MULTIPROCESSING RESULTS

1.

The first algorithm tested with multiprocessing was verifying prime numbers. The pool function was initially paired with a set of 18 large prime numbers. The 'X' axis denotes the task being solved, while the 'Y' axis is the time (seconds) it takes for the given task to be solved in relation to the start of the entire function.

While there is a significant increase in performance from 1 processor to 2 processors, this has diminishing returns as more processors are added to the workload:

1. 179 seconds.
2. 119 seconds.
3. 95 seconds.
4. 94 seconds.

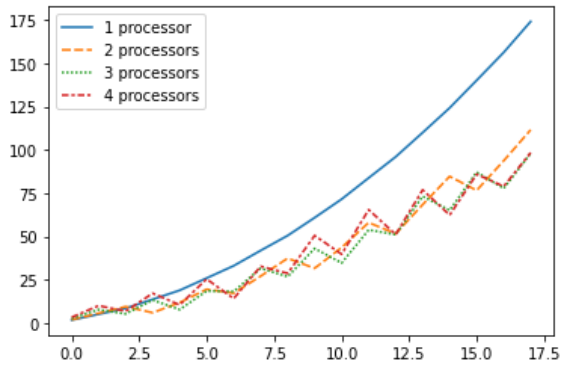

Figure 1 - Line Plot of Large Prime Numbers (x=tasks, y=seconds)

However, this difference in performance becomes more and more pronounced as the number of tasks and time to compute increase. This phenomenon can be explained by Amdahl's Law.

An interesting point to note while graphing this work is that the line plot has a non-linear trend. In fact, it looks rather exponential. While on initial inspection this appeared to be a result of multiprocessing, it is in fact because of the increasing complexity of solving for each subsequent larger prime. As a result, it would be a more valuable exercise to examine a larger list of similarly-sized prime numbers. Thus, 1000 smaller primes were generated in an effort to capture the trend in multiprocessing.

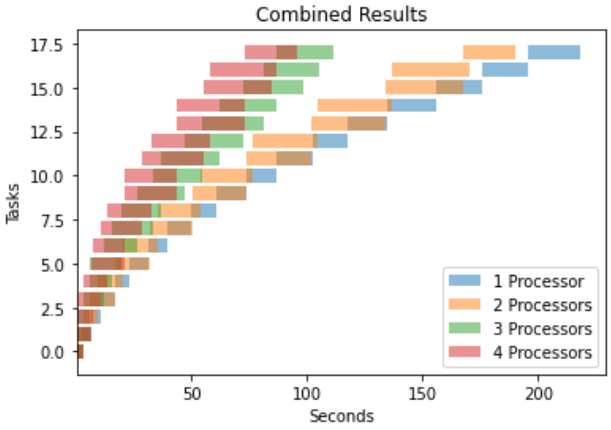

Figure 2 – Horizontal Bar Chart of Large Prime Numbers

By using many smaller primes, it was much easier to identify trends. For each line in the graph, the number of 'grouped spikes' is directly proportional to the number of processors it is using. This demonstrates the pooling function clearly, as tasks are grouped and handed off to each processor accordingly. This is why the plot for '1 processor' is a single straight line – it is solving each prime one after another, all tasks are essentially in one large pool. It is also important to examine the highest point of each line, as that is the finishing point of the entire workload. Once again, there is a significant increase in productivity between 1 and 2 processors, but this becomes more marginal as more processors are added.

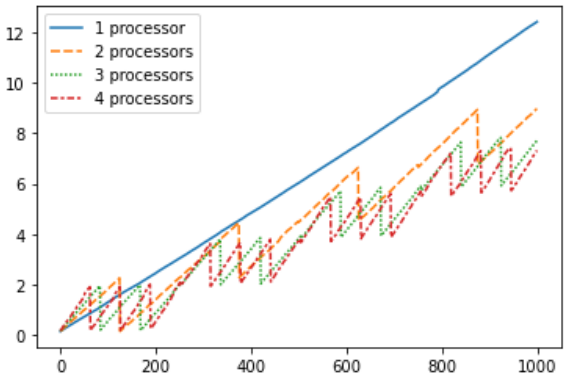

Figure 3 - Line Plot of Small Prime Numbers (x=tasks, y=seconds)

One point of interest is that the average running time of any given task appears to be shortest when using a single processor. This has been illustrated rather clearly by the horizontal bar chart.

This is due to the overhead associated with multiprocessing, where tasks are divided up between the various processors. However, this overhead is mitigated by the nature of parallel processing.

PART 1 (Small Prime Numbers)			
	TOTAL TIME	**INDIVIDUAL TIME**	**INDIVIDUAL STD**
1 PROCESSOR	12 seconds	0.01134 seconds	0.0020
2 PROCESSORS	9 seconds	0.01414 seconds	0.0030
3 PROCESSORS	8 seconds	0.01757 seconds	0.0035
4 PROCESSORS	7 seconds	0.02300 seconds	0.0024

2. (A)

For the second question, each task involved the creation of a list of variable length filled with integers between 10,000 and 20,000. Each element in the list was subsequently summed. This task was ran 20 times in order to test how another series of CPU-intensive operations are affected by multiprocessing. The results trended quite similarly to the prime number function, although the running times were generally much shorter:

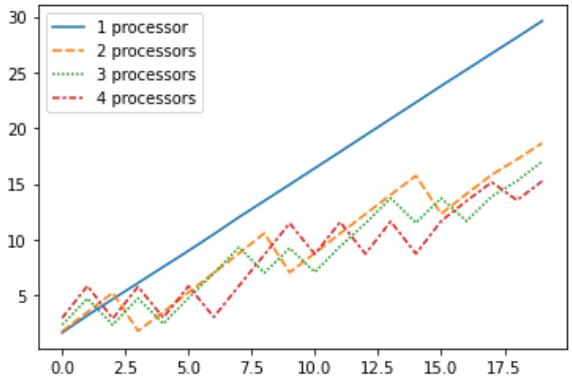

Figure 4 - Line Plot of Sum of List (x=tasks, y=seconds)

The main difference here, is that the running time with 2 processors is much closer to that of 3 and 4 processors. The nature of this task also allows for much more uniform running times, too, which means the linear nature of these processes can be seen more clearly. Like with the prior exercise, it is interesting to note that using more processors does not always result in the fastest approach. For instance, there are points where 3 processors are ahead of 4 processors, particularly when the number of tasks is not a multiple of 4. However, this does not ring true across the board. As the number of tasks increase, it becomes clearer that more processors results in faster task completion.

The earlier point regarding average task running time appears to remain true for this section, too. In fact, it seems that there is a correlation between increased task running time and number of processors in operation. Upon examination of the horizontal bar chart, 1 processor seems to have a relatively short and consistent running time. With 2 processors, this running time for each task increases, along with its inconsistency. This trend continues with 3 and 4 processors. This is likely due to the overhead that comes with multiprocessing.

It would be interesting to engage in further research regarding the use of IO, API calls, and modules such as Numpy in multiprocessing, like in this article.

For more detailed results, please explore the Jupyter Notebook in this folder.

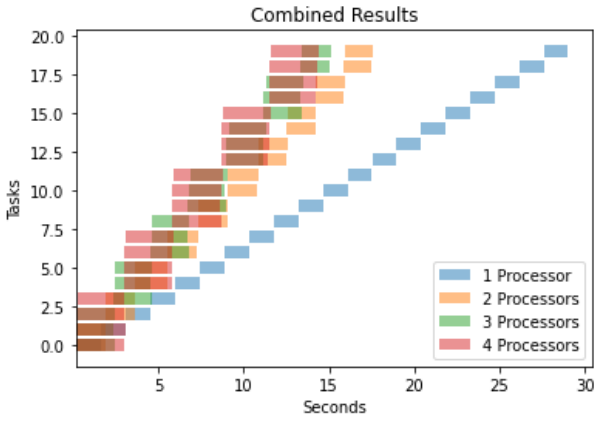

Figure 5 - Horizontal Bar Chart of Sum of List

PART 2			
	TOTAL TIME	**INDIVIDUAL TIME**	**INDIVIDUAL STD**
1 PROCESSOR	31 seconds	1.565 seconds	0.0028
2 PROCESSORS	18 seconds	1.709 seconds	0.0033
3 PROCESSORS	16 seconds	2.449 seconds	0.0057
4 PROCESSORS	14 seconds	2.886 seconds	0.0069