STA 141C - Big Data & High Performance Statistical Computing

Lecture 16: Parallel computing

Instructor: Bo Y.-C. Ning

March 02, 2022

Announcement

Last time

- HW4 & Final presentation

Today

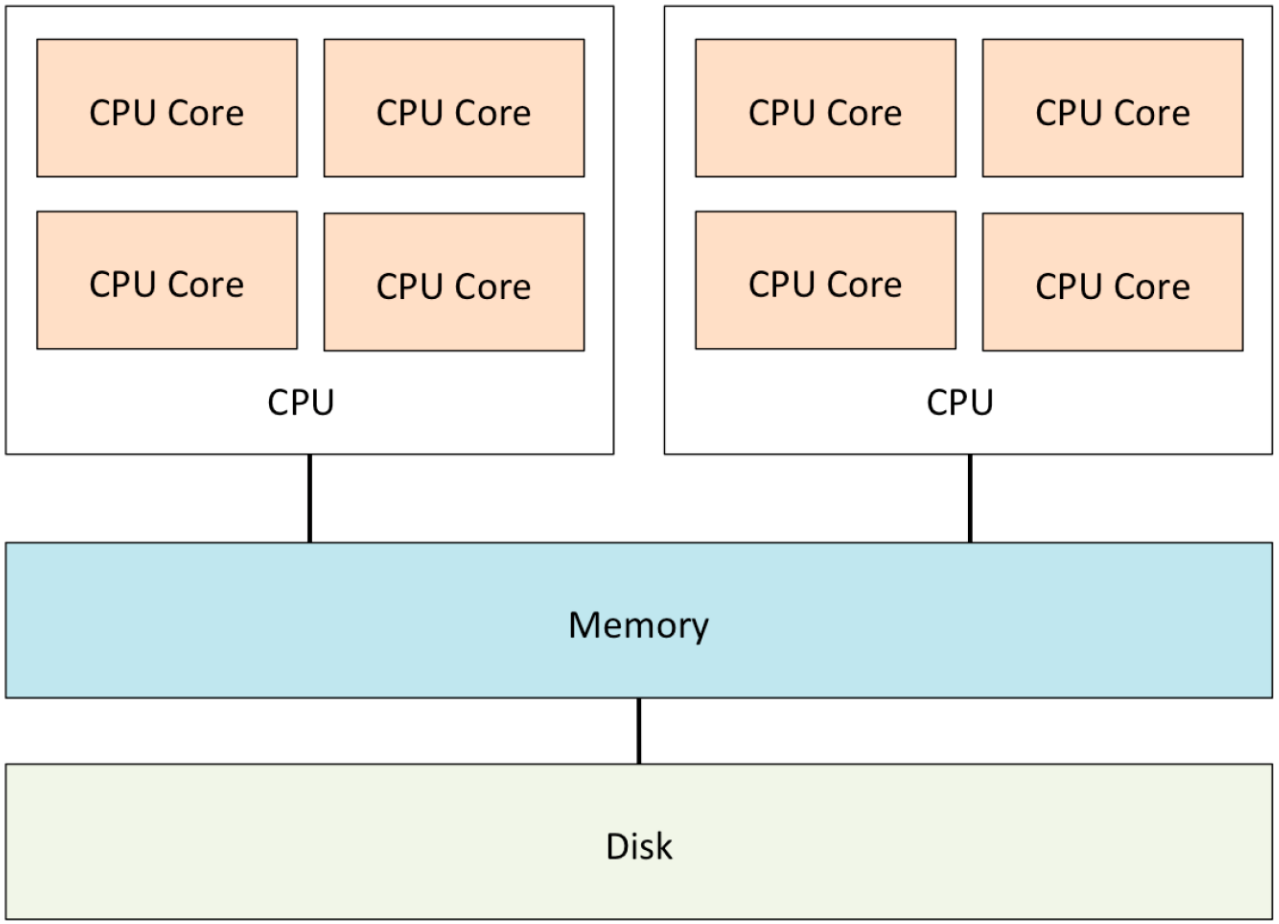
- Parallel computing

References

- Prof. Cho-Jui Hsieh's old lecture note on parallel computing
- The two websites: [web1] [web2]

Computer architecture

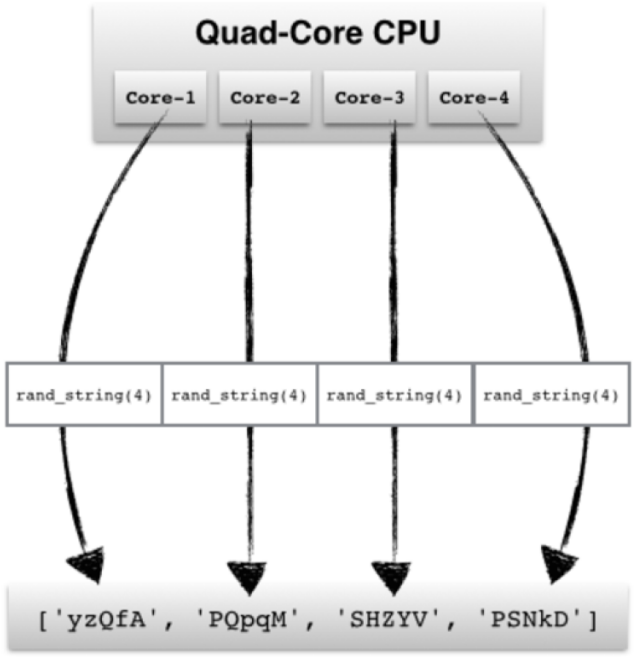
- Each computer can have multiple CPUs, each CPU has multiple cores (e.g., two quad-core CPUs)
- All the CPUs are connected to memory (e.g., 64G memory)
- CPU cores can execute in parallel



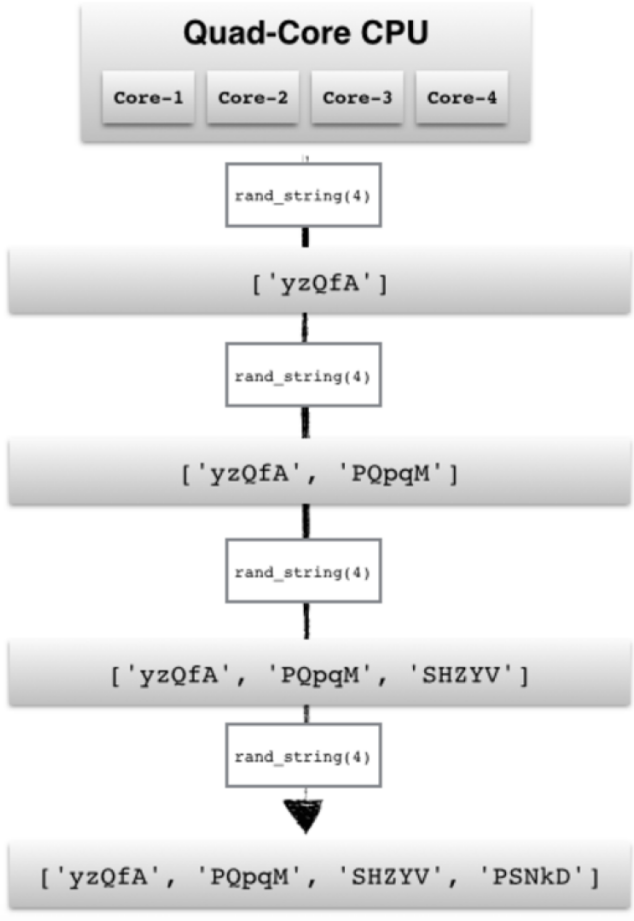
Multicore programming

- Execute tasks simultaneously on many CPU cores

[parallel processing]

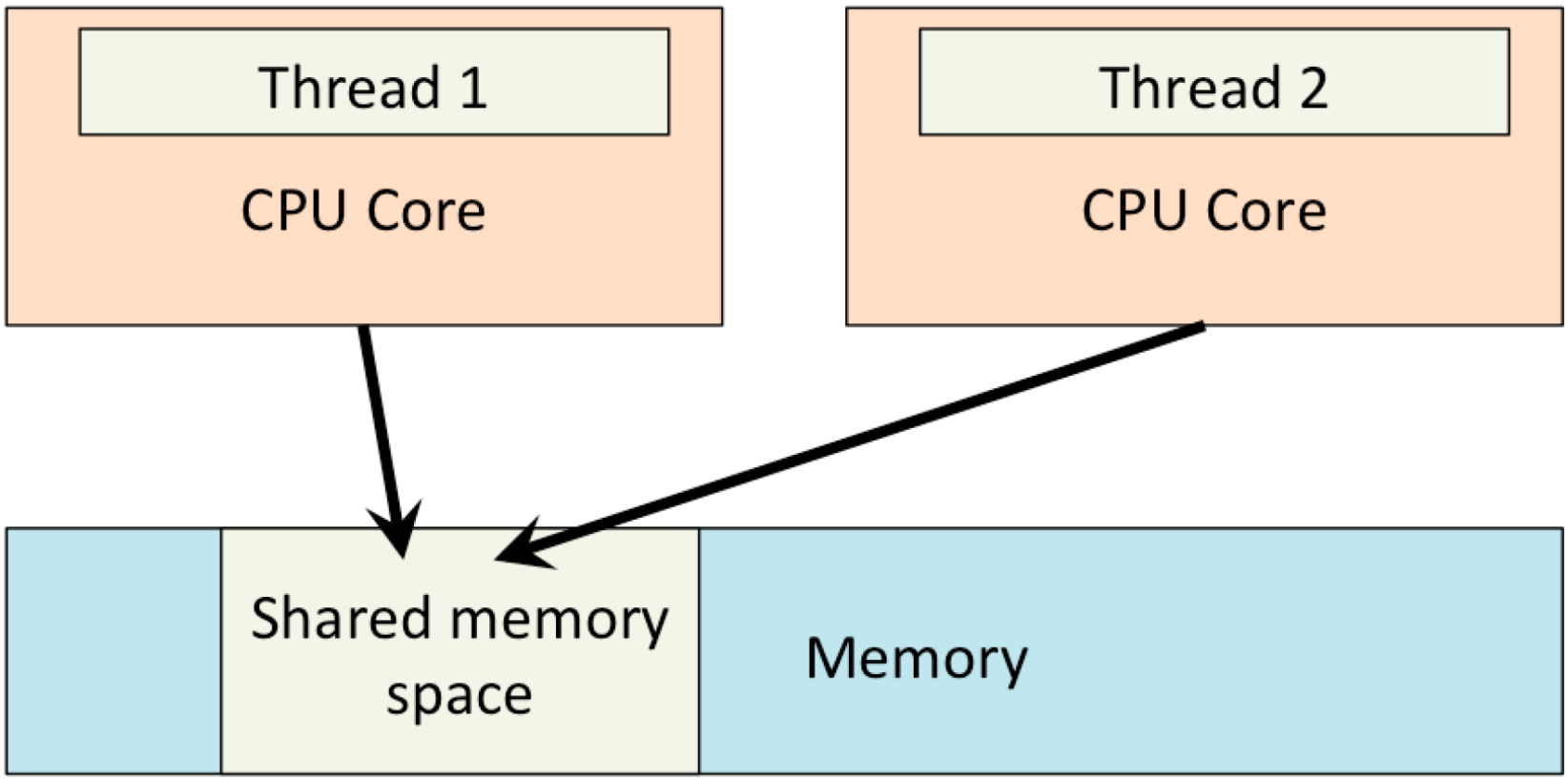


[serial processing]



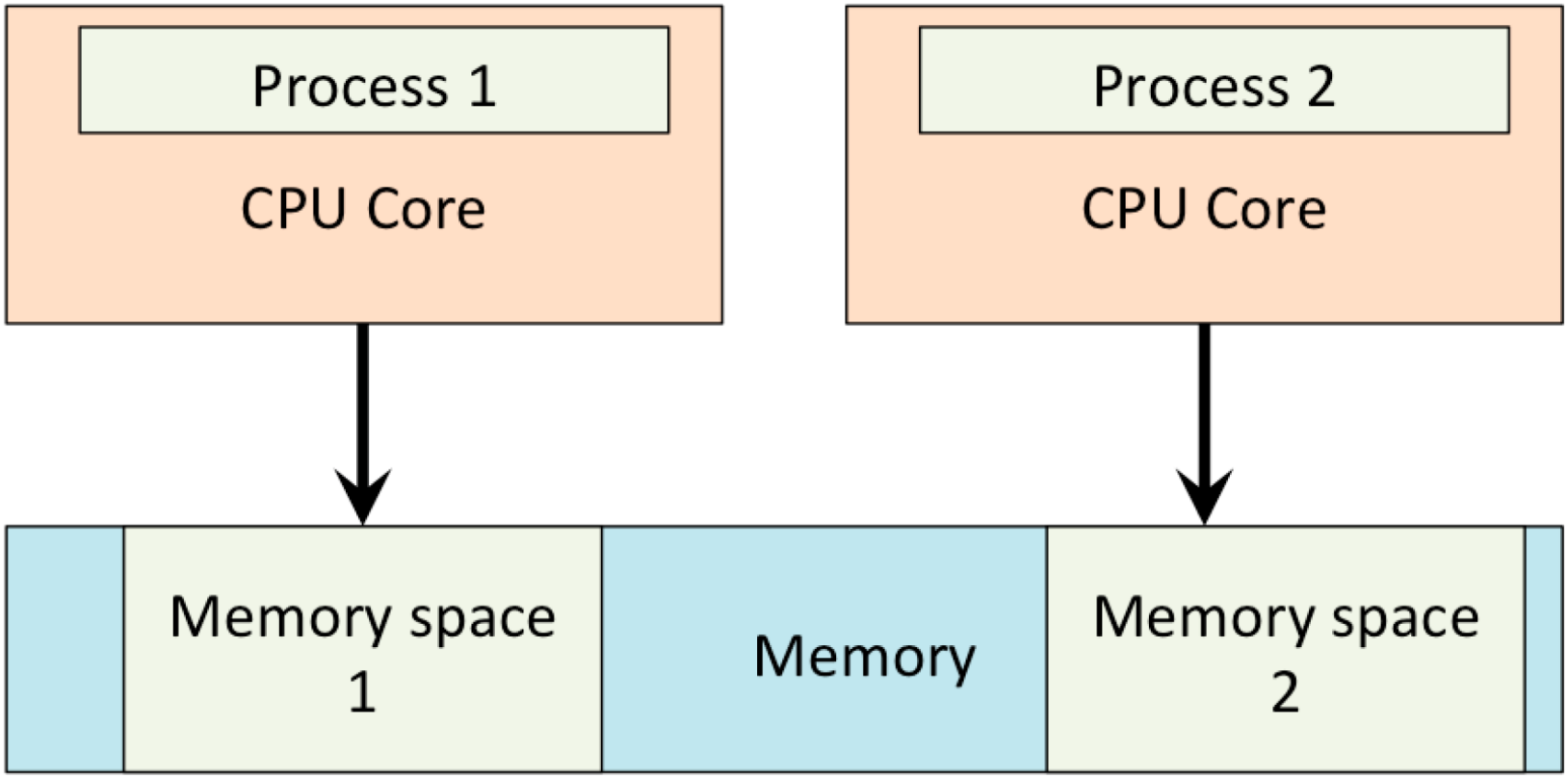
What is a thread?

- Multiple threads share the memory.
- Don't need inter-process communication.
- They are "light-weighted" (not much overhead to fork multiple threads)



What is a process?

- Processes are “share nothing” (independent executing without sharing memory or state)
- Easier to turn into a distributed application.



Python threads

- Package “threading”
- Unfortunately, python only allows a single thread to be executing at once - due to GIL (global interpreter lock)
- Usually no or little speedup - only useful when you want to interleave I/O and CPU execution

Python processes

- Package "multiprocessing"
- You can create multiple processes
  - Automatically run on multiple CPU cores
  - Default no shared memory, each process has its own memory space (larger memory overhead)
  - Can declare some part of memory to be shared (but often harder to use)
- You can also check some tutorials: [T1] [T2]

Example

```
import multiprocessing as mp

def helloworld(x):
    print('Hello World %d\n'%x)

# Setup a list of processes
plist = []

for x in range(4):
    plist.append(mp.Process(target=helloworld, args=(x, )))

# Run processes
for p in plist:
    p.start()

# Exit the completed processes
for p in plist:
    p.join()
```

Output

- Output from the program:

  Hello World 0

  Hello World 1

  Hello World 2

  Hello World 3

- **Process(target=helloworld, args=(x,))**:
  - Specify the target function to run (helloworld)
  - Specify the input argument of the function (only one argument x)
  - Create an object belongs to Process type
- **process.start()**: The process starts to run when execute
- **process.join()**: The process terminates

Exchanging objects between processes

Multiprocessing supports two types of communication channel between processes:

- Queue

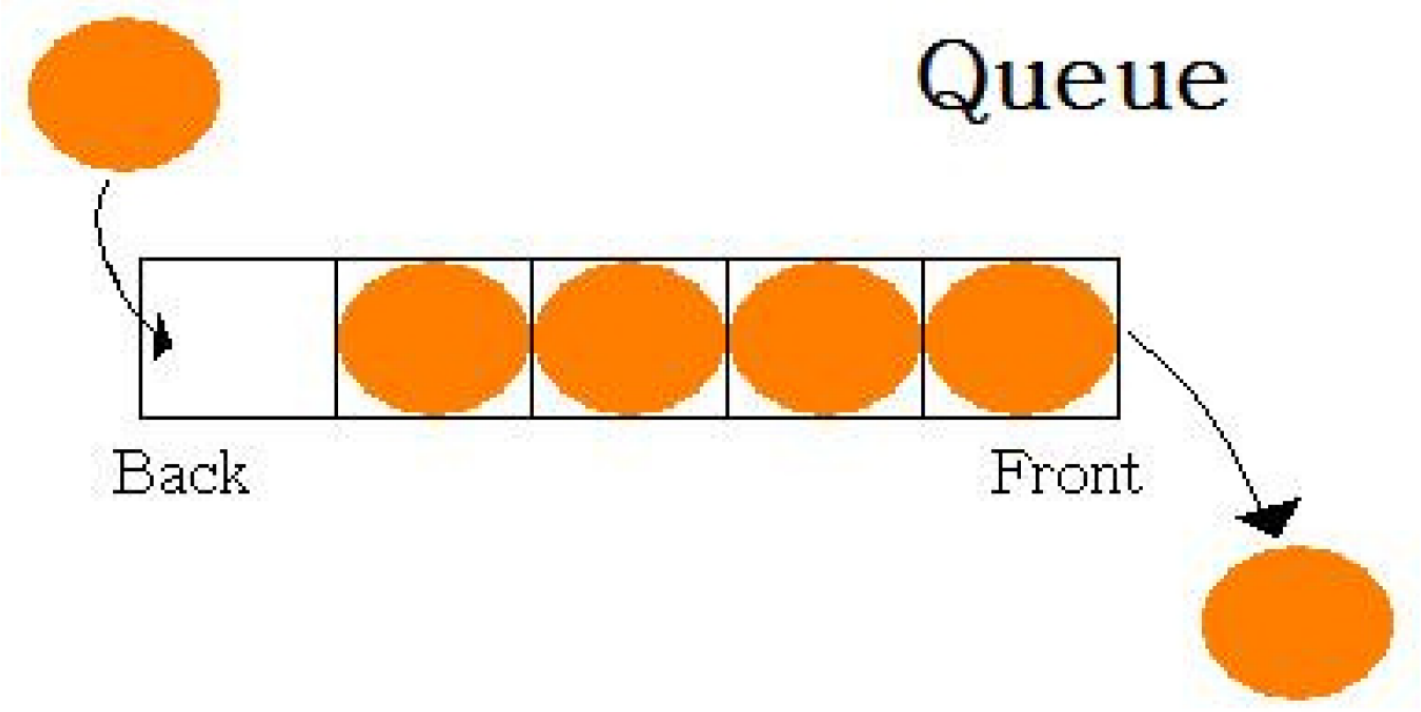
```
from multiprocessing import Process, Queue

def f(q):
    q.put([42, None, 'hello'])

if __name__ == '__main__':
    q = Queue()
    p = Process(target=f, args=(q,))
    p.start()
    print(q.get())    # prints "[42, None, 'hello']"
    p.join()
```

Using mp.Queue

- mp.Queue is a concurrent and "first in first out" data structure
- Can be used to communicate, or gather the results from the processes
- Queue.put(): insert an object to the end of queue
- Queue.get(): remove the first element in the queue
- Queue.empty: check whether the queue is empty



Using mp.Pool

- mp.Pool: another and more convenient approach for parallel processing in python.
- Use mp.Pool(processes=4) to create 4 processes
- Use $[r_1, r_2, \ldots, r_k]$ = pool.map(f, $[x_1, x_2, \ldots, x_k]$) to run multiple processes and get the results
  - f is the function to run for the processes
  - $[x_1, x_2, \ldots, x_k]$ are the input arguments we want to run for the function (this is a size k list)
  - $[r_1, r_2, \ldots, r_k]$ are the output arguments we get after running the functions for each input (this is a size k list)
  - k may be larger than number of processes

spark for machine learning methods

Spark is great for scaling up data science tasks and workloads.

To run spark, you need to use Spark data frames and libraries that operate on these data structures, you can scale to massive data sets that distribute across a cluster.

There are three different ways of achieving parallelization in PySpark

- Native Spark: if you?re using Spark data frames and libraries (e.g. MLlib), then your code we?ll be parallelized and distributed natively by Spark
- Thread Pools: The multiprocessing library can be used to run concurrent Python threads, and even perform operations with Spark data frames.
- Pandas UDFs: A new feature in Spark that enables parallelized processing on Pandas data frames within a Spark environment.

See example in the Spark_example.py file attached to this slides.

Applying parallel computing in R? See this [Link]