High Performance Tricks with Python Cheat Sheet

by gonz95alo via cheatography.com/152363/cs/34482/

A word of advice

Before jumping into parallelization and code optimization, try to understand what you are trying to achieve, in comparison with what you've written. Is your code not clean enough? Perhaps untangling it and using of other libraries will help. Does your code make many read/write operations? Perhaps parallelizing in threads will speed it up. Does your code perform the same task over and over again, with minimal changes? Perhaps using parallelization in processes will aid you.
Perhaps, perhaps, perhaps... think about what you have just coded.

Concurrent processes

Main benefit	Parallelize CPU-bound tasks, independent from each other.
Import class	`from concur ren t.f utures import Proces sPo olE xecutor`
Run parallel tasks	`with Proces sPo olE xec utor() as executor:` `  results = execut or.m ap - (func, list)`
Submit a specific task to a core	`with Proces sPo olE xec utor() as executor:` `  results = execut or.s ub - mit (func)`
Option (number of cores)	`Proces sPo olE xec uto r(m ax_ - wor ker s=10)`

Extra function arguments can be Included before mapping the function in the executor. In order to do so, we need to create a partial version of the function with `partia l_func = functo - ols.pa rti al( func, a=a, b=b, ...)`.

Concurrent threads

Main benefit	Parallelize I/O-bound tasks, independent from each other.
Import class	`from concur ren t.f utures import Thread Poo lEx ecutor`
Run parallel tasks	`with Thread Poo lEx ecu tor() as executor:` `  results = execut or.m ap - (func, list)`
Submit a specific task to a core	`with Thread Poo lEx ecu tor() as executor:` `  results = execut or.s ub - mit (func)`

Concurrent threads (cont)

Option (number of cores)	`Thread Poo lEx ecu tor (ma x_w - ork ers=10)`

Extra function arguments can be Included before mapping the function in the executor. In order to do so, we need to create a partial version of the function with `partia l_func = functo - ols.pa rti al( func, a=a, b=b, ...)`.

Numba library

Main benefit	Useful decorators which compile the function the first time it is used, speeding it up in subsequent runs.
Importing decorators	`from numba import jit, njit`

Using jit	`@jit`
	`def functi on(...):`
	`  ...`

Using njit	`@njit`
	`def functi on(...):`
	`  ...`

This module is very limited, in the sense that only basic and/or numpy operations and classes can speed up. Neither scipy nor pandas or networkx can be improved. If `jit` is used, then those functions are treated as usual. If `njit` is used, then the code will fail to compile as it won't know what to do with them.

Caching