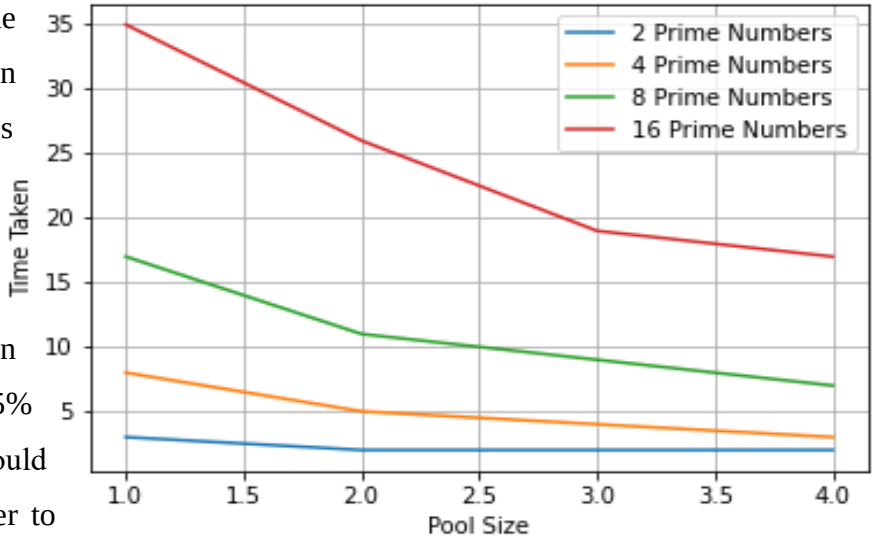
Assignment 2 Multiprocessing- Gus Boothman - 17350796

The aim of the assignment is to examine speedup derived using multiple CPU cores through the multiprocessing facility in Python. Multiprocessing is the use of at least two (often more) Central Processing Units within a single computing system. The Python Module multiprocessing was used. This allows us to bypass the Global Interpreter Lock (GIL) by utilising subprocesses instead of threads. The GIL mechanism is a type of mutex lock which protects shared resources thus making it difficult to implement multiprocessing.

My computer's specification is an Intel(R) Core(TM) i3-7100U CPU @ 2.40GHz, 2400 Mhz, 2 Core(s), 4 logical processors. My computer has 4 logical cores and only 2 physical cores. The reason it has more logical cores than physical cores is because it implements hyperthreading. This means for each physical core the operating system addresses two virtual cores and shares the workload between them. For the notebook, I went as far as 4 cores even though the difference, in theory, should only be seen after changing from 1 to 2 cores as the python module cannot execute threading correctly to see the difference with hyperthreading. It can only utilise physical cores.

It is important to consider Amdahl's law which is the formula that shows the theoretical speedup of the execution of a task. As the checkprime function is a parallelized function meaning the checking of one number does not depend on operation of anything else. Thus, increasing the cores should speedup the task linear fashion.

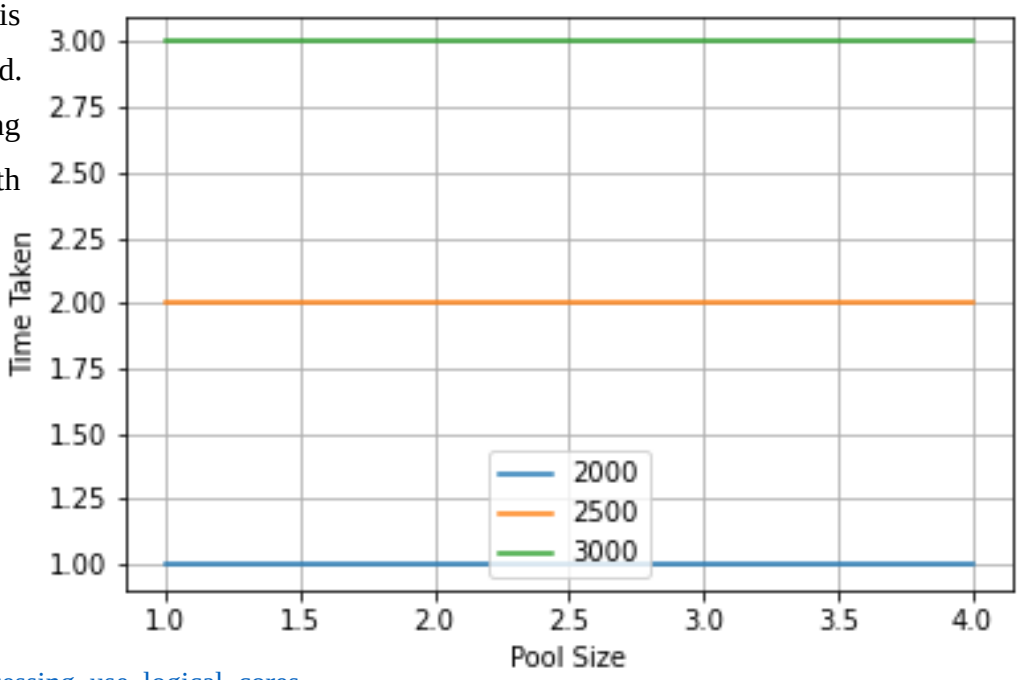
There four levels of testing; with 2 prime numbers, 4 primes, 8 prime numbers, and 16 prime numbers(all 8 digit). As mentioned each level of testing was ran with different pool sizes (1,2,3,and 4). As you can see from the graph there is a big difference between using 1 core and 2 cores. This makes sense as we allow for an extra core to be used and parallelisation occurs which speeds up the computation time. On all levels the decrease in run time between 1 and 2 cores is at 33-35% reduction. From Amdahl's law we would usually be expecting a decrease closer to 50%, the difference may be due to background programs running on my computer. As the cores



increase from 2 to 3 the difference between running times does reduce but not as significantly with a reduction in the region of 17-25%. The running time decreases again with an increase to 4 cores, but it begins to plateau, and the difference is far less.

For **task 2**, I decided to implement an iterative summing factorial algorithm. The function calculated the factorials leading up to and including a given number and then summed all the results. The reason I chose this function is because I wanted to try and implement a function that preformed more as serial program compared to the checkprime function. A serial program is a program that has to run one at a time because the result of the next run depends on the result of the previous run. As you can see from the graph that it makes no difference as the cores increase. That is because each operation has to run on one core as it depends on the result of other operations. The summing factorial of 2000, 3000, 3500 were found.

To conclude, this assignment has given me an understanding of how multiprocessing works. The first tasks results were not exactly what was expected but a definite speed up was apparent. The speed up was not as great as expected possibly due to background programs on my computer. Furthermore, I learned that speedup with multiprocessing also depends on what type of program you are running. As you can see with task 2 a more of a serial style program there is no significant speed up with the different cores. However, a program like checkprime can be speedup because it is more easily parallelised. Finally, it would be interesting to see the results with computer systems that has more cores, as my machine has only two physical cores, so testing is possibly limited.



Reference:https://quick-adviser.com/does-python-multiprocessing-use-logical-cores/#Does_Python_multiprocessing_use_logical_cores