DS610 – Big Data Analytics

Metin Senturk

Topics We Cover

- Programming Level Parallel Computing
- Multiprocessing Python packages
  - Standard library “multiprocessing”
  - Package “numba”
  - Package “ray”

Hardware

- CPUs on your computer
- GPUs on your computer through CUDA
  - NVIDIA

Task Requirements/ Types of Tasks

- Input/ Output (I/O) Bound Task
  - Tasks that are typically associated with disk, but may also refer to networking, communications, etc.
  - Faster with Threading and Multiprocessing
- CPU Bound Task
  - Tasks that requires more time to spend in CPU of the computer.
  - Slower with Threading

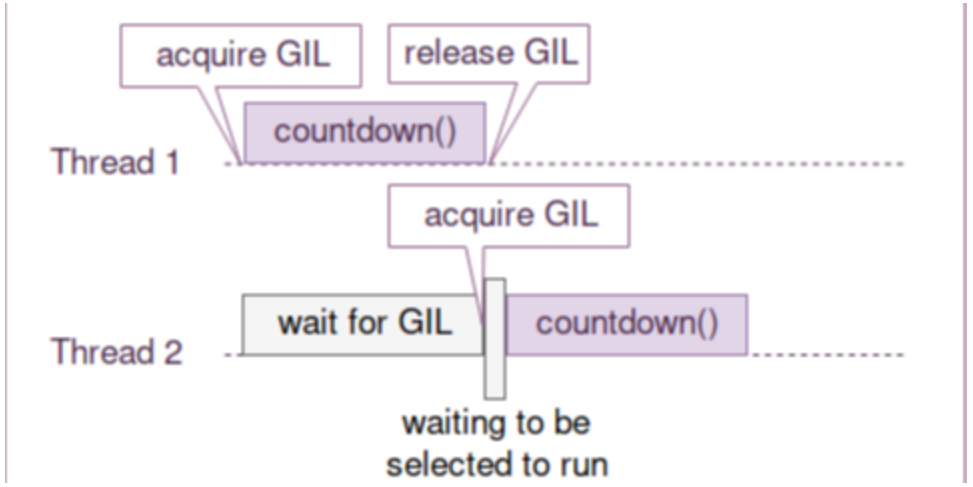
About Python

- Python is by default **single-threaded**/ processes **one** statement at a time!
- Python only executes one thread at a time, due to a type of process lock called Global Interpreter Lock (GIL).
  - Single-threaded and multi-threaded processes are same with GIL.
- If you have four CPUs, and you're trying to do some heavy data crunching using Python code, Python threads won't help. Three of the four CPUs will spend most of their time blocked, waiting to acquire the GIL. In this situation, pools of Python processes are a better approach.

- Wikipedia for definition of GIL
- The commit integrating GIL in python

About Python

- You can avoid GIL issue by
  - Using non C implemented versions of Python
    - Jython and IronPython does this
    - C implementation of python, such as Cython (not CPython)
    - Or Multiprocessing libraries (an interpreter is created for every child process)

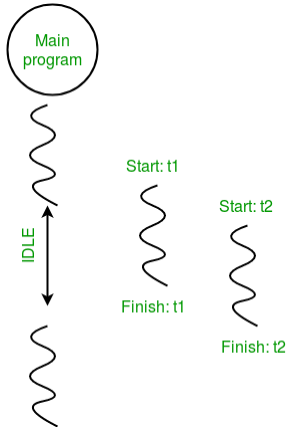


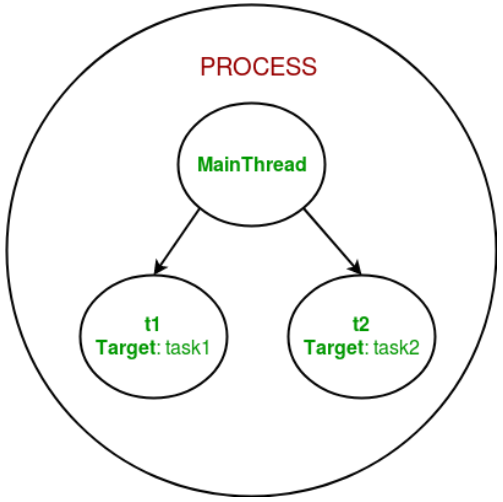
Concept of Thread and Process

- A **process** is an instance of a computer program that is being executed. Any process has 3 basic components:
  - An executable program.
  - The associated data needed by the program (variables, work space, buffers, etc.)
  - The execution context of the program (State of process)
- A **thread** is an entity within a process that can be scheduled for execution. Also, it is the smallest unit of processing that can be performed in an OS (can be assumed a subset of a process)

Multithreading

- In a simple, single-core CPU, it is achieved using frequent switching between threads. This is termed as context switching. In context switching, the state of a thread is saved and state of another thread is loaded whenever any interrupt (due to I/O or manually set) takes place. Context switching takes place so frequently that all the threads appear to be running parallelly





Multiprocessing

- The multiprocessing library uses separate memory space, multiple CPU cores
- In multiprocessing, any newly created process will do following:
  - run independently
  - have their own memory space.
- Drawbacks;
  - I/O overhead from data being shuffled around between processes
  - entire memory is copied into each subprocess, which can be a lot of overhead for more significant programs

Standard Library – Multiprocessing Usage

- There are many libraries in standard python for this purpose.
  - https://docs.python.org/3.8/library/concurrency.html
- In multiprocessing library, there are two ways to do multiprocessing.
  - Process Class
  - Pool Class
- Data Sharing between processes becomes an issue to handle.
  - Queue (FIFO based logic), Pipe, Locks
  - Managers
- Do not confuse subprocess package with multiprocessing.
  - Subprocess is a way to execute other apps, multiprocessing is an API to utilize multiple cores in your machine to do heavy tasks!

Multiprocessing – Where to go?

- https://www.geeksforgeeks.org/difference-between-multitasking-multithreading-and-multiprocessing/
- More on threading, https://realpython.com/intro-to-python-threading/
- https://www.journaldev.com/15631/python-multiprocessing-example
- https://pymotw.com/2/multiprocessing/basics.html
- Awesome article about threading vs processing, https://medium.com/towards-artificial-intelligence/the-why-when-and-how-of-using-python-multi-threading-and-multi-processing-afd1b8a8ecca

Numba

- https://github.com/harrism/numba_examples/blob/master/mandelbrot_numba.ipynb
- http://numba.pydata.org/numba-doc/latest/user/5minguide.html
- https://github.com/ContinuumIO/gtc2017-numba/blob/master/1%20-%20Numba%20Basics.ipynb

Ray

- Ray is a python package for building and running distributed applications.
- Ray takes the existing concepts of functions and classes and translates them to the distributed setting as tasks and actors.
  - This API choice allows serial applications to be parallelized without major modifications.
- Currently, only supports linux based systems, not Windows.

Ray – Where to go?

- Multiprocessing vs Ray, https://towardsdatascience.com/10x-faster-parallel-python-without-python-multiprocessing-e5017c93cce1
- Awesome ray tutorial repository, https://github.com/ray-project

Numba

- Numba provides the ability to speed up applications with high performance functions written directly in Python, rather than using language extensions such as Cython.
- Numba works at the function level. From a function, Numba can generate native code for that function as well as the wrapper code needed to call it directly from Python.
- Works best on code that uses NumPy arrays and functions, and loops.

Other packages

- Redis Queue (https://github.com/rq/rq)
- Modin (https://github.com/modin-project/modin)
- Dask (https://docs.dask.org/en/latest/)