Heterogeneous Parallel Programming
COMP4901D

Parallel Programming Models

Slides based on tutorial by *Blaise Barney*

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

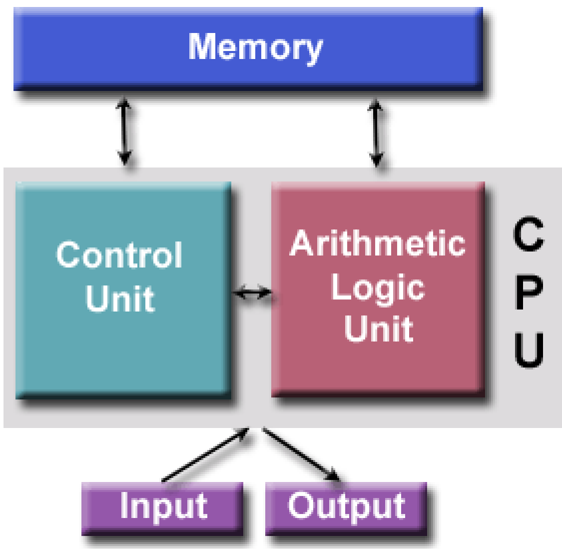
`

Overview

- Motivation for Parallel Computing
- Parallel Programming Models
- Issues in Parallel Programming

The World is Massively Parallel

von Neumann Architecture

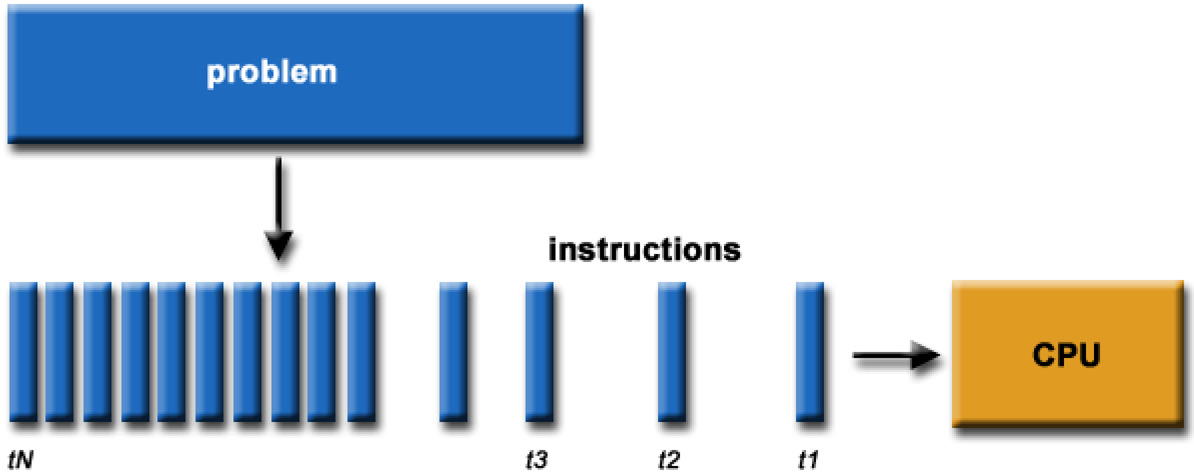


- Memory
  - Serve reads and writes
  - Random addressable
  - Store both data and instructions
- Control unit
  - Fetch instructions from memory
  - Execute instructions sequentially
- Arithmetic Logic Unit (ALU)
  - Perform arithmetic operations
- Input/Output
  - Interface to the human user

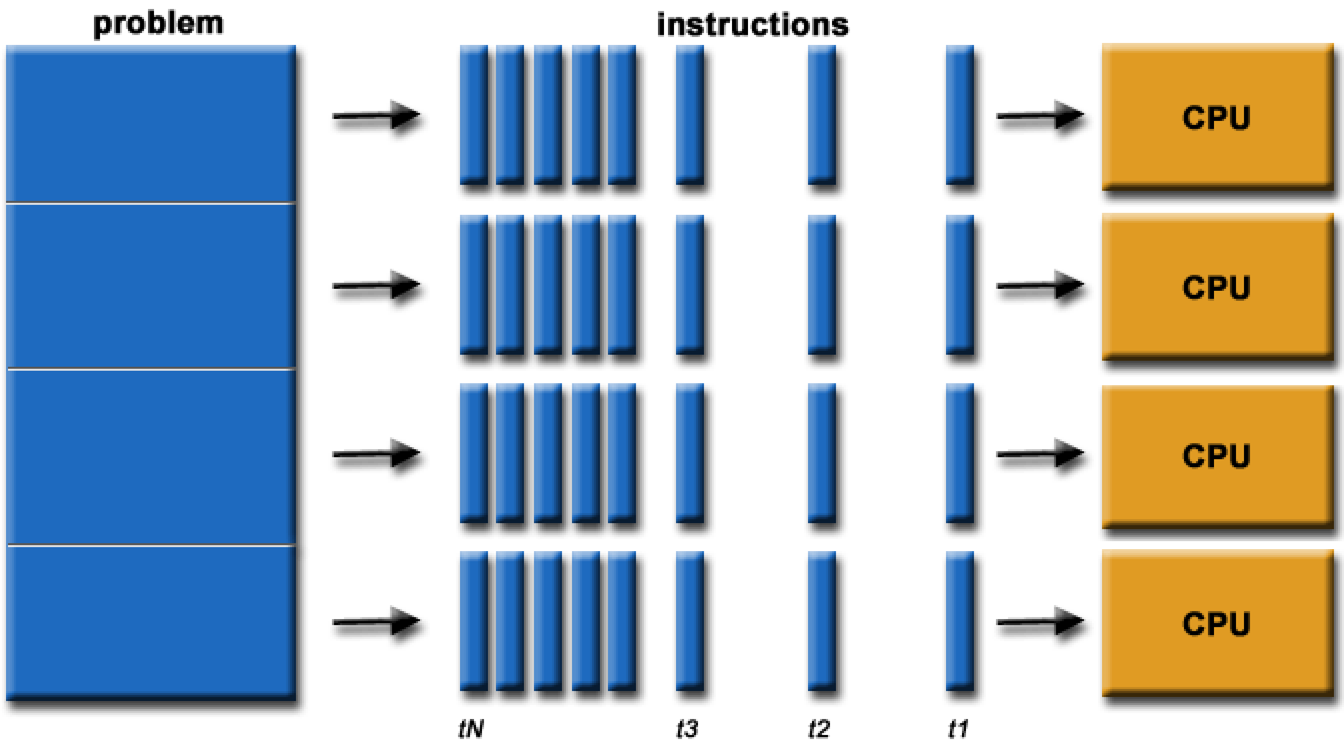
Serial Computation

Traditionally, software has been written for ***serial*** computation:

- To be run on a single Central Processing Unit (CPU)
- Break a problem into a series of instructions
- Execute the instructions in sequence



Parallel Computing



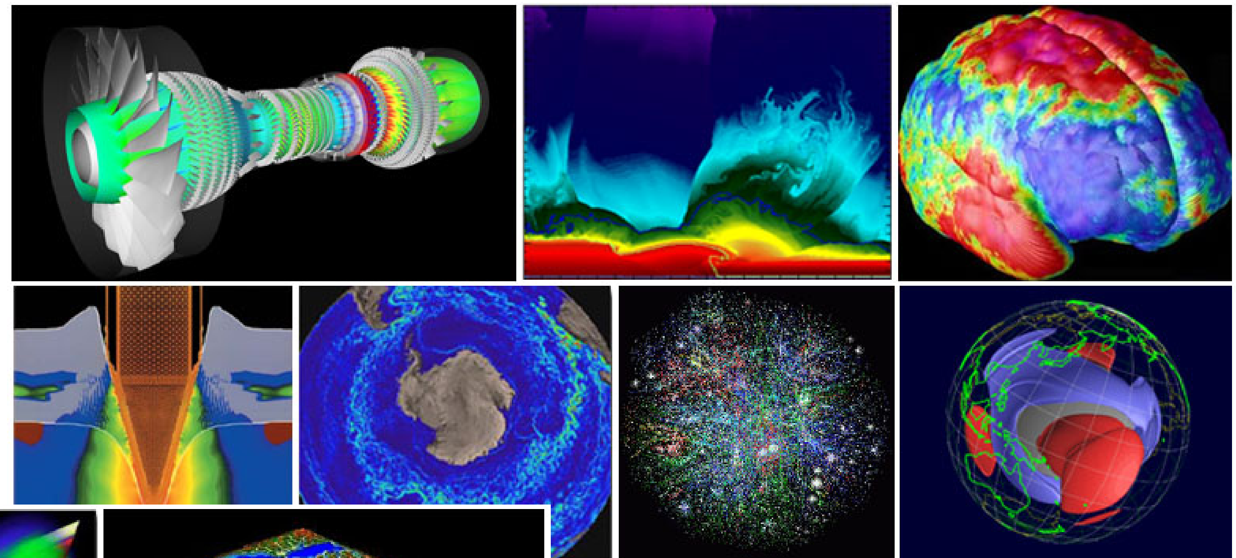
- The program is run on multiple CPUs .
- A problem is broken into discrete parts that can be solved concurrently .
- Each part is further broken down to a series of instructions.
- Instructions from each part execute simultaneously on different CPUs.

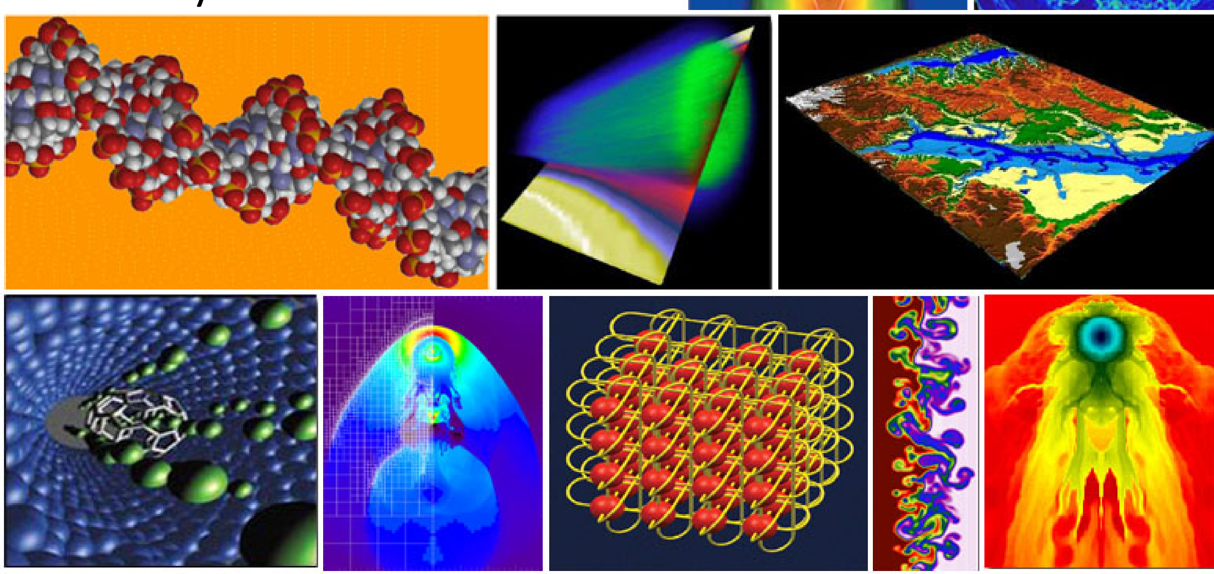
Why Parallel Computing

- Save (wall-clock) time: speedup
- Solve larger problems
- Provide concurrency
- Use of non-local resources
- Limits to serial computing
  - Transmission speeds
  - Limits to miniaturization
  - Economic cost

Parallel Computing Applications

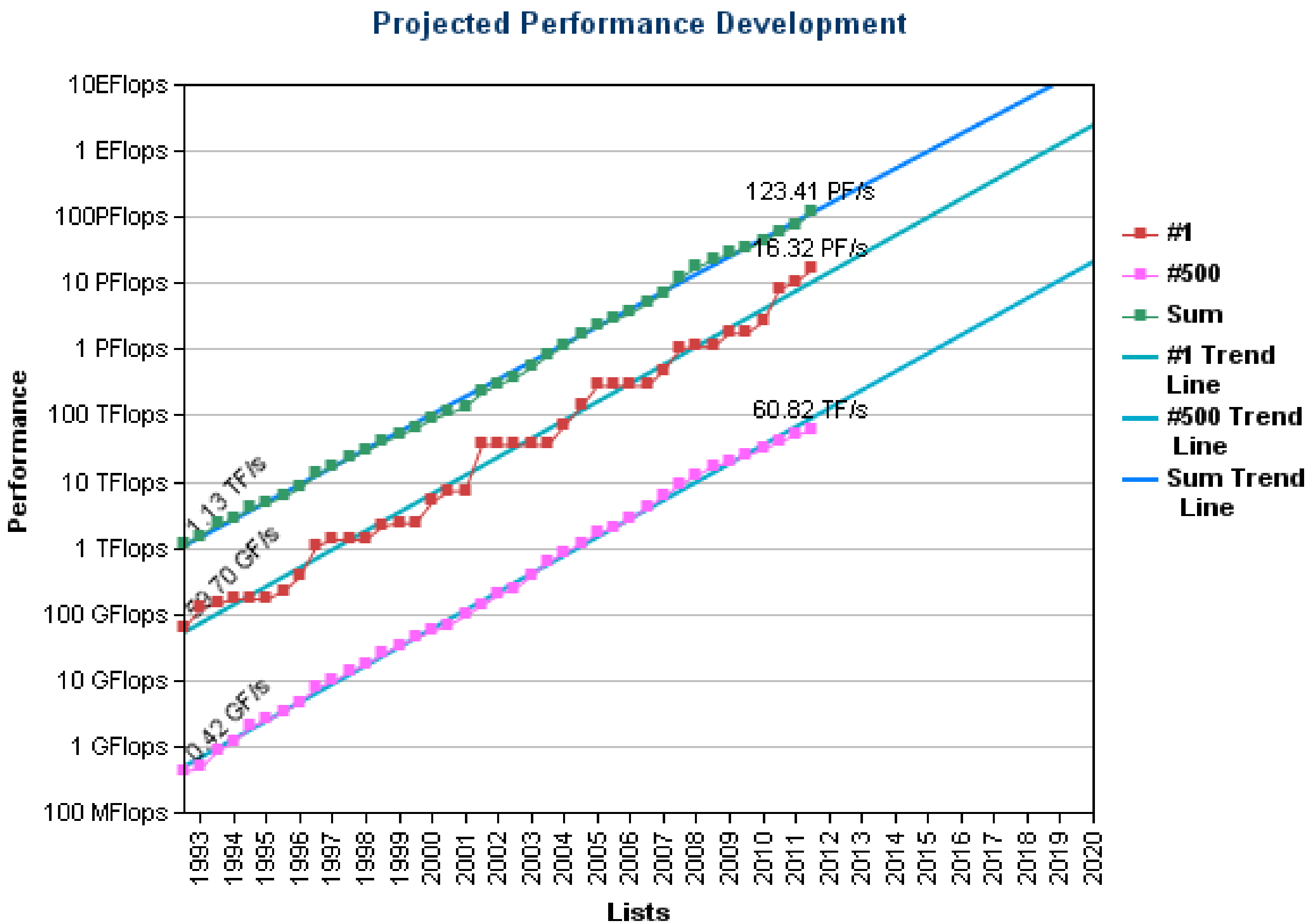
- Astronomy
- Bioscience
- Chemistry
- Computer Science,
- Electrical Engineering
- Geology
- Mechanical Engineering
- Mathematics
- Physics





- Collaborative work environments
- Databases, data mining
- Digital entertainment
- Financial engineering
- Oil exploration
- Pharmaceutical design
- Social networks
- Web search and services

Parallelism is the Future of Computing



Parallel Programming Models

- Shared Memory (without threads)
- Shared Memory with Threads
- Distributed Memory / Message Passing
- Data Parallel
- Hybrid
- Other higher-level programming models

Programming models are independent from underlying hardware architecture.

Shared Memory (without threads)

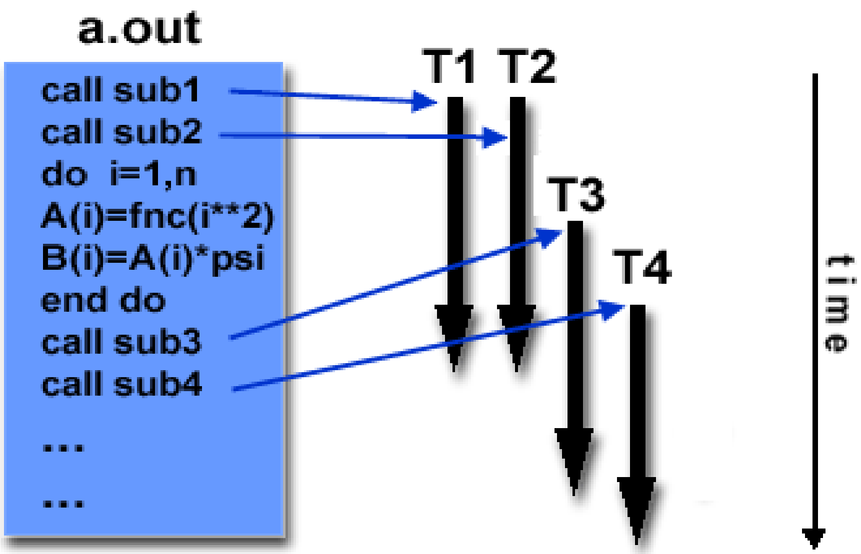
- Tasks share a common address space
- Various mechanisms such as locks / semaphores may be used to control access to the shared memory.

+: no need for data transfer between tasks

-: data locality is hidden

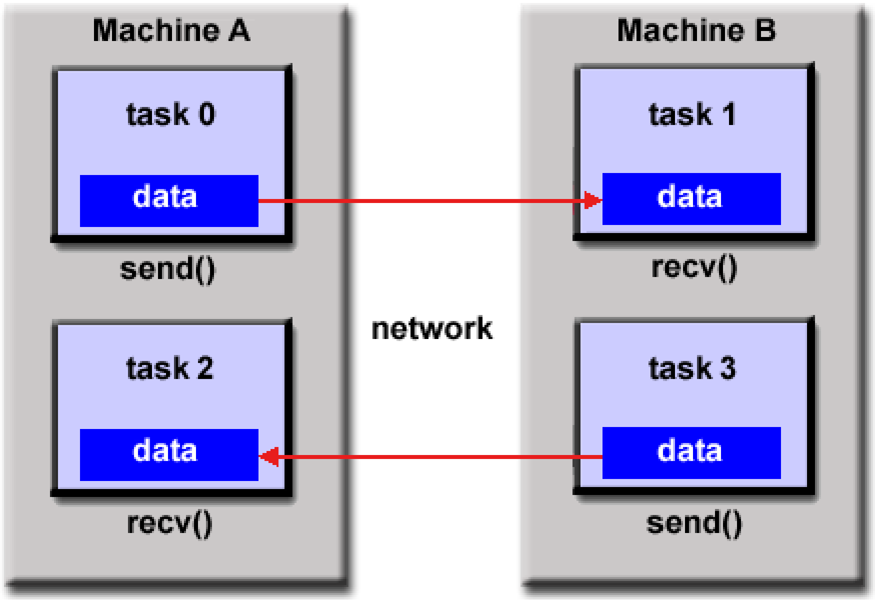
Shared Memory with Threads

- A single process can have multiple, concurrent execution paths.
- Threads communicate through shared memory.



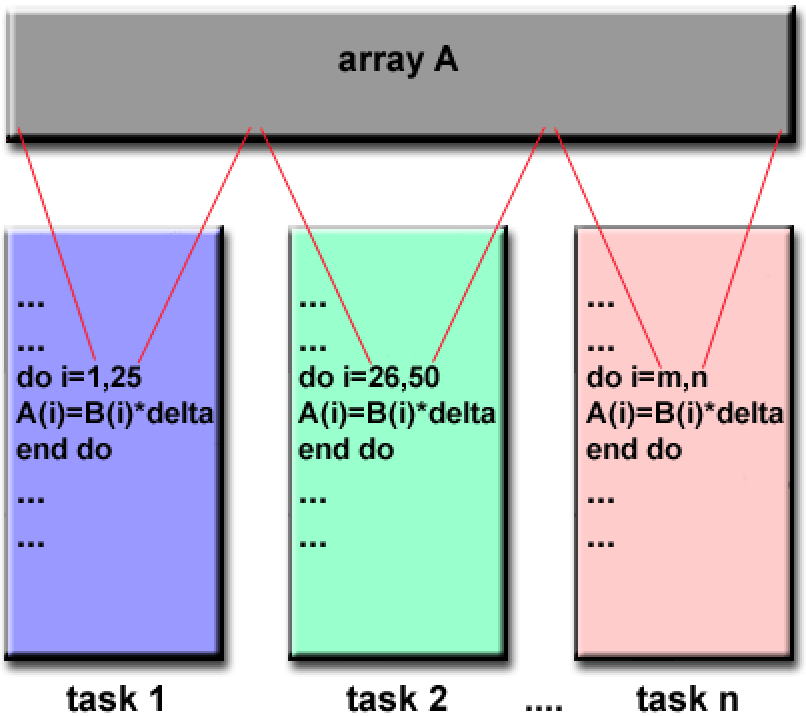
Distributed Memory/Message Passing

- Tasks use local memory during computation
- Tasks exchange data by sending and receiving messages



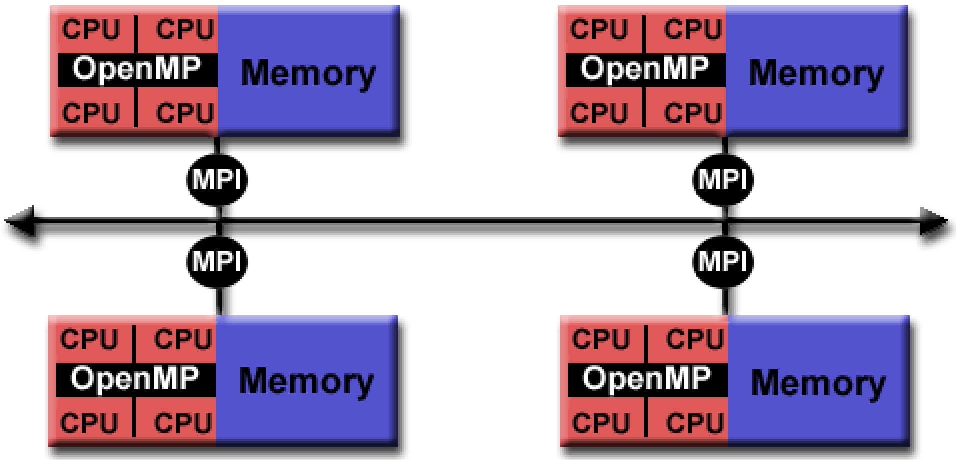
Data Parallel Programming

- A set of tasks work collectively on the same data structure, however, each task works on a different partition of the same data structure.
- Tasks perform the same operation on their partition of work



Hybrid Model

- A common example: the combination of the message passing model (MPI) with the threads model (OpenMP).

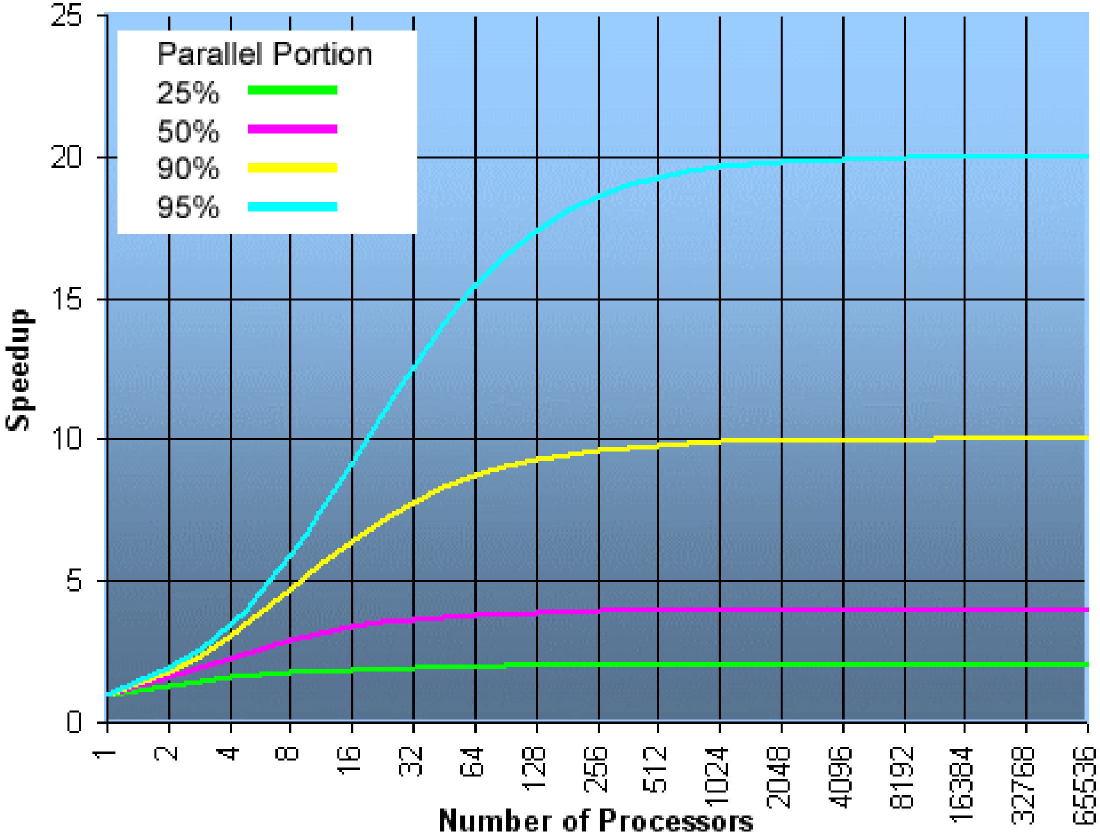


Amdahl's Law

- The maximum speedup of a program is defined by the fraction of code (P) that can be parallelized: Speedup = 1/(1-P)
- Suppose the parallel portion is shared by N processors, the maximum speedup is:

$$\text{Speedup} = 1/\ (P/N + (1-P))$$

Theoretical Speedup by Parallelism



Limits in Parallel Programming

- Inherent parallelism in the problem/solution
- Complexity in algorithms and implementation
- Portability: languages, OS, hardware
- Resources: CPU, memory, disk bandwidth
- Scalability: hardware, software

Summary

- Parallelism is the future of computing.
- There are several parallel programming models:
  - Shared memory (without threads), Threads, MPI, Data parallel, Hybrid, etc.
  - CUDA is mainly data-parallel programming.
- Parallel programming is a complex task with limitations.