Doc on multiprocessing

Mp.pool

Yue Danny Wang

Multiprocessing is useful when the main program can be divided into parallel processes. This doc shows the code for ‘embarrassingly parallel’ multiprocessing. Note that it cannot be run in interactive IDE like Jupyter. It should be run in IDEs like Pycharm or Sublime.

import multiprocessing as mp

import time

from functools import partial

Data to be used for 8 processes, and 8 lists are to be passed to them.

data = (

['a', '2'], ['b', '4'], ['c', '6'], ['d', '8'],

['e', '1'], ['f', '3'], ['g', '5'], ['h', '7']

)

Another data, also to be passed to each process.

Data2 = [1,2,3,4,5,6,7,8]

Define the worker function, which represents the work for each process. Inside each process, we print something to show which process it is and what data it is working with. Each process will sleep for some time as the ‘slow running’ function. And then use return to return the results, which is the sum of the sleeping time and the constant input1. The results will be collected by the main function.

def mp\_worker(const\_input1, const\_input2, data, data2):

inputs, the\_time = data

print('Process number ' + str(data2))

print(const\_input2 + ' constant is ' + str(const\_input1))

print (" Processs %s Waiting %s seconds" % (inputs, the\_time))

time.sleep(int(the\_time))

print (" Process %s DONE" % inputs)

return {inputs: int(the\_time) + const\_input1}

A handler function is written, to wrap the multiprocessing preparation. Partial is used to put the function and constant inputs together. Then starmap is used to map the partial function with each combination of variable inputs. In our example, we have zipped 8 sets for both data and data2, so there will be 8 performances, i.e., 8 processes. The returned results are recorded in performance, as a list of dictionaries. And then it is converted to a whole dictionary, for convenience.

def mp\_handler():

pool = mp.Pool(mp.cpu\_count())

const\_input1 = 4 # Constant input 1

const\_input2 = 'Haha' # Constant input 2

func = partial(mp\_worker, const\_input1, const\_input2) # Partial to include the function with constant inputs

performance = pool.starmap(func, zip(data, data2)) # Use map to wrap the function with input sets

pool.close()

pool.join()

performance = dict((key,d[key]) for d in performance for key in d)

return performance

Use the main function to run multiprocessing, with Final\_res being the dictionary for all results (with label being the key).

if \_\_name\_\_ == '\_\_main\_\_':

tic = time.time()

print('The number of cores of this computer is: '+ str(mp.cpu\_count()))

Final\_res = mp\_handler()

print(Final\_res)

print('All done in: '+ str(time.time()-tic))

After running for 8.4 seconds, Final\_res is printed:

{'a': 6, 'b': 8, 'c': 10, 'd': 12, 'e': 5, 'f': 7, 'g': 9, 'h': 11}