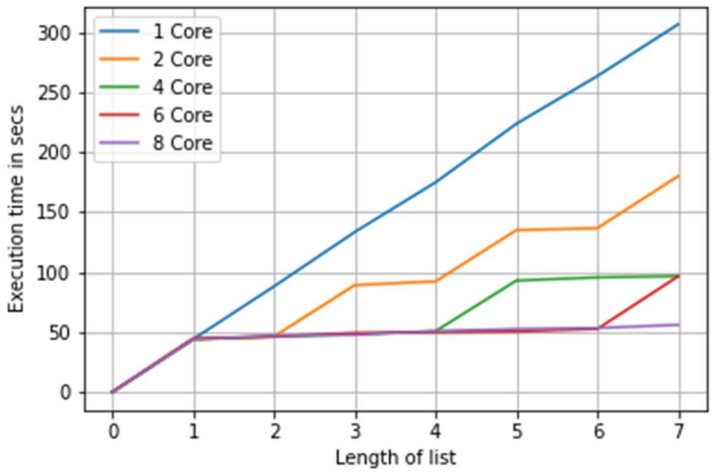
Multiprocessing in Python

1.What lessons can be learned from these results?

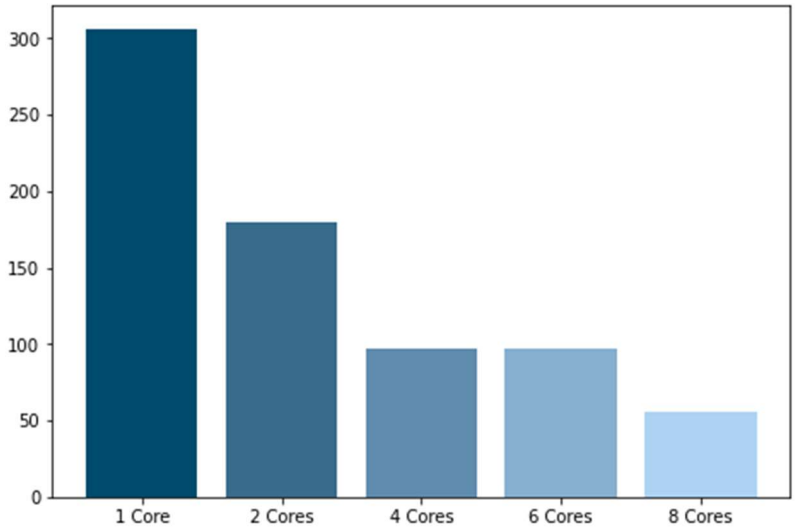
After adjusting the pool_process function to return the time taken. A for loop was created to iterate over an 8-item list, each time increasing the number of objects passed into the pool_process until all objects had been dealt with. Times were appended to a separate list along with the length of that current list, this was repeated for 1 Core, 2 Cores, 4 Cores, 6 Cores and 8 Cores. Using these lists, the following graph was plotted to visualise the speedup achieved with multiple cores.



(Figure 1 from Test 1)

As expected, we can see noticeable speed increases as more cores are involved in the process. By implementing multiprocessing, the python code was able to execute much faster and leverage all the available CPU cores on my machine.

The next test involved appending the overall runtime of each core to a list and plotting it on the following bar chart as another visual aid.



(Figure 2 from Test 2)

With more cores, the time spent to complete the overall process decreases from 306.4 secs (1 Core) to 56.1 secs (8 Cores). This is an astonishing 81.7% execution time decrease between 1 Core and 8 Cores. Another important lesson to be learned here is that multiprocessing is effective, however it reaches a plateau under hardware limits. This is apparent when we compare the overall running times between the cores. The execution time decreased by *41.28%* between *1 Core* & *2 Cores*, *46.28%* between *2 Cores* & *4 Cores*, *0.17%* between *4 Cores* & *6 Cores* and *41.89%* between *6 Cores* & *8 Cores*. Please see *"Comparison of Results"* in the notebook for the full calculations involved.

This illustrates what Amdahl's Law can tell us about multiprocessing. That is that the speedup of a parallel algorithm is limited by the fraction of the problem that must be performed sequentially. Essentially, Amdahl's Law is used to predict these maximum speedups for program processing using multiple processors.

We can conclude that multiprocessing has provided massive computational speed increases (81.7% decrease in execution time in our test). However, we have learned that python does not inherently take advantage of multiple cores as it is limited to using a single processor caused by the GIL (Global Interpreter Lock), which was implemented to handle a memory management issue. By utilizing python's built-in multiprocessing module for test 1 and test 2, we were able to designate a portion of our code to bypass the GIL and send the code to multiple processors (Up to 8 in our case) for simultaneous execution, which led to our significant reduction in execution times. We have now learned how to utilize our processors to their full potential.

2. b)

Repeat the exercise in 1 running on a VM through VirtualBox and assess the impact (performance hit) of using the VM.

I have repeated the exercise from part 1, however this time on a virtual box that is running Ubuntu Linux 20.04 and assigned it 8 cores. The run times of test 1 and test 2 on the Ubuntu server were appended to a list and plotted. Please see the notebook for the graphs of the Ubuntu process.

The performance hit from running on Ubuntu vs Windows is extremely evident. The Ubuntu server took *199.75%* longer to run with *1 Core*, *194.64%* longer with *2 Cores*, *189.43%* longer with *4 Cores*, *190.53%* longer with *6 Cores* and *176.9%* longer with *8 Cores*. On average, Ubuntu took **190.25%** longer to complete than Windows. Please see *"Comparison of Results"* in the notebook for the full calculations involved.

In normal circumstances Linux Ubuntu is a much leaner operating system. It has far less overhead, less background tasks (that are mostly dormant) and read-write operations are incredibly fast. However, in this case Linux is running on a virtual machine. Because the operating system is virtualized, you get penalty over running it on the bare hardware. That means that the OS running in virtual machine will run slower than if it would run directly on that same machine without virtualization layer.

In conclusion, the huge performance hit (190.25% slower on average) we are seeing from running the process on Linux is due to it being on a virtual machine rather than the OS itself. What is important to note here is that the variances in execution times between each of the cores is consistent between both Linux and Windows.