COMP 30660

Assignment 2

Brian McMahon

15463152

UCD School of Computer Science

University College Dublin

March 24, 2024

1 Question One

1. Connect the check_prime function to the Pool processing function. Generate sets of work (numbers to be checked) to be processed by the pool. Quantify the speedup achieved with multiple cores (at least 2). What lessons can be learned from these results?

1.1 Leveraging Multiprocessing to Check Primes

The supplied Jupyter notebook was used as a starting point. The pool function was connected to the check prime function, also present in the zip download. The function was used with a variety of pool sizes to establish the relationship between pool size and calculation length.

Using 450 prime numbers, taken from the supplied resource (all 8 digits in length), the check prime function is used to determine how long the PC takes to check there prime 'ness'. The function was rerun with varying pool sizes from 1 to 15 to determine the relationship between pool size and program speed. As seen in fig 1 the program completion time is proportional to the negative exponential of pool size. Local instabilities occur as places relating to poor work splitting of the workload at those pool sizes.

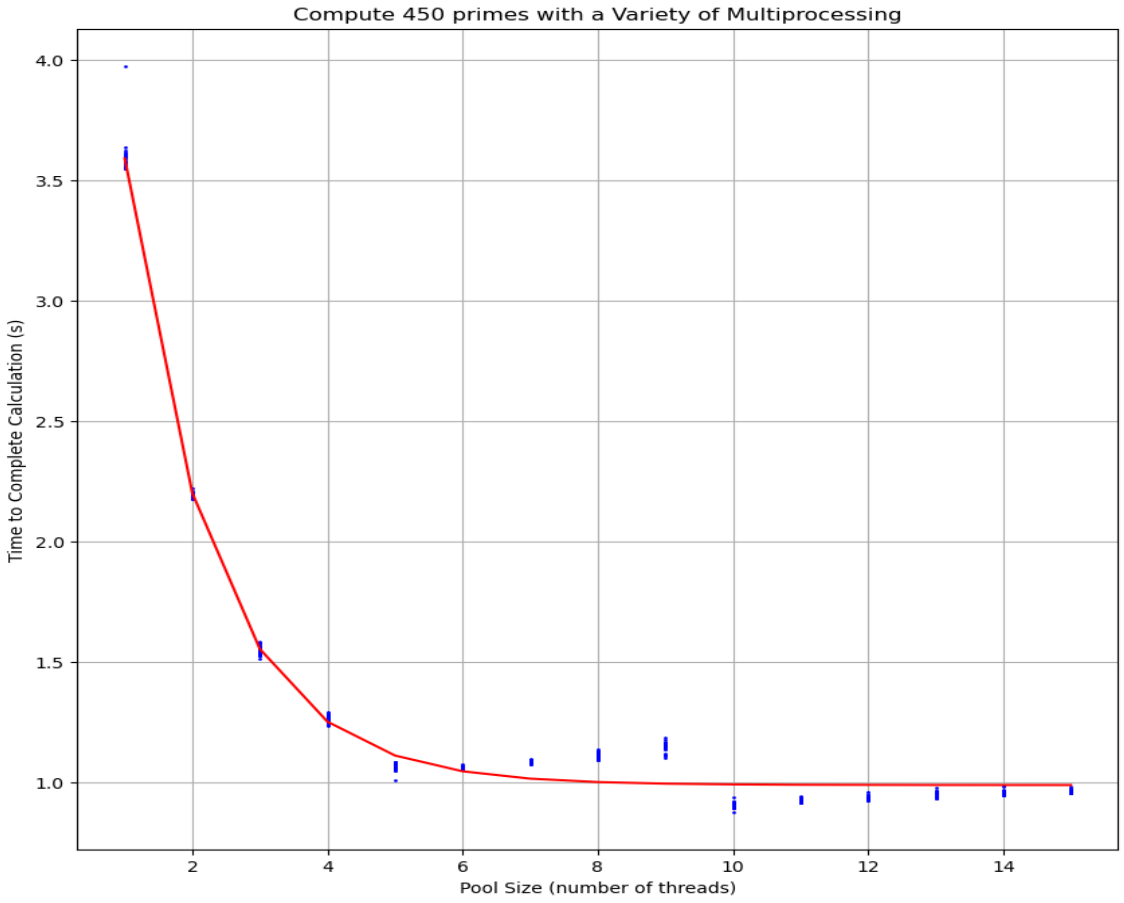

Figure 1: The time for process completion can be seen to decrease exponentially with increasing pool size. (blue) data. (red) curve fit of $ae^{-bx} + c$

1.2 Consistent Results Under Different Workloads

To avoid the possibility that the time reduction was just a feature of the specific chosen primes we recompute the time experiment for 4 different subsets of prime numbers not yet tested. This demonstrates the consistent shape of the time feature as the pool size increases, see fig 2.

1.3 Lessons Learned from Results

From these results we have learned that we can leverage more processing power using by using multiprocessing. The more pools we establish (an analog for number of worker processes) the more performance we can get out of our CPU up to a threshold. At some point the number of processes will actually limit performance as the overhead to manage them outweighs the benefit. We see that the limit of performance is being reached in fig 1 as the time asymptotically reaches the minimum.

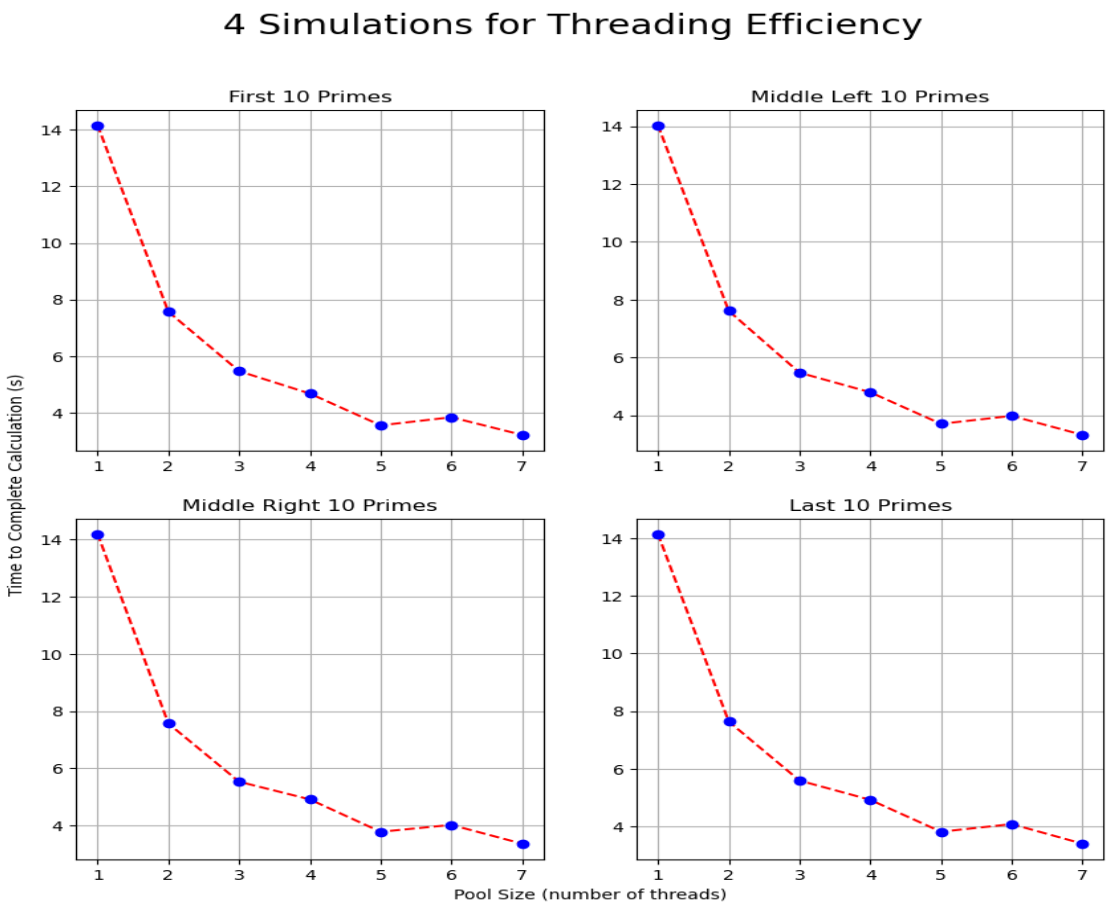

Figure 2: Four separate investigations of the effect of pool size on processing time. This shows a unified response across different work sets to demonstrate that the working set has little effect on completion time when work is of same magnitude

2 Question Two

> 2.(a) Complete only one of the following. Identify an alternative processing task that will also test the CPU and repeat the assessment. What lessons can be learned from these results?

2.1 Leveraging Multiprocessing for Chemical Kinetics

Small amount of background

Chemical kinetics is the study of the rates of chemical reactions, and the factors that influence them. Chemical re- actions are the process whereby a reactant turns into a product. More complex systems have several reactants and products. One such reaction is the Three Variable Autocatalator. In which chemical substances combine and transform to output different chemicals. Depending on the initial amounts of each substance the output will vary. The three variable Autocatalator is an example of a chaotic system; a system for which a slight variation in initial conditions yields an unpredictable outcome.Therefore computing the equations governing this reaction is computationally intense.

In the folder I have added a python file: "bifurcation.py". Inside are a set of functions which define the differential equations which govern the TVA. We compute the TVA with shifting initial parameters to demonstrate an effect known as period doubling to chaos. Notably for this assignment multiprocessing is leveraged to compute the result and is the effectiveness is shown by means of a plot of time against pool size. We also see how changing some initial conditions yields a different time gain compared to others, demonstrating the importance of setting up a problem correctly when leveraging multiprocessing.

2.2 Response to Multiprocessing

It can be seen from fig 3 that a similar response is given to multiprocessing in this use case. A negatively exponential proportionality can be seen between pool size and processing time, and the time taken asymptotically arrives at the minimum. The TVA is a really good example of an appropriate time to use multiprocessing due to the problem architecture matching well with that of multiprocessing. In this problem we have a set of differential equations which need to be computed as time increases. But we also need to calculate the equations again with different initial conditions. So each set of initial conditions can be thought of as a process or thread and the time evolution is what the thread computes. Included also is the bifurcation diagram of the TVA interaction showing at some initial conditions (x-axis) the number of y solutions is becoming chaotic whilst is stable at other initial conditions fig 4.

2.3 Lessons Learned from Results

One of the key lessons that can be taken away from this experiment if the focus on framing your problem correctly to leverage multiprocessing. Observe fig 3 two experiments are run. Firstly the experiment is run using 100,000 time steps for the TVA differential equations, using a variety of pool sizes. Secondly we rerun this experiment but with 10,000 time steps. The minimum computational time occurs at a pool size of 12 for experiment 1 but occurs at a pool size of 7 for experiment 2. This shows that there's not just an optimum number of processes when using multi-programming but also a factor matching the problem type with the machine architecture.

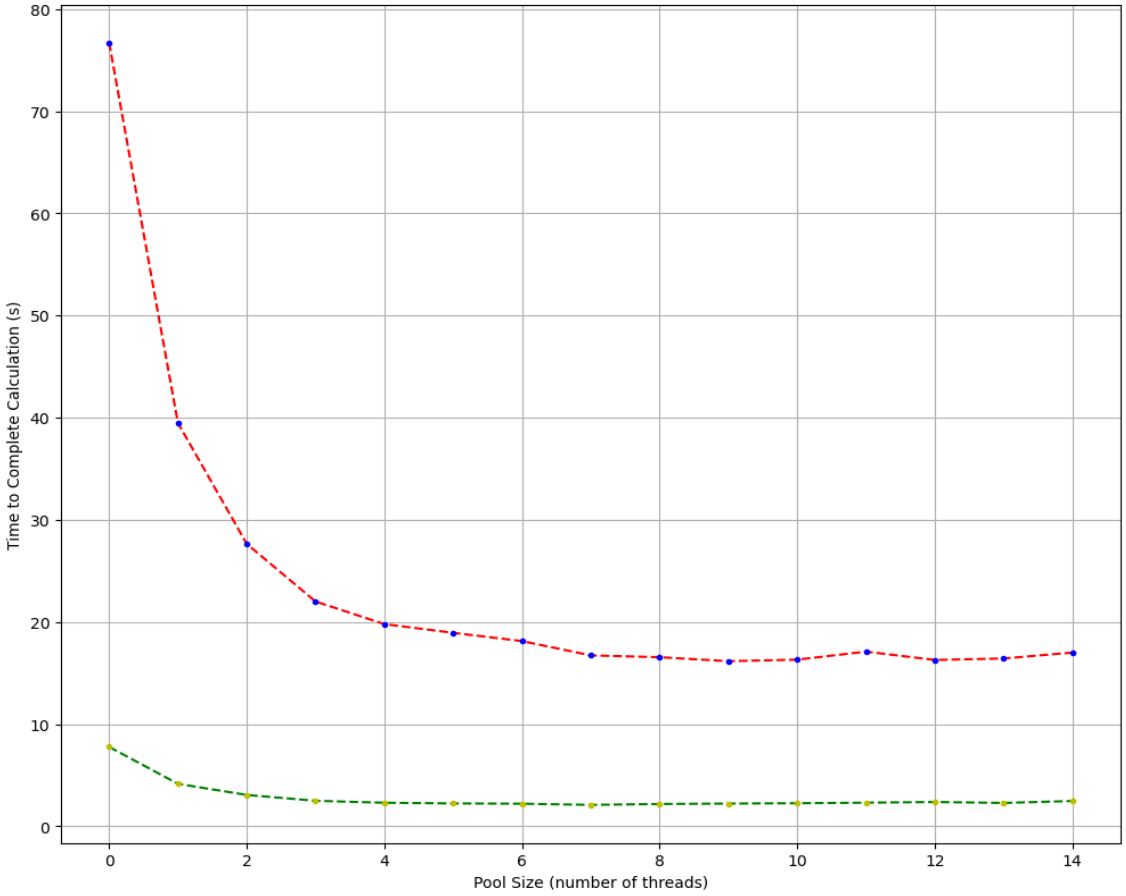

Figure 3: Red: Time of process completion when using 100,000 time steps and variable pool size. Green: Time of process completion when using 10,000 time steps and variable pool size. Minimums occur at different positions showing the effect of problem architecture matching computer architecture.

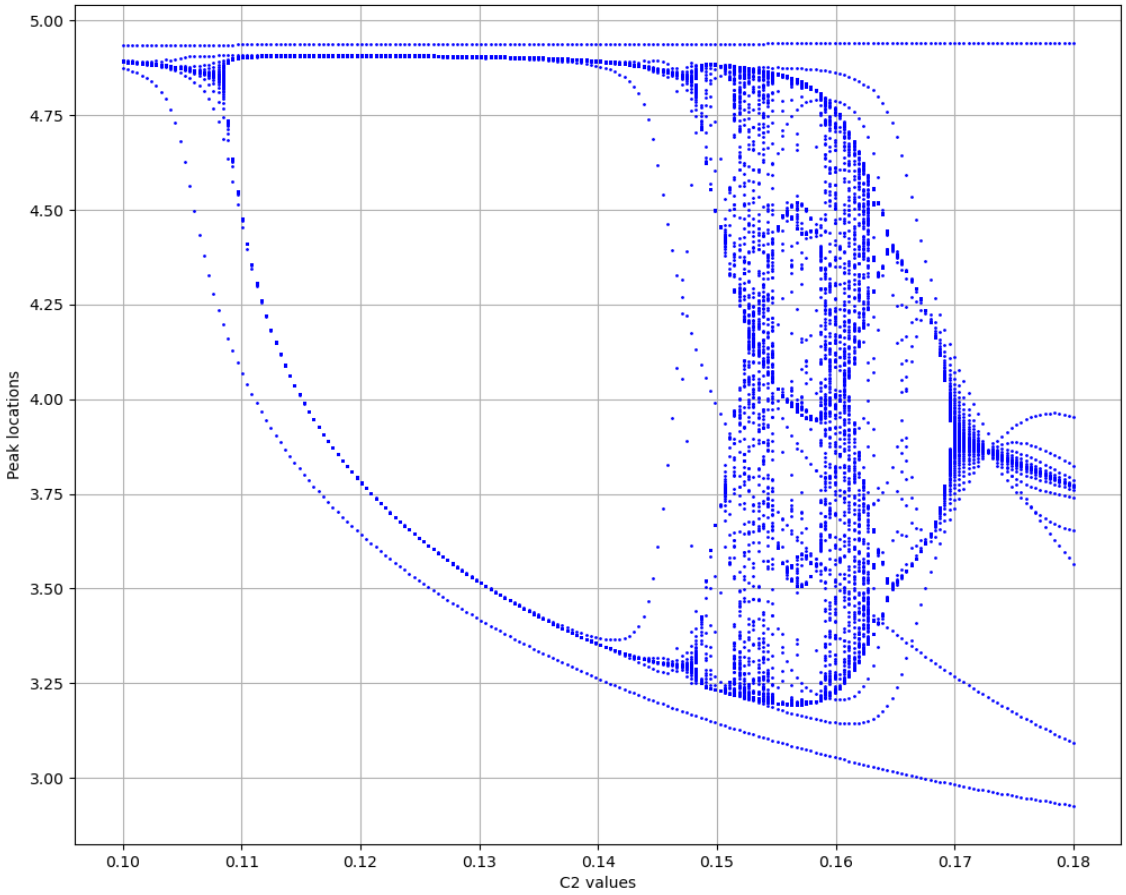

Figure 4: Bifurcation diagram demonstrates the number of possible steady state solutions as initial parameter c2 changes. Initially only 2 or 3 steady state solutions exist, but when 0.15<c<0.17 the equation becomes chaotic with many possible solutions which are not predictable based on other parameters