GPU PROGRAMMING

INTEL CORP.

Hi!

M. Alfin Delvan Joeyantu

1

2

M. Zaky Mahdavikia Zein

Dimas Wahyu Saputro

3

4

Muhammad Dafha Syahrizal

Table of contents

01

Introduction to Python and Parallel Computing

Introduction to parallel computing

Working with a Supercomputer

User Experience

- Multiuser system
- Unix OS
- Optimized software

Compute power

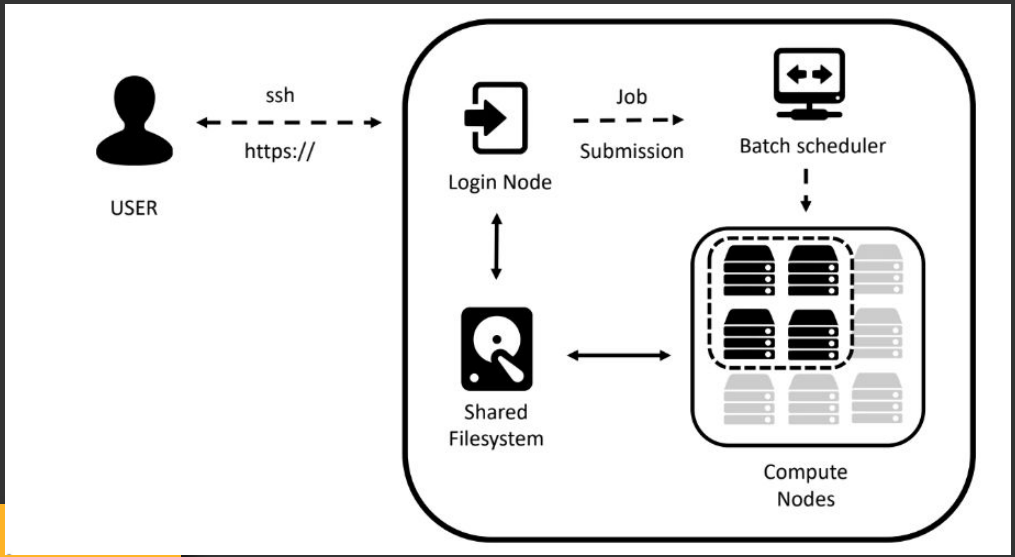
- Many CPUs system
- Specialized Hardware
- Low-latency/High bandwidth Connections

Storage

- Efficient I/O
- Large Memories

Introduction to parallel computing

Working with a Supercomputer



Service node(s)
- Editing and transferring files
- Compile programs
- Prepare simulations

Compute nodes
- Multicore architectures
- Large memory
- GPUs and other accellerators

Network
- High-speed interconnections
- Low latency
- High bandwidth

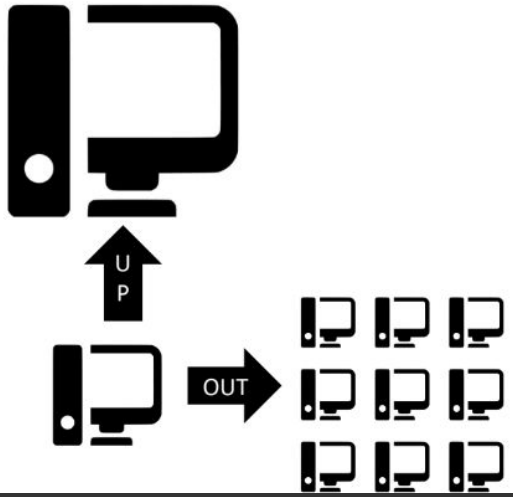
File system
- Parallel read/write
- Long term storage
- Node local disks

Introduction to parallel computing

Working with a Supercomputer

Why, or more, when you need a Supercomputer?

- **Scale up**
  - Faster CPUs
  - Large memories
  - Specialized Hardware/Software
- **Scale out**
  - Large parallel applications
  - Many small- to medium- size jobs



Introduction to parallel computing

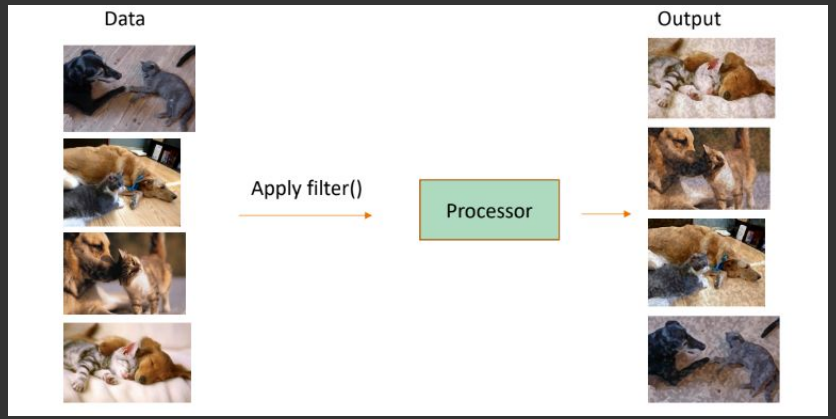
The Goal :

1. Merits and limits of parallel computing
2. Parallel programming models (task / data parallelism)
3. Differences between shared and distributed memory systems

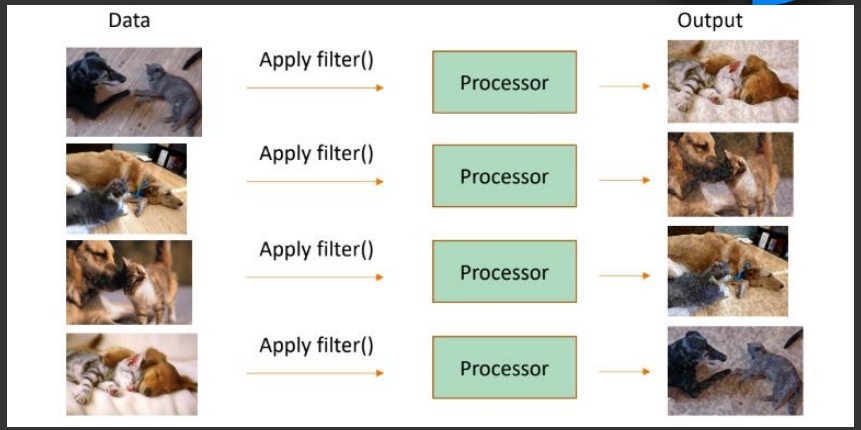
Introduction to parallel computing

Pararrel Computing

Multiple processors or computers working on a single computational problem



AND



Introduction to parallel computing

Pararrel Computing

Benefit :

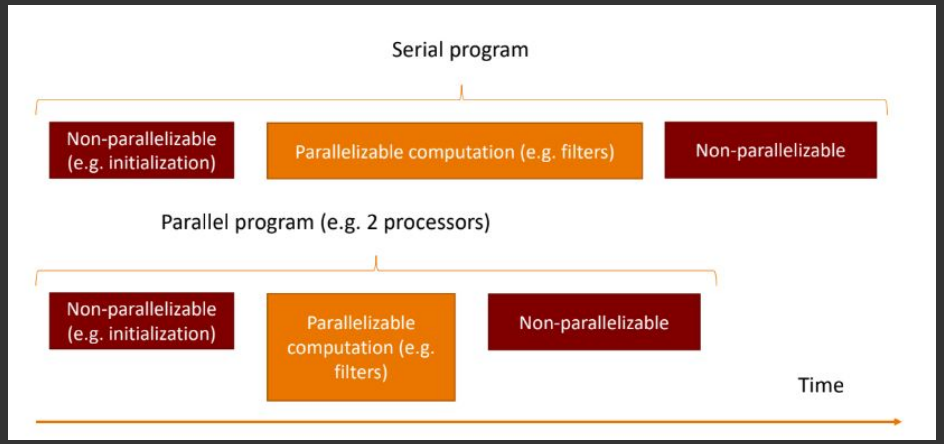
- Solve computationally intensive problems (speedup)
- Solve problems that don’t fit a single memory (multiple computers)

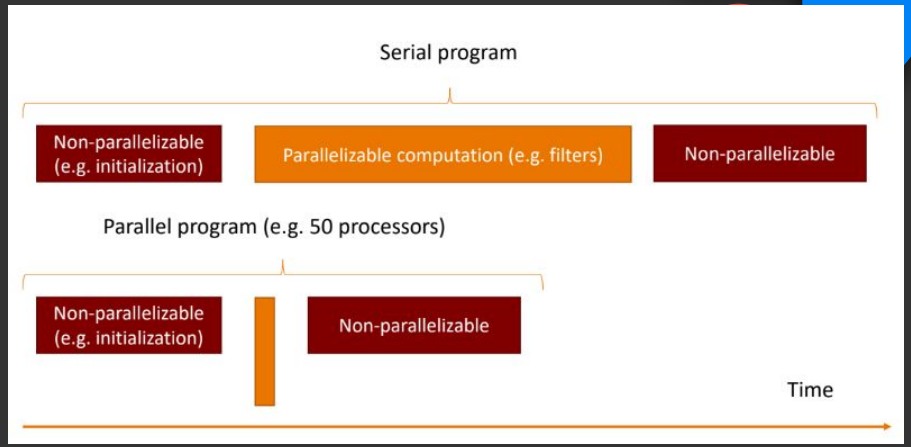
Requirements :

- Problem should be divisible in smaller tasks

Introduction to parallel computing

Pararrel Computing



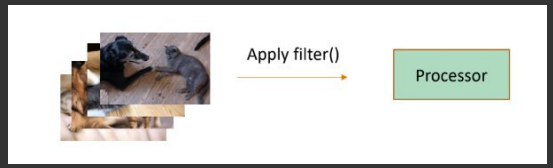


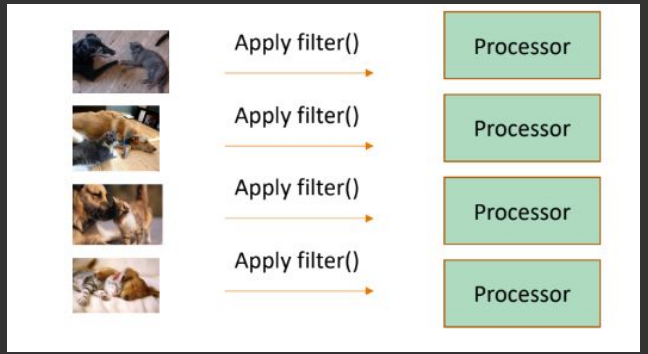
Introduction to parallel computing

Pararrel Computing

Strong scaling :

- Variation of solution time with #processors for fixed total problem size
- Possibility to run the same problem in shorter time





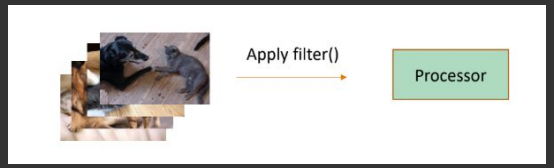
Introduction to parallel computing

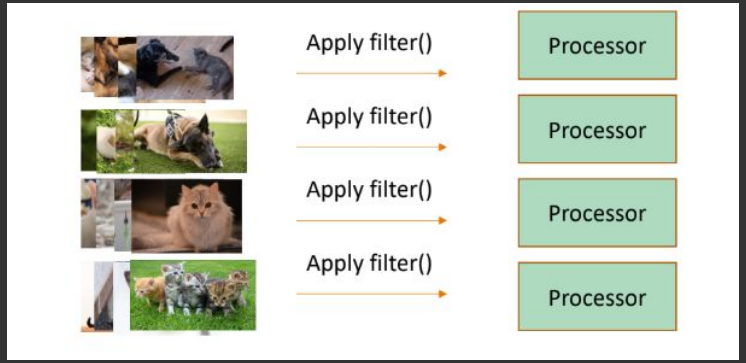
Pararrel Computing

Weak scaling :

• Variation of solution time with #processors for fixed problem size per
processor

•Possibilities to run a bigger problem in the same time



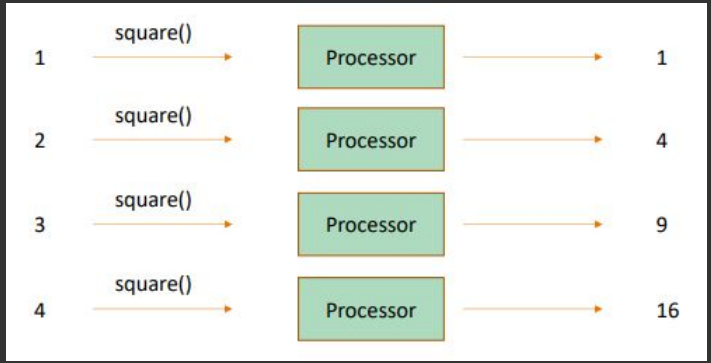


Introduction to parallel computing

Parallel programming models

Two well-known programming models

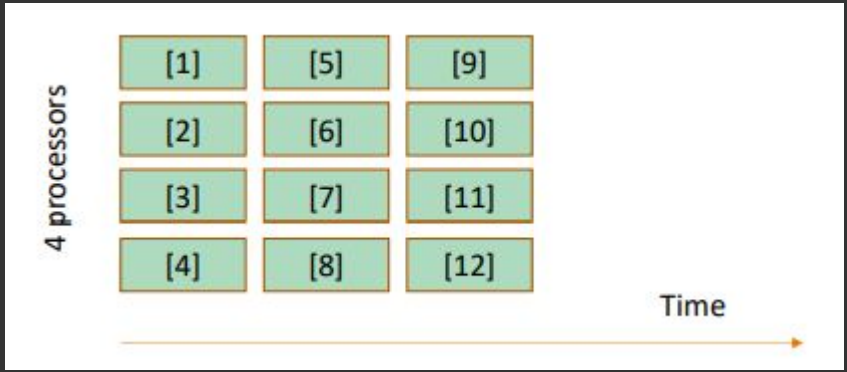
- Data parallelism
  - Each processor performs the same task on different data



Introduction to parallel computing

Parallel programming models

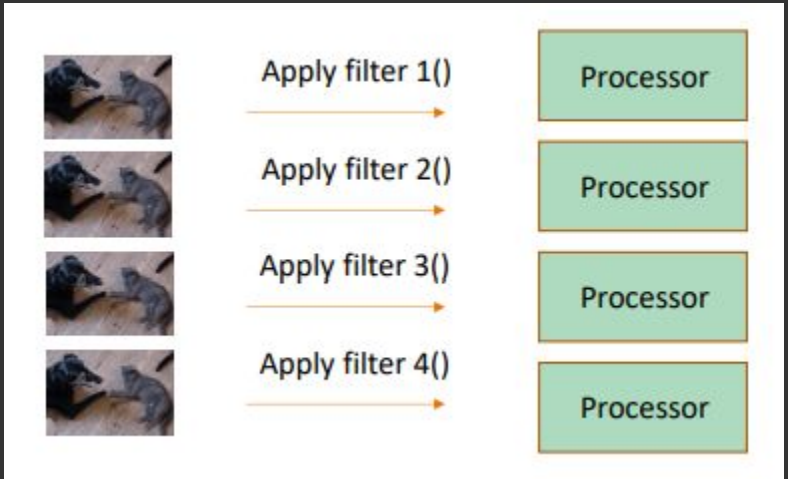
- Data parallelism
  - Amount of parallelization depends on input data size
  - Load balancing may be relatively easy
    - Same task on each data element
    - Approximately same time per element



Introduction to parallel computing

Parallel programming models

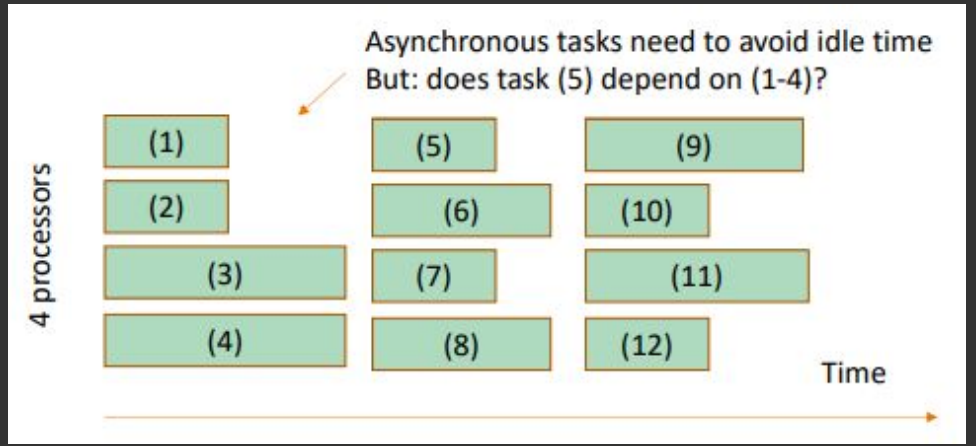
- Task parallelism
  - Each processor performs a different task on the same data



Introduction to parallel computing

Parallel programming models

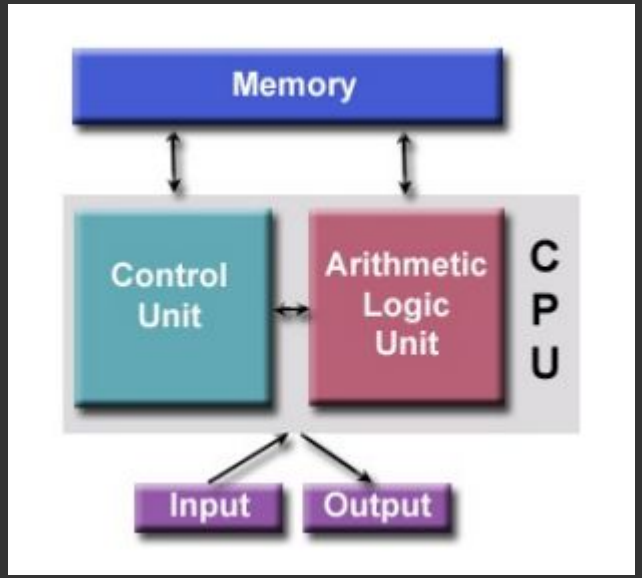
- Task parallelism
  - Amount of parallelization depends on the number of tasks
  - Load balancing can be very difficult
    - Heterogeneous tasks may be executed over the same data
    - Each task may take a very different amount of time



Introduction to parallel computing

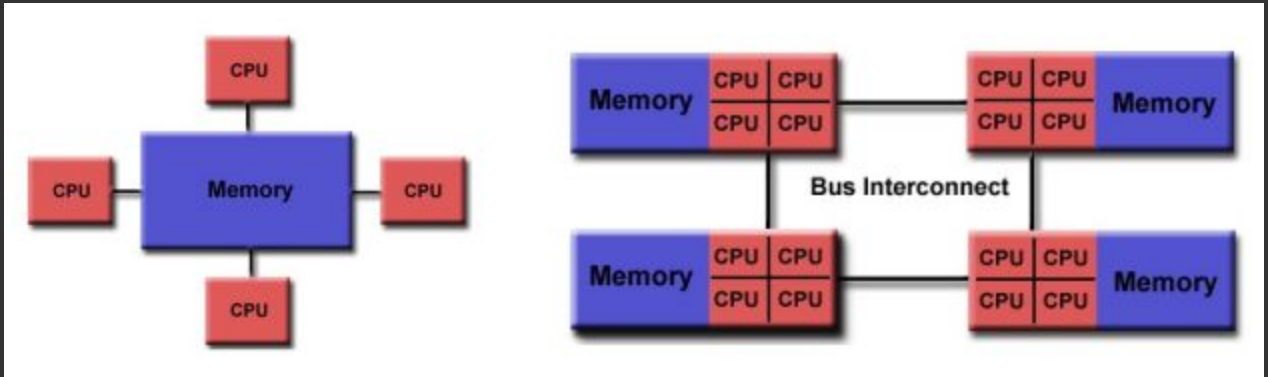
General computing architectures

- Four main components:



Introduction to parallel computing

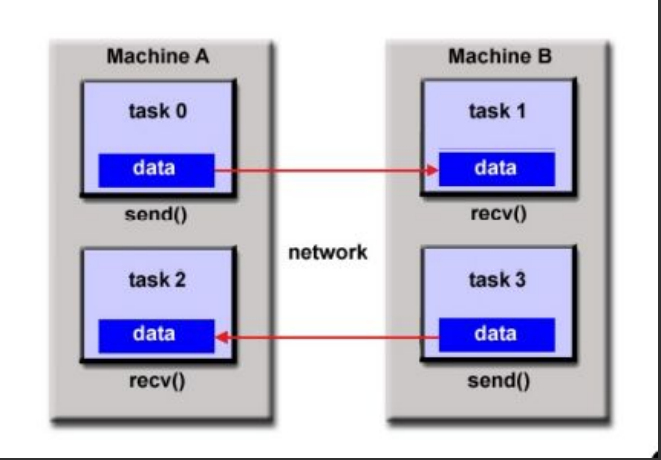
Parallel computing architectures



Shared memory

Introduction to parallel computing

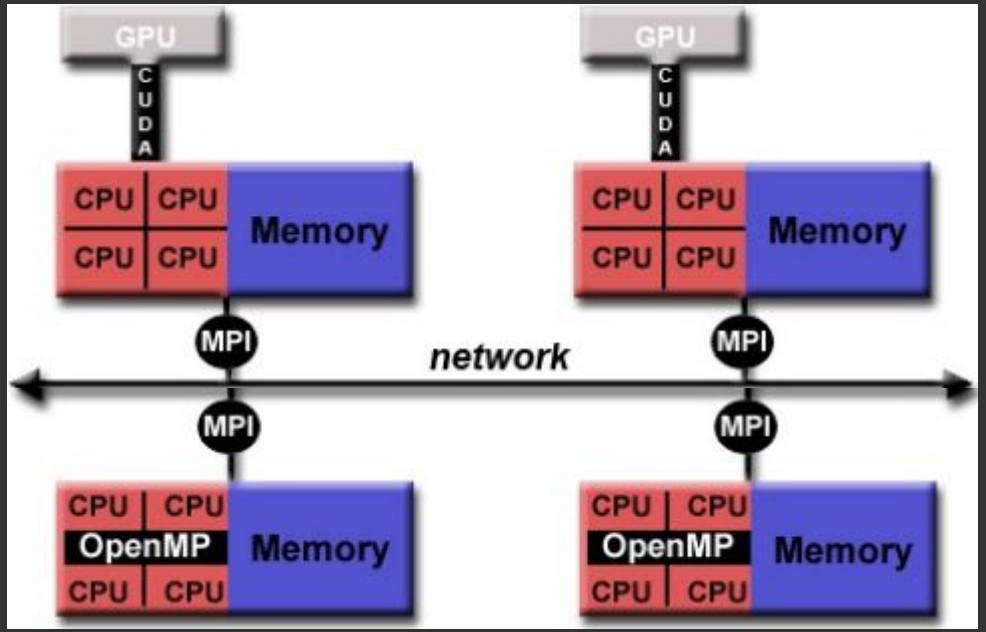
Parallel computing architectures



Distributed memory

Introduction to parallel computing

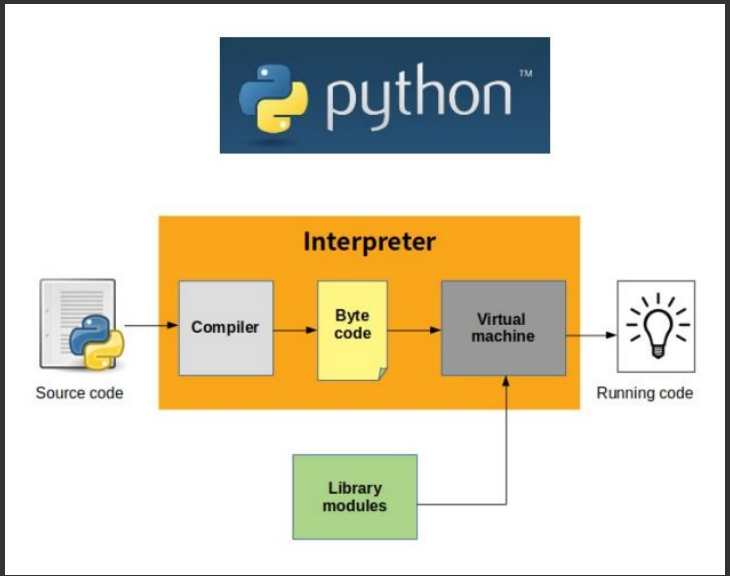
Parallel computing models



Hybrid model (e.g. Supercomputers)

Parallel computing with Python

Programming with Python



Parallel computing with Python

Python for High-Performance Computing

```
Python 3.8.2 (default, Jul 31 2020, 22:06:31)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.13.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: import os

In [2]: import numpy

In [3]: m = numpy.array([0,0,1])

In [4]: m.dot()
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-4-79e2982f1c0e> in <module>
----> 1 m.dot()

TypeError: dot() missing required argument 'b' (pos 1)

In [5]: b = numpy.array([0,1,0])

In [6]: m.dot(b)
Out[6]: 0

In [7]:
```

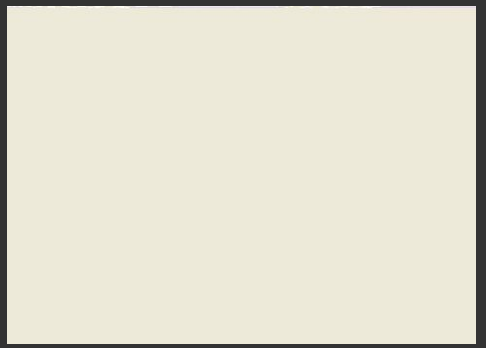
02

Parallel Programming for CPU Architectures in Python

Concurrent vs. parallel execution

Concurrent

Concurrency means executing multiple tasks at the same time but not necessarily simultaneously.



Parallel

Parallelism is the execution on multiple processors of two or more tasks that are running at the same time.

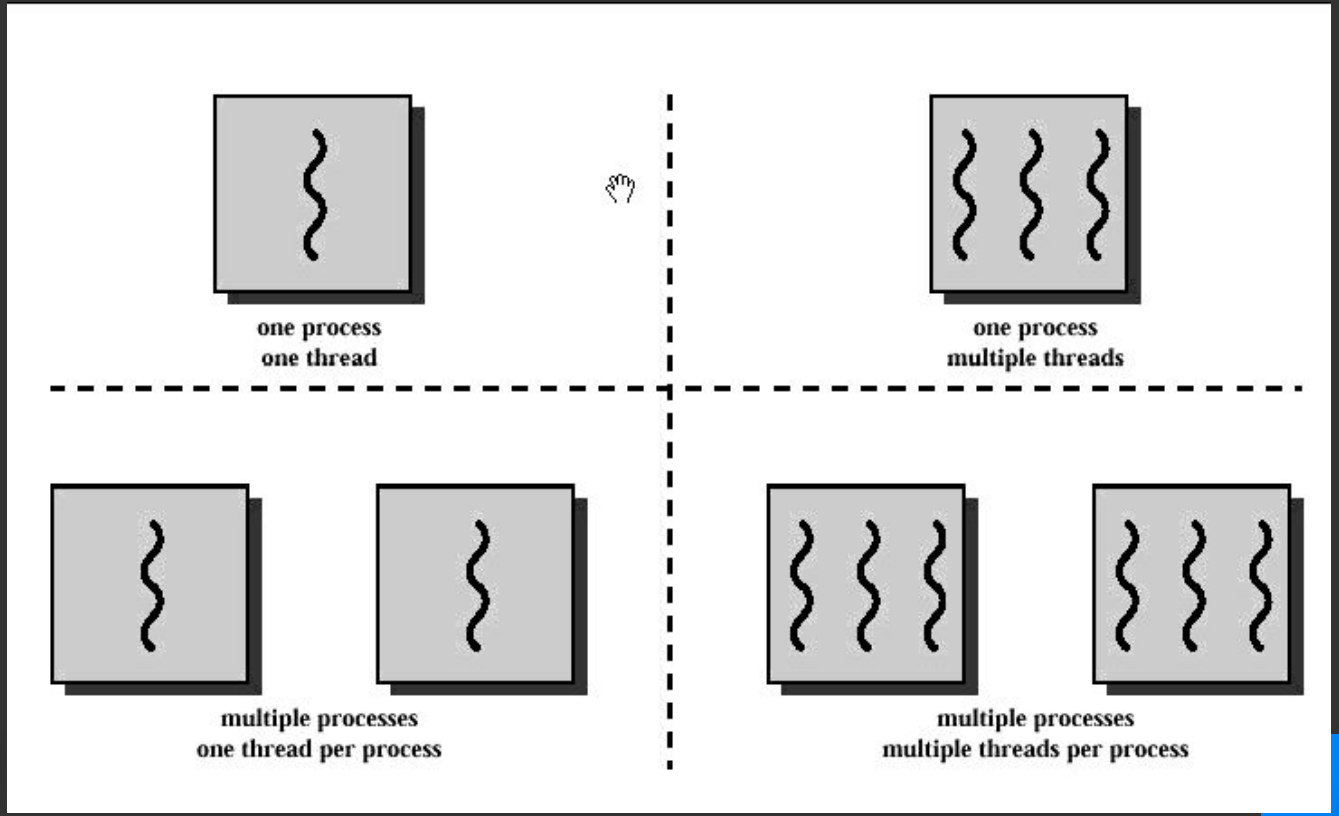
Advantages of concurrency

- Concurrent processes can reduce duplication
- The overall runtime of the algorithm can be significantly reduced
- More real-world problems can be solved than sequential algorithm alone.

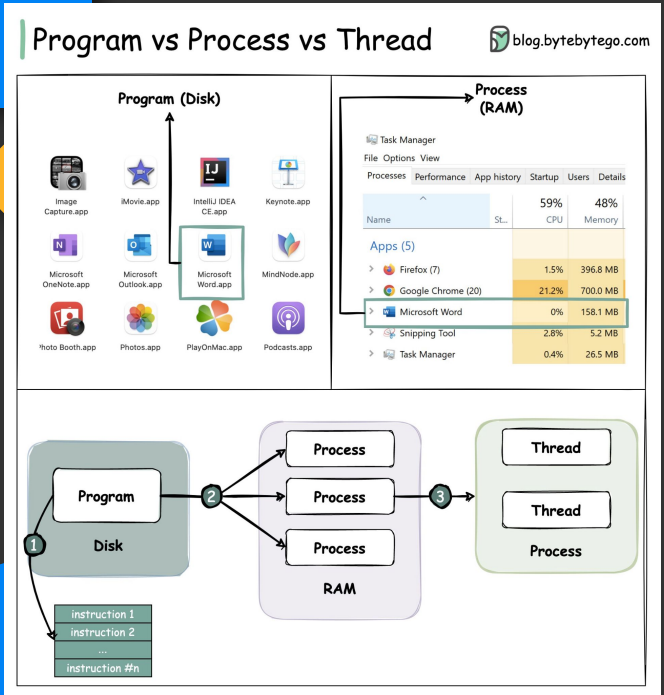
Disadvantages of concurrency

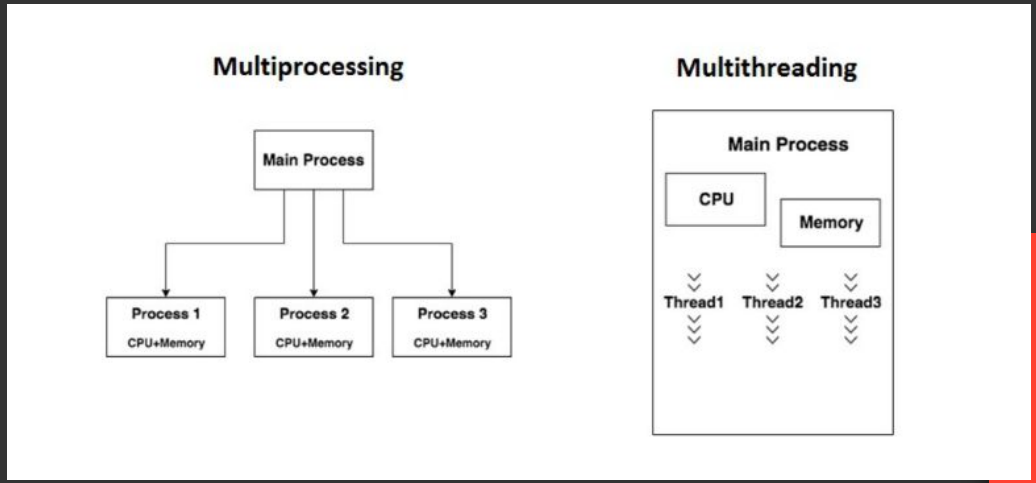
- Runtime is not always reduced, so careful planning is required.
- Concurrent algorithms can be more complex than sequential algorithms
- Communication between tasks is needed

Processes vs. Threads



Processes vs. Threads





Where to use what?

CPU and I/O Bounded Processes

- CPU bound process
  - process spends most of its time using processor
  - very long CPU bursts
  - compiler, simulator, scientific application
- I/O bound process
  - process spends most of its time using I/O
  - very short CPU bursts
  - word processors, database applications
- Processes usually either CPU or I/O bound
  - behavior may change over time (drastically)

I/O Bound

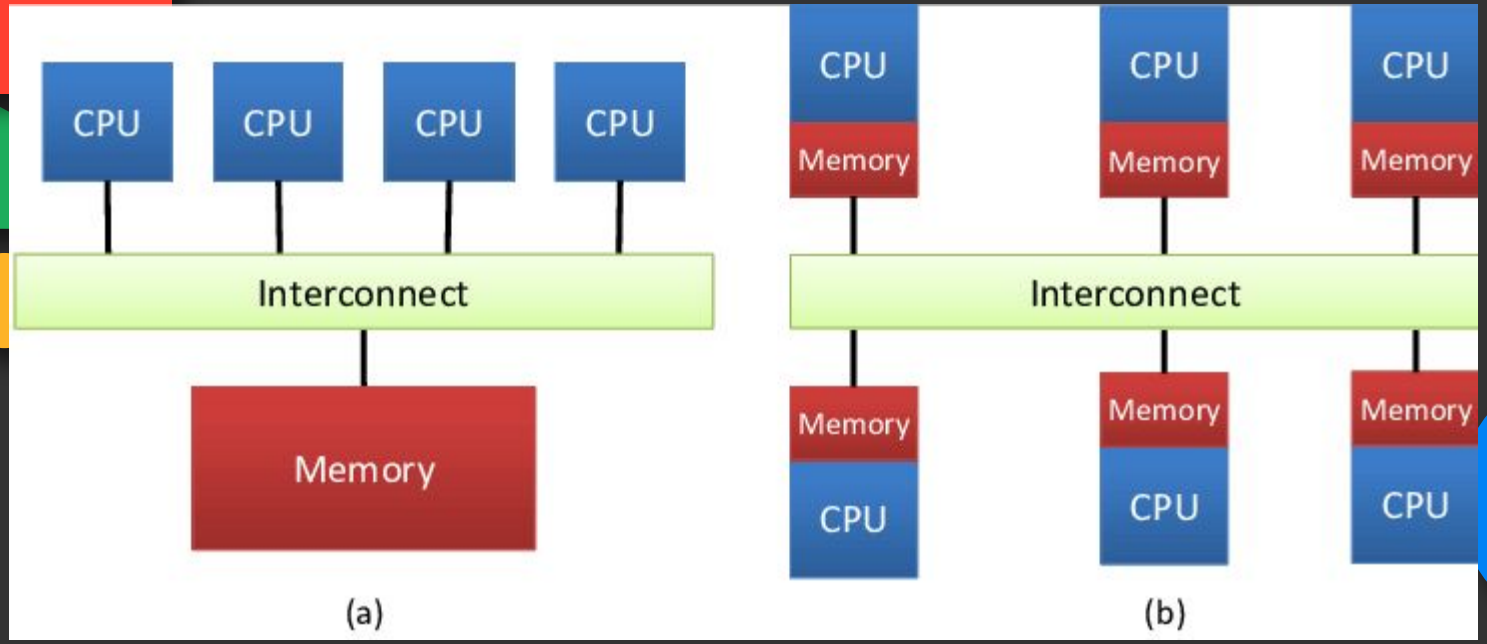
Multi Thread

CPU Bound

Multiprocessing

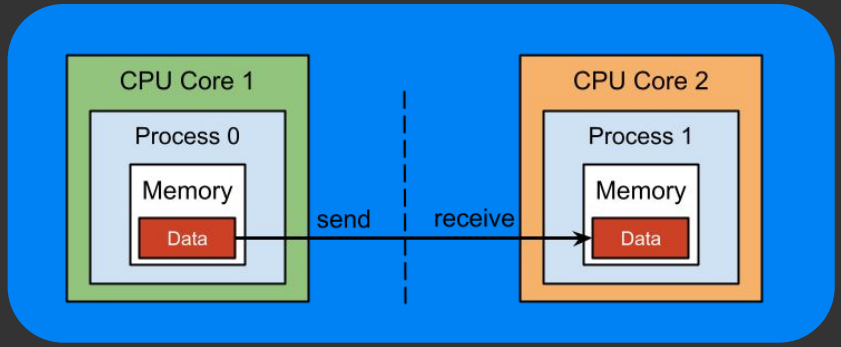
03

Distributed Memory Parallelism with mpi4py


CPU
CPU
CPU
CPU
Interconnect
Memory
(a)
CPU
CPU
CPU
Memory
Memory
Memory
Interconnect
Memory
Memory
Memory
CPU
CPU
CPU
(b)

What is MPI?

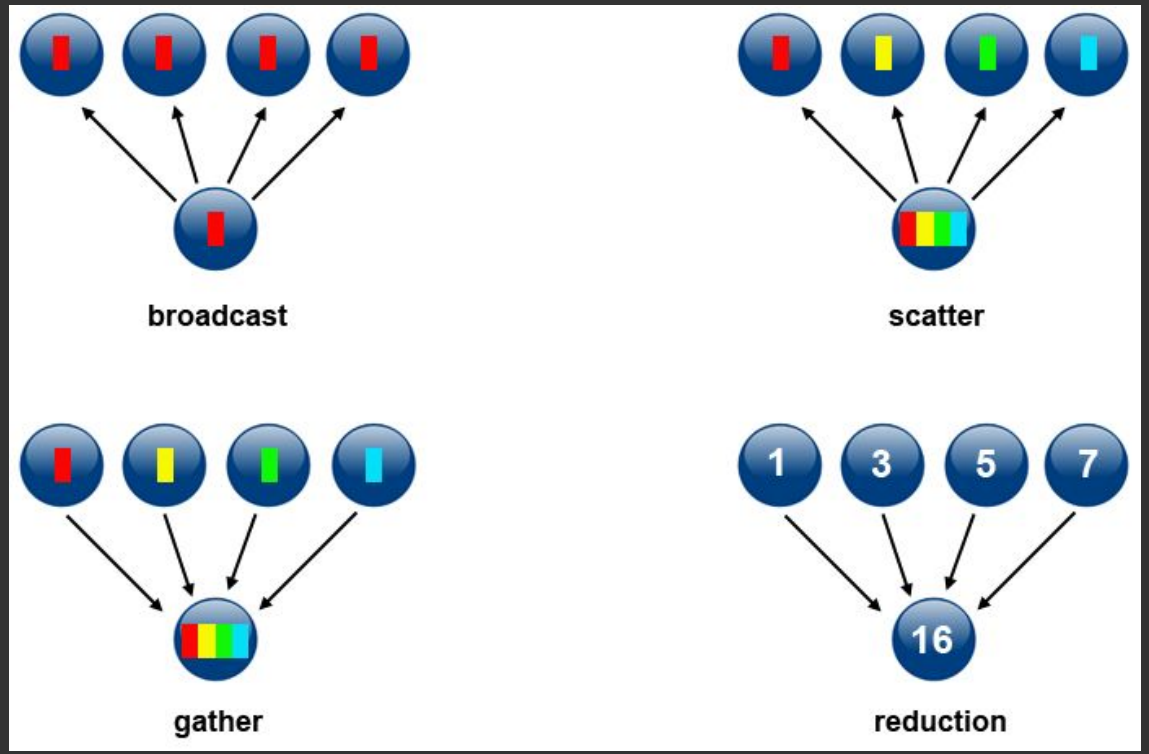
MPI stands for Message Passing Interface.
MPI calls are for sending messages between processes.
Initiative started in 1991; first release 1994.



MPI_COMM_WORLD

There is always a global communicator

Collective Communication



04

Graphic Processing Units (GPUs)

Moore's Law

Gordon Moore (co-founder of Intel) predicted in 1965 that the transistor density of semiconductor chips would double roughly every 18 months.

More transistors = more functionality. Improved technology = faster clocks = more speed. Thus, every 18 months, we obtained better and faster processors. They were all sequential: they execute one operationper clock cycle.

Graphic Processing Units (GPUs)

GPU is one of the main hardware components of current computer architecture (e.g: phones, game consoles, HPC!)

Originally designed to render images to display, modern graphic processing units are used for some of the most complex calculations, such as big data, ML/DL and AI.

Within the years GPUs evolved from single core, fixed functional hardware, into a set of programmable parallel cores. This started the era of General Purpose GPU (GPGPU).

Graphic Processing Units (GPUs)

HISTORY

- Before 1990's graphics were done on CPU's as well as framerate:
  reduced quality images
  No 3D rendering

- 1996 introduction Voodoo graphic chip one of the first video card for parallel work

- NVIDIA 1997 released the first chip to combine 3D acceleration with traditional 2D and video acceleration.

- 1999 first "GPU" with the GeForce 256

Graphic Processing Units (GPUs)

Tahun 80-an

Tahun 2022

GPU VS CPU

The difference with the CPU (the processor on the motherboard), the GPU is specifically designed to process image/graphic data so that it can be displayed to the monitor quickly and precisely. Because the GPU has a special ability, namely to perform complex mathematical calculations to form an image (geometry) in 3-D space, which is impossible for the CPU to handle.

The CPU only has a few processor cores (dual-core, quad-core, octa-core) that work in serial. While the GPU consists of THOUSANDS of tiny cores that work in parallel.

This is what makes GPUs able to solve math problems quickly.

GPU VS CPU

- CPU for ”heavy” tasks with complex control flows and branching (VM, branching, security, etc.)
- GPU do one thing well: handling billions of repetitive ”light” tasks
- GPUs have 1000s of ALU compared to traditional CPUs that are commonly build with 4-8

GPU VS CPU

CPU strengths	GPU strengths
Very large main memory	High bandwidth main memory
Very fast clock speeds	Latency tolerant via parallelism
Latency optimized via large caches	Significantly more compute resources
Small number of threads can run very quickly	High throughput
	High performance/watt

CPU weaknesses	GPU weaknesses
Relatively low memory bandwidth	Relatively low memory capacity
Low performance/watt	Low per-thread performance

CUDA

Introduced by NVIDIA in 2006

CUDA is a parallel computing platform and programming model developed by NVIDIA for general computing on its own GPUs (graphics processing units). CUDA enables developers to speed up compute-intensive applications by harnessing the power of GPUs for the parallelizable part of the computation

CUDA Streaming Multiprocessor executes threads in warps (32 threads) There is a maximum of 1024 threads per block (for our GPU) There is a maximum of 1536 threads per multiprocessor (for our GPU)

CUDA

The GPU is viewed as a compute device that:

- Has its own DRAM (device memory)
- Runs many threads in parallel
- Data-parallel portions of an application are executed on the device as kernels which run in parallel on many threads

Thanks!