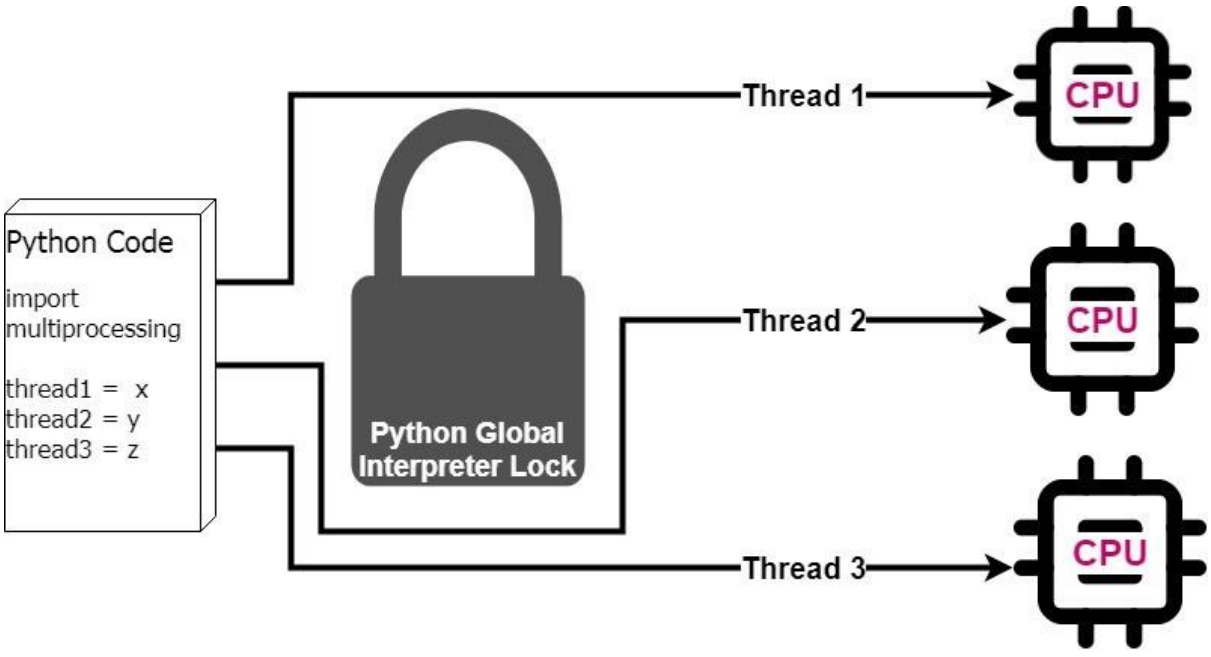
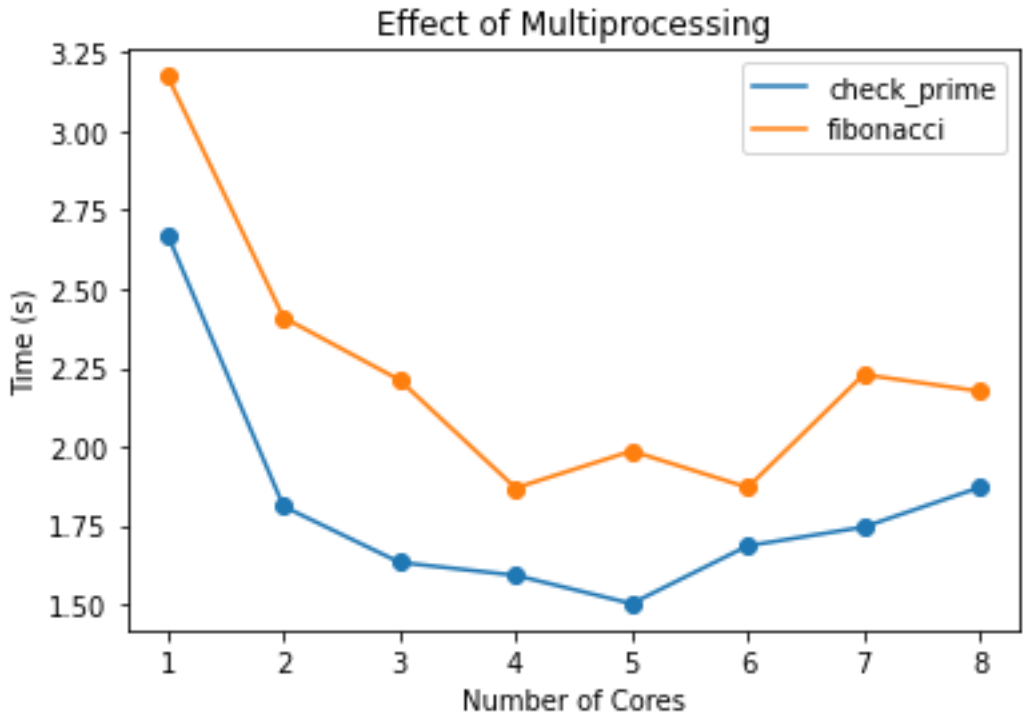
COMP30660
Assignment 1:
Multiprocessing

Peter Murphy, Conor McNicholl

Multiprocessing is the use of two or more central processing units (CPUs) within a single computer system. By using the multiprocessing module built into python we can bypass the Global Interpreter Lock (GIL) and reap the rewards of multiprocessing. Multiprocessing offers the advantages of assigning certain sections of code to more processors allowing for simultaneous execution which is meant to decrease the running time of the process. These results will attempt to document this process.



To test the run time of process of different numbers of CPU cores in python the check prime function which checks if a number is a prime number or not was connected to the pool processing function. Two lists of numbers were generated for the function to be tested on called “check_work” and “check_work2”. For part 2(b) of the assignment the Fibonacci function was used as another function to test the effect of multiprocessing on runtime. The Fibonacci function calculates the first n Fibonacci numbers in the sequence.



As we can see from the above graph both the check_prime and Fibonacci functions had drastic decreases in runtime from the initial adding of a core from 1 to 2. Check_prime dropped from 2.67 seconds to 1.81 seconds while Fibonacci dropped from 3.17 seconds to 2.41 seconds. The drop in runtime continued but not with the same intensity as extra CPU cores were added up until 4 cores for check_prime and 5 cores for Fibonacci. So far, these results show that the increase in number of cores reduces the runtime of functions.

There are a multitude of reasons as to why multiple cores in a computer reduce runtime relative to single CPU core computers. CPU cores deal with processes one at a time in the form of threads i.e., they solve one problem at a time. When multiple cores are added more problems can be processed at the same time or a large complex problem can be broken into several parts and each part solved by the separate cores. Looking at the results we can see that adding cores did decrease runtime up until a point then adding more cores did not seem to help much. Amdahl's law should be considered when multiple cores are added. If there is a section of code that cannot be parallelised i.e., another CPU core cannot be added to do a section of the code then the runtime of the process is not decreased by the addition of multiple CPU cores because they all must wait for one core to solve the non-parallelised section of code. This could be the reason as to why the addition of cores decreases runtime up until a certain point (around core 4 or 5).