Parallel Machine Learning and Artificial Intelligence

Dr. Handan Liu

h.liu@northeastern.edu

Northeastern University

Parallel in Python

Preface

- A number of Python-related libraries exist for programming in parallel on:
  - multiprocessors in a symmetric multiprocessing (SMP) or shared memory environment, or
  - potentially huge numbers of computers in a cluster or supercomputing environment.

Symmetric Multiprocessing

- Some libraries employ parallel processing techniques which accommodate their relevance to SMP-based hardware.
  - Multiprocessing
    - ✔ process-based parallelism using either fork on Unix or the subprocess module on Windows
    - ✔ https://docs.python.org/dev/library/multiprocessing.html#module-multiprocessing
  - Joblib
    - ✔ a set of tools to provide lightweight pipelining in Python.
    - ✔ easy simple parallel computing (single computer)
    - ✔ https://joblib.readthedocs.io/en/latest/generated/joblib.Parallel.html
  - pp (parallel python)
    - ✔ process-based, job-oriented solution with cluster support (*Windows, Linux, Unix, Mac*)

Cluster Computing

- Unlike SMP architectures and especially in contrast to thread-based concurrency, cluster architectures offer high scalability via networking.
- Libraries:
  - Ray: https://ray.io/
  - Dask: https://dask.org/
  - mpi4py: https://mpi4py.readthedocs.io/en/stable/
  - pp: https://www.parallelpython.com/
  - .....

Cloud Computing

- StarCluster
  - StarCluster is an open source cluster-computing toolkit for Amazon's Elastic Compute Cloud (EC2).
  - http://star.mit.edu/cluster/
- Google App Engine
  - Supports Python.
  - https://cloud.google.com/appengine/

Python fork()

- Unix/Linux: fork()
  - The parent process: return the child process ID
  - The child process: return 0; using getppid() to obtain the parent process ID

- In Python, os module encapsulates common system calls, including fork, which can easily create subprocesses in Python programs:
  - os.fork()
  - os.getpid()
  - os.getppid()

Python: multiprocessing

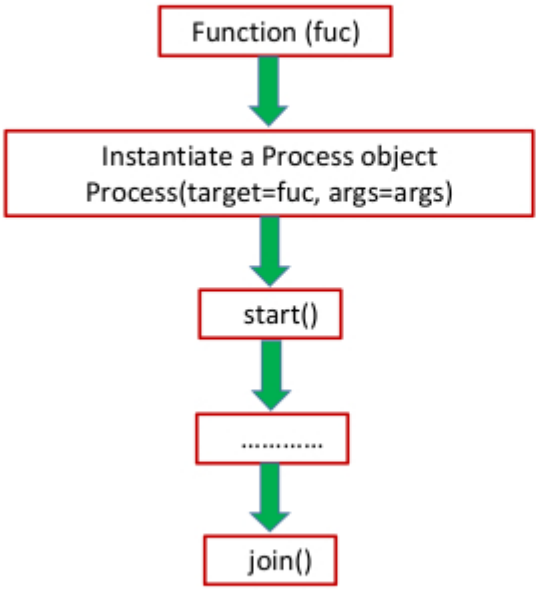
- The Python multiprocessing module is a cross-platform module for multiple processes.
- It is a “real” parallel in python uses multiple processes.
- It allows the programmer to fully leverage multiple processors on a given machine.

Python multiprocessing: Process Class

- In multiprocessing, processes are spawned by creating a Process object and then calling its start() method.
- multiprocessing.Process

  from multiprocessing import Process
- Start the Process instance with start() method
- The join() method can wait for the child process to finish, which is used for synchronization between processes.

Parallelize using Process



Python multiprocessing: Pool Class

- The Pool object creates child processes in batches.
- multiprocessing.Pool

  from multiprocessing import Pool
- The Pool object offers a convenient means of parallelizing the execution of a function across multiple input values, distributing the input data across processes (data parallelism).
- The methods of a “Pool” should only ever be used by the process which created it.

Synchronous and Asynchronous execution in Pool

- Synchronous:
  - A synchronous execution is one the processes are completed in the same order in which it was started.
  - Implementation:
    - Pool.map() and Pool.starmap(); Pool.apply()
- Asynchronous:
  - Asynchronous doesn't involve locking. As a result, the order of results can get mixed up but usually gets done quicker.
  - Implementation:
    - Pool.map_async() and Pool.starmap_async(); Pool.apply_async())

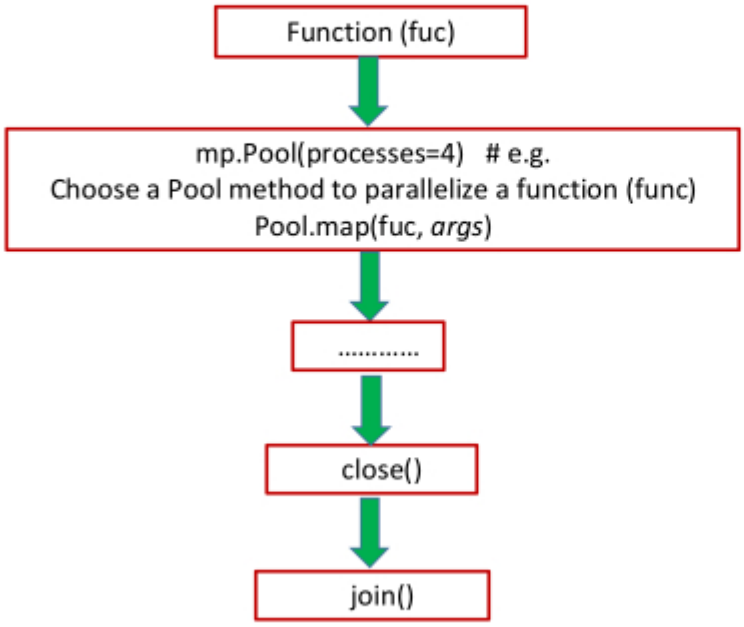
Pool Methods

- Both apply and map take the function to be parallelized as the main argument.
  - The apply() takes an *args* argument that accepts the parameters passed to the '*function-to-be-parallelized*' as an argument, whereas, map can take only one iterable as an argument.

- In starmap(), each element in that iterable is also an iterable.

- The asynchronous equivalents apply_async(), map_async() and starmap_async() lets you do execute the processes in parallel asynchronously, that is the next process can start as soon as previous one gets over without regard for the starting order. As a result, there is no guarantee that the result will be in the same order as the input.

Parallelize using Pool



Synchronous Pool methods:

- apply
- map
- starmap

Asynchronous Pool methods:

- apply_async
- map_async
- starmap_async

Multiprocessing: Process Class vs. Pool Class

Pool:

- When you have large amounts of tasks (data) , you can use Pool class.
- Only the processes under execution are kept in the memory.
- I/O operation: It waits till the I/O operation is completed & does not schedule another process. This might increase the execution time.
- Uses FIFO scheduler.

Process:

- When you have a small data or functions and less repetitive tasks to do.
- It puts all the process in the memory. Hence in the larger task, it might cause to loss of memory.
- I/O operation: The process class suspends the process executing I/O operations and schedule another process parallel.
- Uses FIFO scheduler.

Tips

- If you know how to structure and represent your data, parallelization is convenient and feels completely natural. You should pick up the basics of functional programming for this reason.

- Python is a joy to work with and eminently suitable for these kinds of programming tasks.

Python Multithreading

- Python standard library provides two modules:
  - _thread: low-level module and be encapsulated.
  - threading: high-level module and usually be used.

- To start a thread is to pass a function and create a Thread instance, and then call start() to start execution.

- https://docs.python.org/3/library/threading.html

Python GIL – Global Interpreter Lock

- The mechanism used by the CPython interpreter to assure that only one thread executes Python bytecode at a time.
- The GIL is always released when doing I/O.
- The Python interpreter was designed with a GIL global lock, which caused multithreading to fail to utilize multicore.

Read Materials

- multiprocessing — Process-based parallelism
  - https://docs.python.org/3/library/multiprocessing.html

- Stay safe!
- See you next class!

Next Lecture will Continue:

Parallel Python