COMP30660 - Assignment 2: Multiprocessing

Authors: Michael Davitt - 17309836, Tarik Carvalho - 21207065

Task 1:

Introduction:

Our objective is to perform an analysis of the use of multiprocessing to complete tasks in parallel. We wish to examine the degree to which multiprocessing increases efficiency. For the purpose of this analysis, we have used runtime in seconds as a measure of efficiency.

Our initial expectation is that the total runtime of a set of processes should decrease proportionally as the number of cores increases. For example: If we have four processes that each take 2 seconds to complete, we expect that this will take 8 seconds in the case where we are using 1 core, and it will take 2 seconds in the case where we are using 4 cores, because each core can execute a 2-second task simultaneously.

In this section, we will analyse the increase in efficiency when multiprocessing is applied on a slow process. Our process in this section is as follows: Use the check_prime function to check whether the positive integer 15,488,801 is a prime number. We will run this process a varying number of times, from one iteration to twenty iterations (ie: we will start by running it once and get the total runtime, then we will run it twice and get the runtime etc.).

All plots and further analysis will be included in the notebook.

Observations:

We see that as the number of process iterations increases, the total runtime increases for each core. However, the increase is a linear function for the one-core case, whereas in the other cases, it is more of a stepwise function. This is because when we use more than one core, processes can be completed simultaneously, so increasing the number of processes up to a point will not drastically increase the total runtime.

We note that the increase in efficiency is not exactly proportional to the increase in cores. This is because the addition of more than one core creates additional overhead, which increases the runtime by a small amount. However, in this task where we are running tasks that take a long time to complete, the increase in runtime due to the increased overhead is offset by the decrease in runtime due to collaboration between multiple cores. The net result is that multiprocessing increases efficiency in this example.

Task 2:

Introduction:

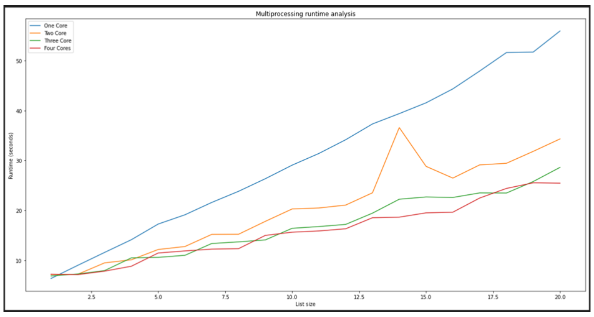
In our next task, we decided to re-run our check_prime function with the input 15,488,803. This is a non-prime number, which only requires 10 iterations of the for loop in the check prime function to verify that it is not prime. This is significantly faster than our process in task 1, which needed to iterate through all numbers between 2 and 15,448,801 in order to determine that it is prime. We would expect, therefore, that the efficiency of multiprocessing would be reduced, as the overhead involved in setting up the multiple cores would have a material impact on our total runtime.

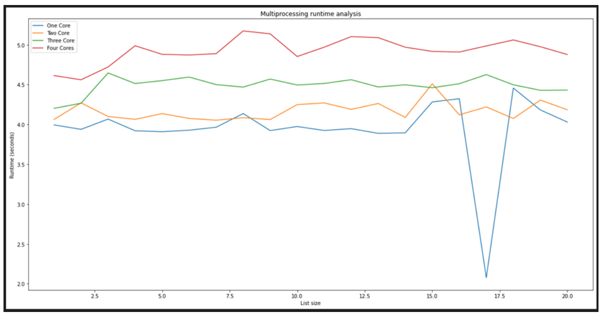
Observations:

As per our expectations, the use of multiprocessing in this example had a negative impact on efficiency. When we used more than one core, the total runtime was larger than it was when we used a single core. This is because when a process is quick to complete, the runtime cost of setting up the multiple cores becomes larger relative to the total task completion time.

We can conclude from our above analysis that multiprocessing is only efficient in certain circumstances. The processes that are fed into the different cores should individually take a long time to complete. For example, in our first task, the runtime for a given call to the check_prime function was approximately 3 seconds. This means that the implementation of multiple cores increased efficiency, as multiple function calls could be shared amongst the cores, as opposed to relying on one core to do all of the work. In our second task, the runtime was significantly smaller for an individual function call, so sharing these very short tasks amongst the cores did not have a material impact on the total runtime, because the increased overhead of setting up the multiple cores had a larger proportional impact on the total runtime.

Plots (see notebook for a larger version):


Multiprocessing runtime analysis
One Core
Two Core
Three Core
Four Cores
Runtime (seconds)
List size


Multiprocessing runtime analysis
One Core
Two Core
Three Core
Four Cores
Runtime (seconds)
List size