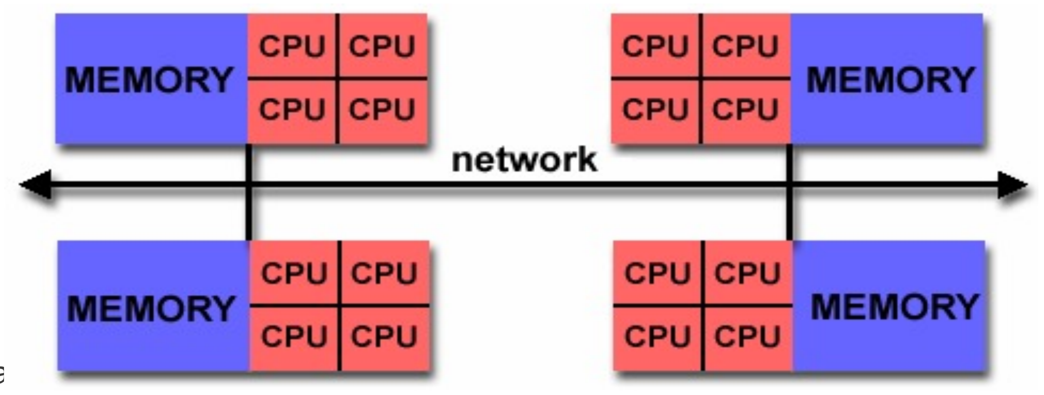
- Distributed memory systems require a communication network to connect inter-processor memory.
- Processors have their own local memory. There is no concept of global address space across all processors.
- Because each processor has its own local memory, it operates independently. Changes it makes to its local memory have no effect on the memory of other processors. Hence, the concept of cache coherency does not apply.
- When a processor needs access to data in another processor, it is usually the task of the programmer to explicitly define how and when data is communicated. Synchronization between tasks is likewise the programmer's responsibility.
- The network "fabric" used for data transfer varies widely, though it can be as simple as Ethernet.



Parallel Computer Memory Architectures

Hybrid

- The largest and fastest computers in the world today employ both shared and distributed memory architectures.
- The shared memory component is usually a cache coherent SMP machine. Processors on a given SMP can address that machine's memory as global.
- The distributed memory component is the networking of multiple SMPs. SMPs know only about their own memory - not the memory on another SMP. Therefore, network communications are required to move data from one SMP to another.
- Current trends seem to indicate that this type of memory architecture will continue to prevail and increase at the high end of computing for the foreseeable future.
- Advantages and Disadvantages: whatever is common to both shared and distributed memory architectures.



Parallel Programming Models

Overview

- There are several parallel programming models in common use:
  - Shared Memory
  - Threads
  - Message Passing
  - Data Parallel
  - Hybrid

- Parallel programming models exist as an abstraction above hardware and memory architectures.

- Although it might not seem apparent, these models are NOT specific to a particular type of machine or memory architecture. In fact, any of these models can (theoretically) be implemented on any underlying hardware.

Single Program Multiple Data (SPMD) structure

- Another programming structure we may use is the **Single Program Multiple Data (SPMD)** structure.
- In this structure, a single source program is written and each processor will execute its personal copy of this program, although independently, and not in synchrony.
- This source program can be constructed so that parts of the program are executed by certain computers and not others depending on the identity of the computer.
- For a **master-slave** structure, the programs could have parts for the master and parts for slaves.

Multiple Program Multiple Data (MPMD) structure

- Within the MIMD classification, each processor will have its own program to execute. This could be described as MPMD.
- In this case, some of the programs to be executed could be copies of the same program.
- Typically only two source programs are written, one for the designated **master processor**, and one for the remaining processors, which are called **slave processors**.

Three ways to create executable code for a shared memory multiprocessor

1. **Parallel Programming Languages**
   - With special parallel programming constructs and statements that allow shared variables and parallel code sections to be declared.
   - Then the compiler is responsible for producing the final executable code.
2. **Threads**
   - Contain regular high-level language code sequences for individual processors. These code sequences can then access shared locations.
3. **Sequential programming**
   - Use a regular sequential programming language and modify the syntax to specify parallelism.

Programming message-passing multicomputer

- **Dividing** the problem into parts that are intended to be executed simultaneously to solve the problem

- Common approach is to use message-passing library routines that are inserted into a conventional sequential program for message passing.

- A problem is divided into a number of concurrent processes that may be executed on a different computer.

- Processes **communicate by sending messages**; this will be the only way to distribute data and results between processes.

3. Parallel computing performance

Potential for increased computational speed

Process

In all forms of MIMD multiprocessor/multicomputer systems, it is necessary to divide the computations into tasks or processes that can be executed simultaneously.

The size of a process can be described by its granularity, which is related to the number of processors being used.

In coarse granularity, each process contains a large number of sequential instructions and takes substantial time to execute.

In fine granularity, a process might consist of few instructions or perhaps even one instruction.

Medium granularity describes the middle ground.

Sometimes granularity is defined as the size of the computation between communication or synchronization points.

Generally, we want to increase the granularity to reduce the cost of process creation and inter-process communication, but of course this will likely reduce the number of concurrent processes and amount of parallelism. A suitable compromise has to be made.

The ratio

$$\frac{Computation}{Communication} = \frac{Computation\ Time}{Communication\ Time} = \frac{t_{comp}}{t_{comm}}$$

can be used as a **granularity metric**.

Speed-Up factor

A measure of relative performance between a multiprocessor system and a single processor system in the ***speed- up factor***, *S(n),* defined as

$$S(n) = \frac{\text{Execution time using one processor (single processor system)}}{\text{Execution time using a multiprocessor with n processors}}$$

$$= \frac{t_S}{t_P}$$

The factor $S_{(n)}$ gives the increase in speed due to using a multiprocessor

For comparing a parallel solution with a sequential solution, we will use the fastest known sequential algorithm for running on a single processor. The underlying algorithm for a parallel implementation might be (and is usually) different.

The speed up factor can also be cast in terms of computational steps:

$$S_{(n)} = \frac{\text{Number of computational steps using one processor}}{\text{Number of parallel computational steps with processors}}$$

The **maximum speedup is n** with n processors (linear speed up).

Superlinear Speedup

where $S(n) > n$, may be seen on occasion, but usually this is due to using a suboptimal sequential algorithm or some unique feature of the architecture that favors the parallel formation.

One common reason for superlinear speedup is the **extra memory** in the multiprocessor system which can hold more of the problem data at any instant, it leads to less relatively slow disk-memory traffic.

Superlinear speedup can occur in search algorithms.

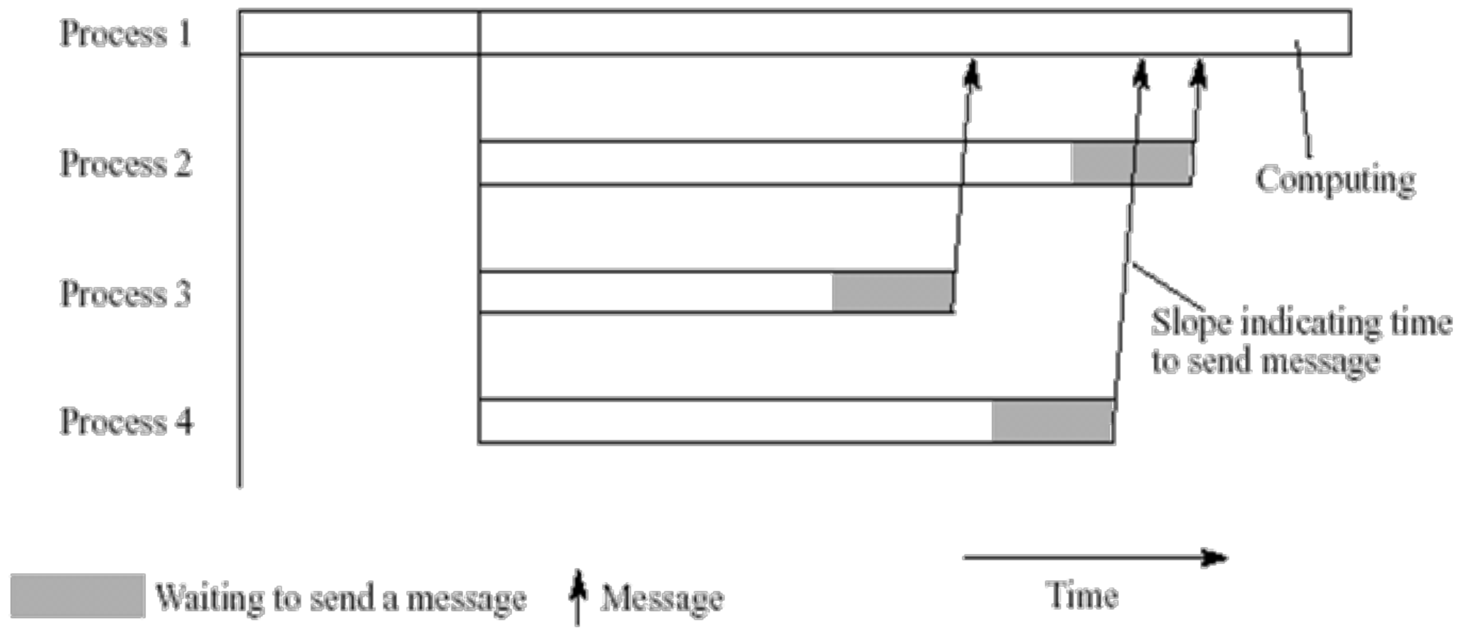
Space-time diagram of a message passing program

It is reasonable to expect that some part of a computation cannot be divided at all into concurrent processes and must be performed serially.

During the initialization period or period before concurrent processes are being setup, only one processor is doing useful work, but for the rest of the computation, additional processors are operating in parallel.

The figure on the next page illustrates a space-time diagram of a message-passing program that has one process on each of four processors. It starts with one process and later others begin to execute and send messages back. In general, we would expect periods when processors are idle and are waiting to send their messages.

Space- time diagram of a message- passing program


Process 1
Process 2
Process 3
Process 4
Computing
Slope indicating time to send message
Waiting to send a message
Message
Time

Maximum speed

If the fraction of the computation that cannot be divided into concurrent tasks is f, the time to perform the computation with *p* processors is given by $ft_s + (1-f)\, t_s/p$ as shown in the next slide.

Illustrated is the case with a single serial part at the beginning of the computation which cannot be parallelized, but the remaining parts could be distributed throughout the computation.

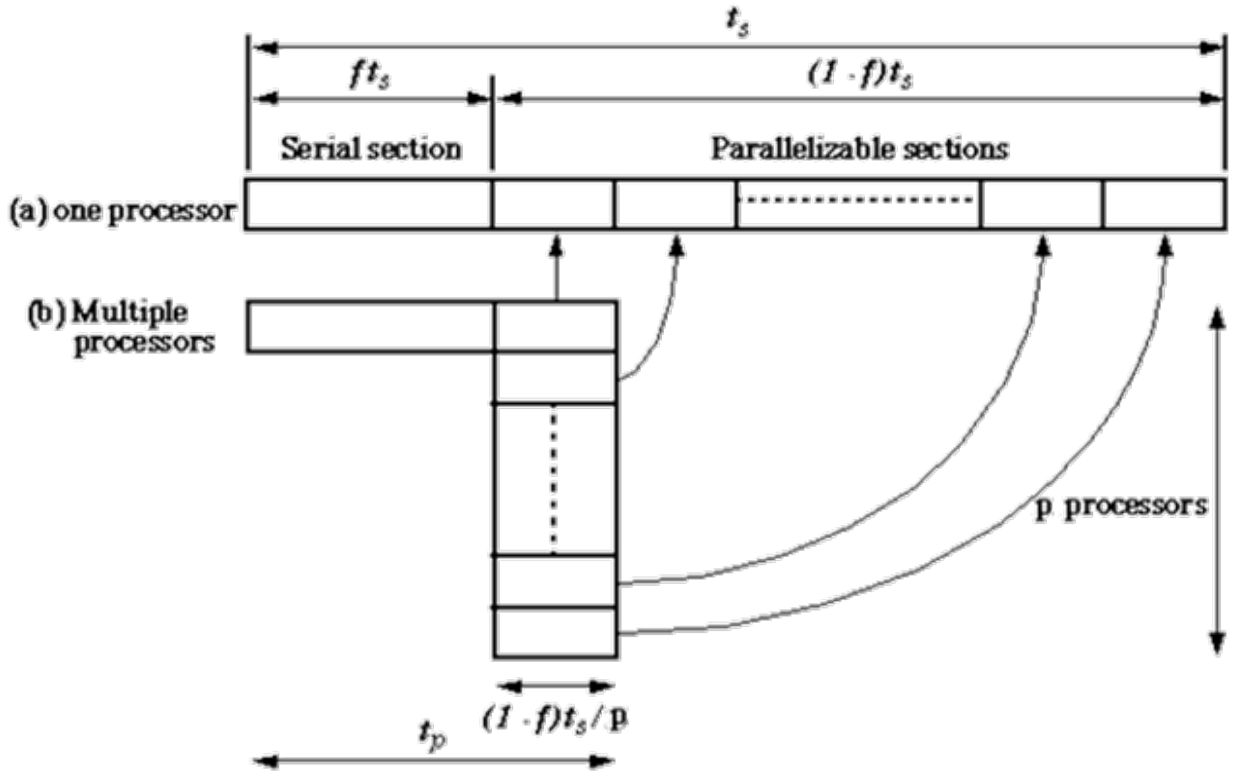
The speed up factor S(p) is given by

$$S(p) = \frac{t_S}{ft_S + (1-f)t_S/p} = \frac{p}{1+(p-1)f}$$

Number of

This whole thing is t_p

Parallelizing sequential problem-Amdahl's law



Alternative:

$$\frac{1}{r_s + \frac{r_p}{p}}$$

r_s is the serial ratio (non-parallelizable portion)

r_p is the parallel ratio (parallelizable portion)

p is the number of processors

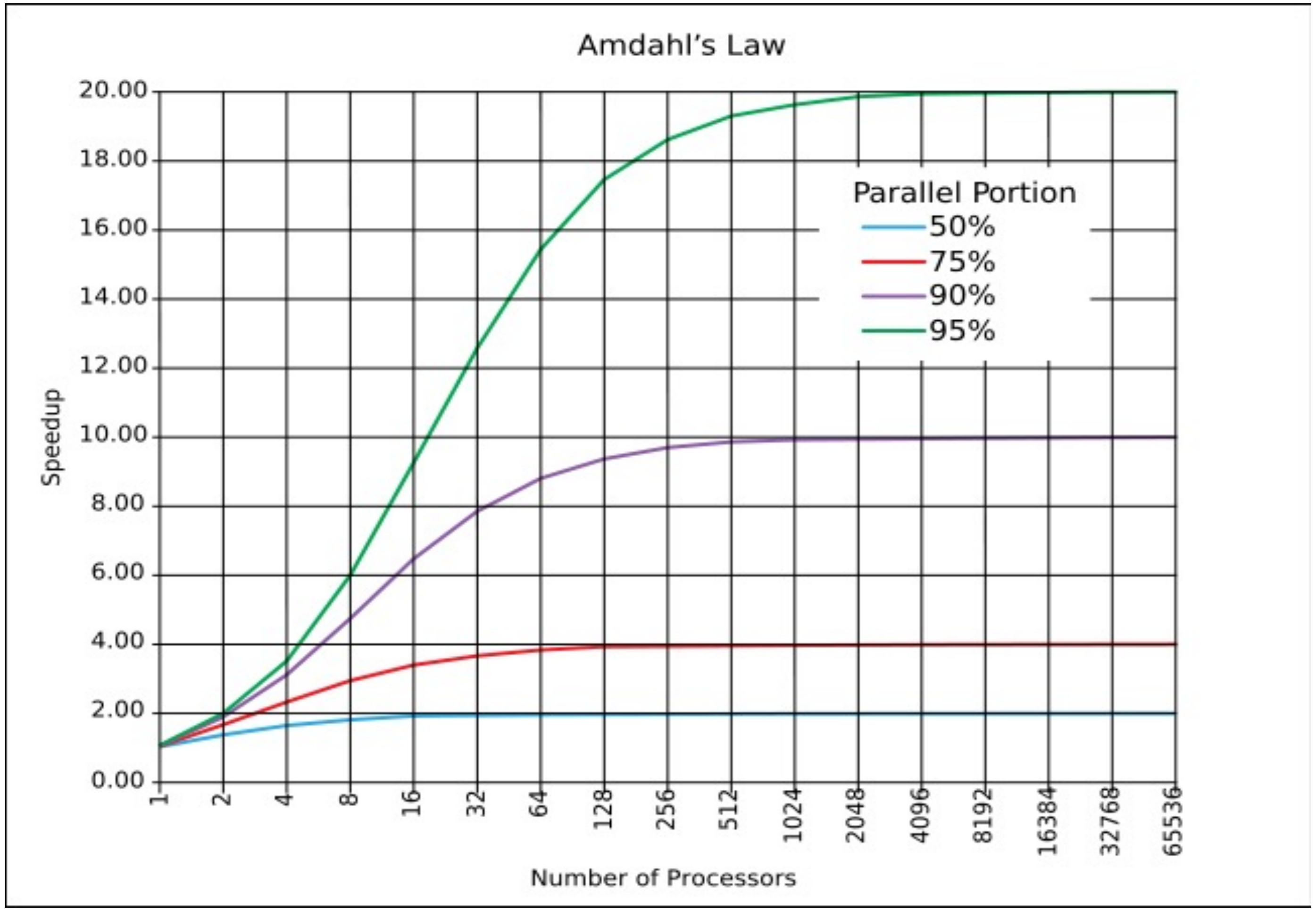
This equation is known as **Amdahl's law**.

It shows S(n) plotted against number of processors and against f. If a significant increase in speed of computation is to be achieved, the fraction of computation that is executed by concurrent processes needs to be substantial fraction of the overall computation.

Even with infinite number of processors, maximum speedup limited to 1/*f*.

i.e., $\displaystyle S(n) \underset{n \to \infty}{=} 1/f$

Amdahl's law



Amdahl's Law
Speedup
20.00
18.00
16.00
14.00
12.00
10.00
8.00
6.00
4.00
2.00
0.00
1
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
Number of Processors
Parallel Portion
50%
75%
90%
95%

Efficiency

- Efficiency

$$\text{Efficiency} = \frac{\text{Speedup}}{\text{Processors}} \qquad \varepsilon(n,p) \le \frac{\sigma(n)+\varphi(n)}{p\sigma(n)+\varphi(n)+p\kappa(n,p)}$$

- $0 \le \varepsilon(n,p) \le 1$
- Amdahl's law

$$\psi(n,p) \le \frac{\sigma(n)+\varphi(n)}{\sigma(n)+\varphi(n)/p+\kappa(n,p)}$$

$$\le \frac{\sigma(n)+\varphi(n)}{\sigma(n)+\varphi(n)/p}$$

- Let $f = \sigma(n)/(\sigma(n) + \varphi(n))$; i.e., f is the fraction of the code which is inherently sequential

$$\psi \le \frac{1}{f+(1-f)/p}$$

Examples

- 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?

$$\psi \leq \frac{1}{0.05 + (1 - 0.05)/8} \cong 5.9$$

- 20% of a program's execution time is spent within inherently sequential code. What is the limit to the speedup achievable by a parallel version of the program?

$$\lim_{p \to \infty} \frac{1}{0.2 + (1 - 0.2)/p} = \frac{1}{0.2} = 5$$

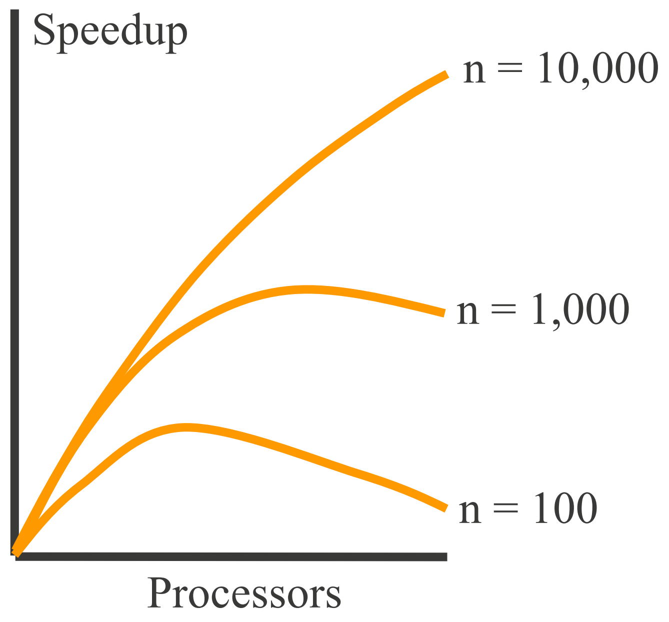
Amdahl's law

Limitations of Amdahl's Law

- Ignores $\kappa(n,p)$ - overestimates speedup
- Assumes f constant, so underestimates speedup achievable

Amdahl Effect

- Typically, $\kappa(n,p)$ has lower complexity than $\varphi(n)/p$
- As n increases, $\varphi(n)/p$ dominates $\kappa(n,p)$
- *As n increases, speedup increases*
- *As n increases, sequential fraction f decreases.*



Gustafson’s law

- **Gustafson's Law** (also known as **Gustafson-Barsis' law,** 1988) states that any sufficiently large problem can be efficiently parallelized.

- Gustafson's Law is closely related to Amdahl's law, which gives a limit to the degree to which a program can be sped up due to parallelization.

$$S_s(p) = p + (1 - p)f t_s$$

where *p* is the number of processors, *s* is the speedup, and *f* the non-parallelizable part of the process

- Gustafson's law addresses the shortcomings of Amdahl's law, which cannot scale to match availability of computing power as the machine size increases.

Scaled speedup Factor

- If we fix the parallel execution time t_p and the fraction of the process which cannot be parallelized *f*, we can use Gustafson’s law to figure out the speed-up factor given the number of processors, *p*.
- This is called the *scaled* speedup factor because the t_p is scaled to 1.

$$S_s(p) = \frac{ft_s + (1-f)t_s}{ft_s + (1-f)t_s / p} = \frac{p + (1-p)ft_s}{1} = p + (1-p)ft_s$$

How the numerator becomes this will be explained on the following slide.

Scaled speedup Factor (cont.)

- The denominator t*p* which we scaled to 1 was
  - $ft_s + (1-f)t_s/p = 1$
  - $pft_s+(1-f)t_s = p$
  - $t_s = p + (1-p)\ ft_s$
- Hence, we get

$$S_s(p) = \frac{ft_s + (1-f)t_s}{ft_s + (1-f)t_s / p} = \frac{p + (1-p)ft_s}{1} = p + (1-p)ft_s$$

Summary

- Definition of parallel computer and parallel programming
  - Parallel computer: computer with multiple internal processors, or a group of multiple computers interconnected
  - Parallel programming: programming multiple computers, or computers with multiple internal processors, to solve a computationally expensive problem
- Flynn's taxonomy
  - SISD, MISD, SIMD, MIMD
- Parallel computer memory architectures
  - Shared memory, Distributed memory, Hybrid
- Parallel programming models
  - Shared memory, Threads, Message Passing, Data Parallel & Hybrid
- Performance
  - Speed up factor: Measure of relative performance
  - Amdahl's Law
  - Gustafson's Law