A simple Monte Carlo parallelization in numpy

March 18, 2022

1 Multiple processors

After importing *psutil*, in Python3 use:

```
psutil.cpu_count(logical = True)  # counts total virtual CPUs
psutil.cpu_count(logical = False) # only physical CPUs
```

In general it is believed that using all the virtual cpus gives a speedup of +30% w.r.t. using only all the physical one. Take it with a grain of salt and always remember to keep an eye on RAM memory usage.

2 Parallelizing in python with multiprocessing

In few words, it's the *map* builtint spreaded across multiple subprocesses:

```
# Displays [1, 2, 3]
from multiprocessing import Pool          # Standard parallel library

def f(x_int):
    return x_int + 1

pool = Pool(processes = 3)                # run 3 processes in paral.
argument_list = [0, 1, 2]                 # prepare to run f(0), f(1), and f(2)
results = pool.map(f, argument_list)      # run f(0),.., store the results here
print(results)
```

3 Parallelization in Monte Carlo

MC estimations consist of averaging among multiple simulated random variables assumed to be independent and equally distributed (so to apply the central limit theorem and compute confidence intervals).

If parallelization in itself is a complex topic, in the case of MC it simplify due to the fact of having completely independent processes (no need to send messages between them, i.e. no MPI, etc...).

Just remember to initialize the seed on every subprocess differently, to avoid having the *same* sequence of pseudorandom numbers on every subprocess. The attached code contains a complete implementation of pi estimation.

For an alternative approach, you can try with Numba to achieve CPU and GPU parallelization - we'll deal with it on another note.