Lab 3 Report

Colin Brum (cmbrum) - 20518388
Yangmengyuan Zhao (y482zhao) - 20704842

Timing Measurement Data

N	B	P	C	AVG Time Processes	AVG Time Threads	STD Time Processes	STD Time Threads
100	4	1	1	1.914494	0.352712	0.198659482	0.070754937
100	4	1	2	1.952178	0.358086	0.176059156	0.058330323
100	4	1	3	1.862372	0.43271	0.142391761	0.068543752
100	4	2	1	1.953714	0.362282	0.177451493	0.040750589
100	4	3	1	1.95958	0.416392	0.120698548	0.092911304
100	4	2	2	1.969334	0.408622	0.159482884	0.067477219
100	4	3	3	2.288894	0.459708	0.108721786	0.09092891
100	8	1	1	1.79506	0.341504	0.196792745	0.029157263
100	8	1	2	1.869824	0.351404	0.15452903	0.056431275
100	8	1	3	1.765788	0.426842	0.099514899	0.062723497
100	8	2	1	1.775992	0.365546	0.095561791	0.047455662
100	8	3	1	1.879592	0.4071	0.061669324	0.069936757
100	8	2	2	1.761494	0.389384	0.083373965	0.067693726
100	8	3	3	2.256466	0.474842	0.104782617	0.118774918
398	8	1	1	2.525492	0.567462	0.18818028	0.036729995
398	8	1	2	2.942564	0.587086	0.167202314	0.060239046
398	8	1	3	2.893624	0.810682	0.150798696	0.078350245
398	8	2	1	2.86755	0.580638	0.102380875	0.06401096
398	8	3	1	3.128358	0.69286	0.066055415	0.100383606
398	8	2	2	2.440838	0.666036	0.106344326	0.077047691
398	8	3	3	2.966372	0.74983	0.153445761	0.097669653

An average time was used for both thread and process execution times to obtain a more accurate representation of the execution time, which could be influenced by variables extraneous to the implementations such as the amount of load on the CPU at the time of execution, the number of processes running, etc. The standard deviation demonstrates a bound for how much runtimes will deviate from the average overall execution time.

Comparison of Process and Thread Implementations

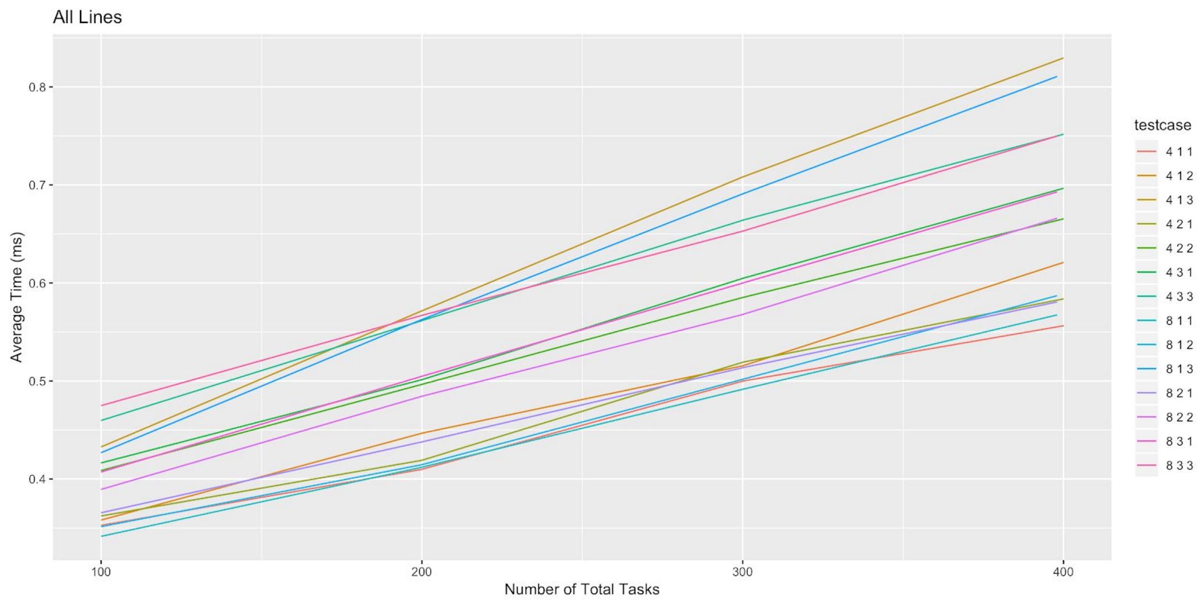

Figure 1: Average thread implementation execution time with respect to changing N/B/C/P

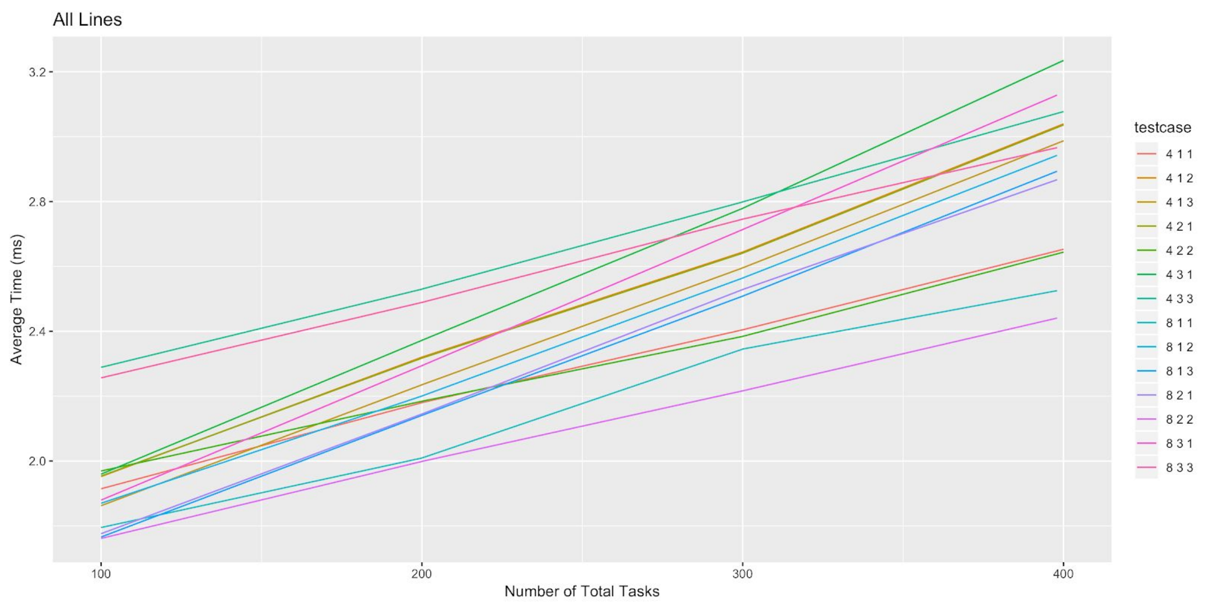

Figure 2: Average process implementation execution time with respect to changing N/B/C/P

At a high level, the thread implementation is much more efficient than the process implementation. This can be seen by comparing the average execution times of Figure 1 (threads), to Figure 2 (processes). The minimum execution time calculated for the thread implementation is approximately 0.3 ms, whereas the minimum execution time for processes is around 1.8 ms. This makes sense, because the process implementation has much more overhead.

In the process implementation, syscalls must be performed to fork the producers and consumers, which involves switching between kernel and user mode, which takes time. Also, each process must maintain its own execution state, and be scheduled by the operating system in parallel. The message passing using *mq_send()* and *mq_receive()* also requires yet another syscall to do the IPC, which again has a certain amount of overhead associated to it. In general, the amount of time required to manage a process is greater than the amount of time required to manage threads, due to the difference in complexity.

For the thread implementation, no IPC is done to communicate between them, saving system calls. However, the thread calls still involve syscalls, but it is assumed that the thread syscalls are faster than the process related syscalls. Additionally, there is much less state to initialise and manage for a thread, and global variables can be shared amongst the threads.

Advantages/Disadvantages of Process and Thread Implementations

As discussed earlier, one of the advantages of using the thread implementation is that the average execution times are faster given an identical set of N/B/C/P. However, if one thread within the process segmentation faults, all the producers and consumers will die. This is not the case for the process implementation, where a producer or consumer could be terminated, or experience a fault, and then be replaced by another one with no issues. So, the process implementation is a bit more robust than the thread. Additionally, from the perspective of implementing the multiproducer/multiconsumer code, the process implementation was much easier. The APIs to do this in a process are much higher level and more abstract, with details such as making sure the queue is not full, blocking when appropriate, etc all being abstracted from the programmer. However, this additional level of abstraction comes at the cost of a slower execution time.

Effects of Changing N/B/C/P on Execution Time

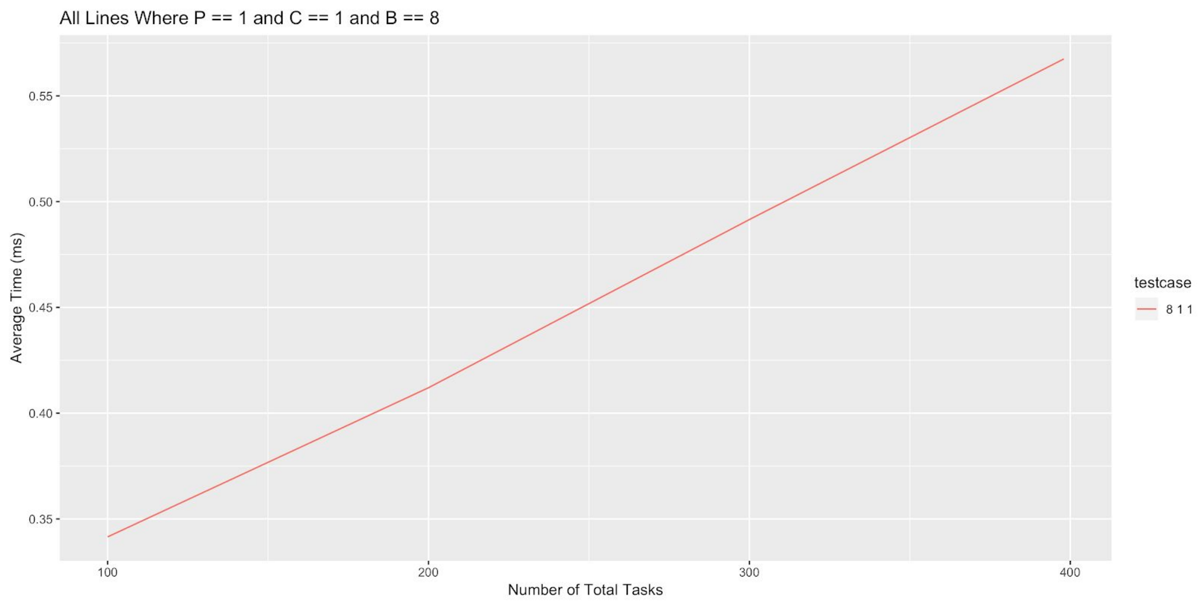

Figure 3: The effect of changing N on thread implementation execution time

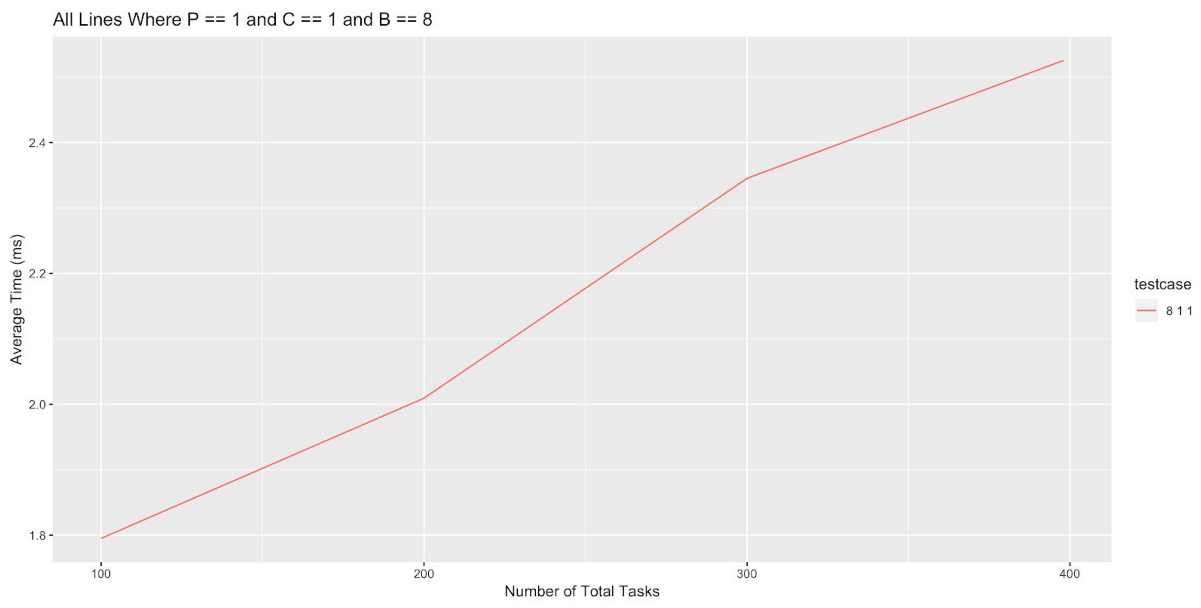

Figure 4: The effect of changing N on process implementation execution time

As can be seen in Figure 3 and Figure 4, as the amount of total tasks increases, the execution time increases fairly linearly. This makes sense intuitively because there is more work to get done, and therefore takes a longer time.

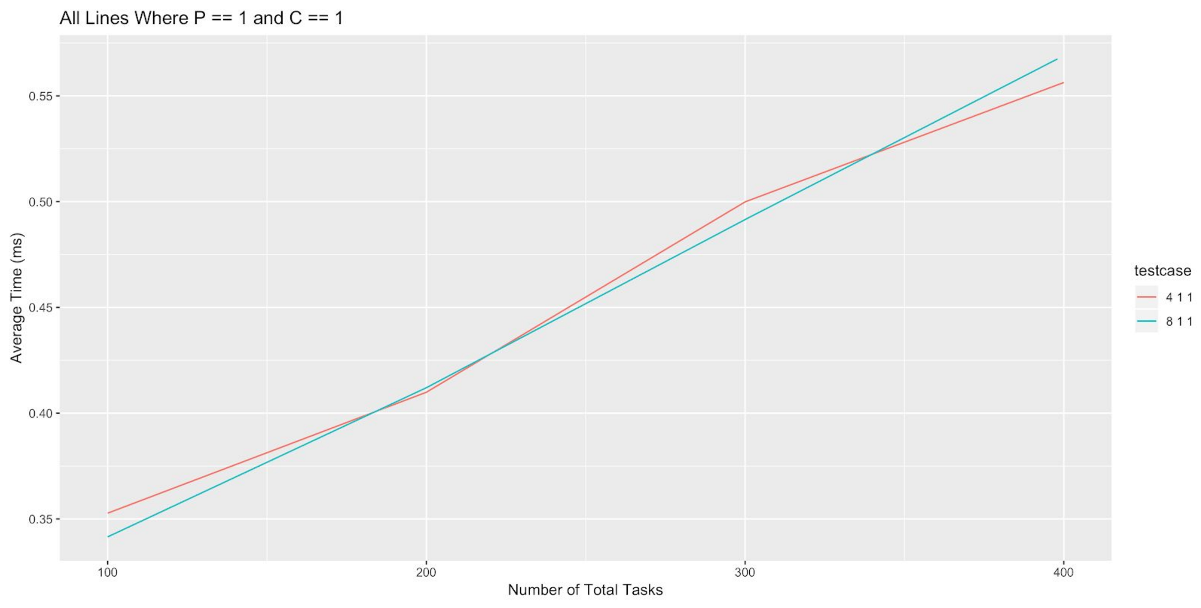

Figure 5: The effect of changing B on thread implementation execution time

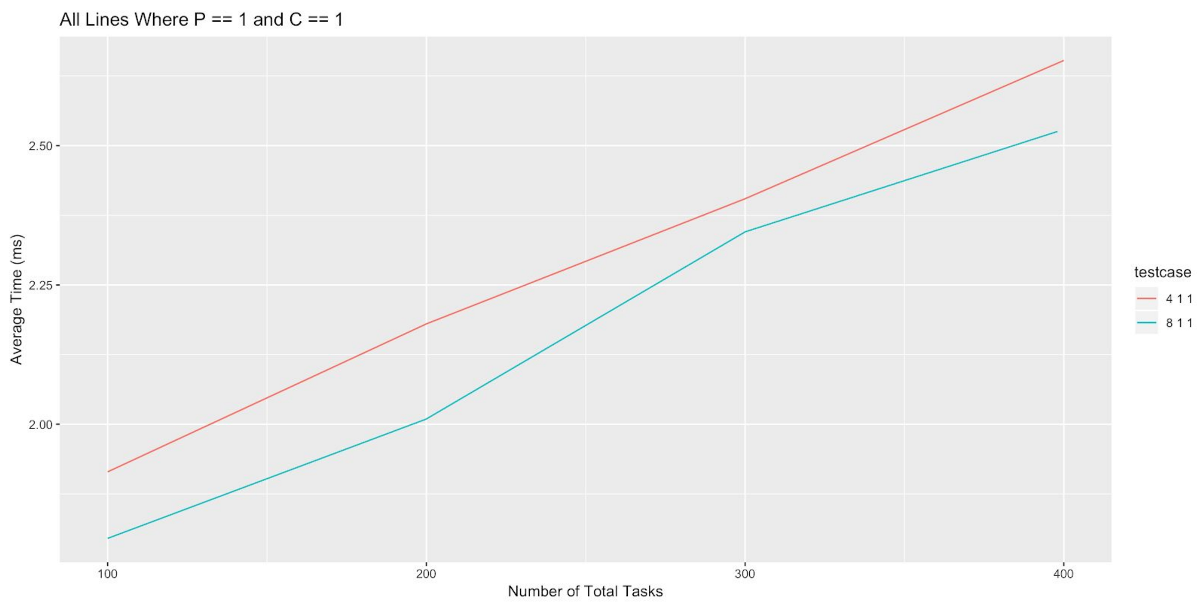

Figure 6: The effect of changing B on process implementation execution time

Figure 5 and Figure 6 show the effect that changing B has on the execution time. A bigger queue generally means that the execution time will be faster. This is because the producers will be blocked less as they wait for space to free up in the queue, saving some time.

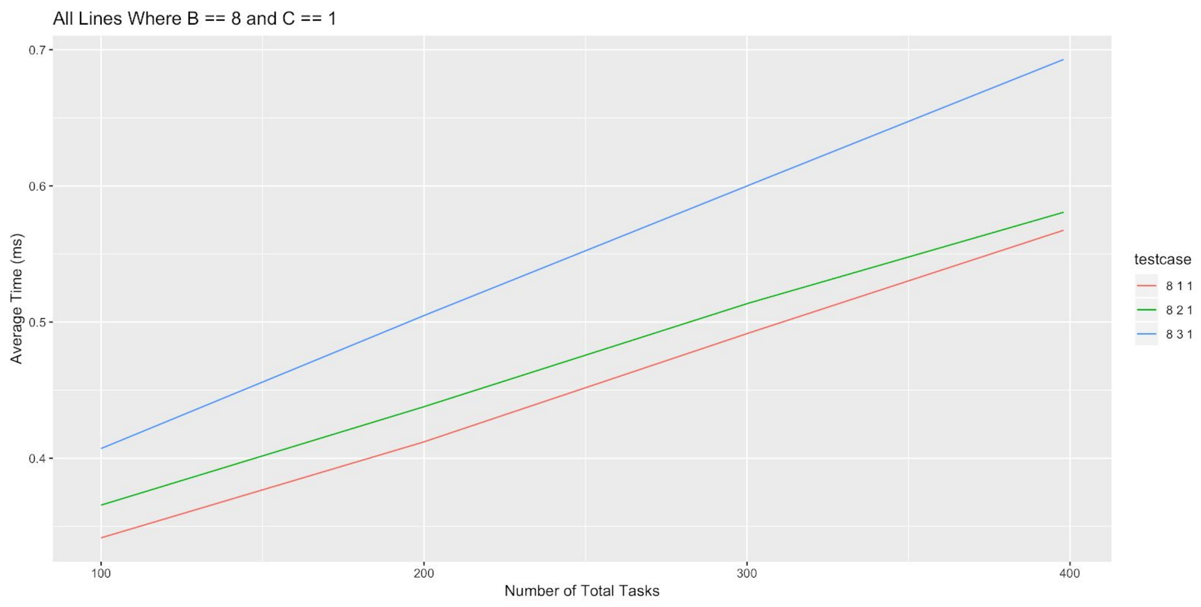

Figure 7: The effect of changing P on thread implementation execution time

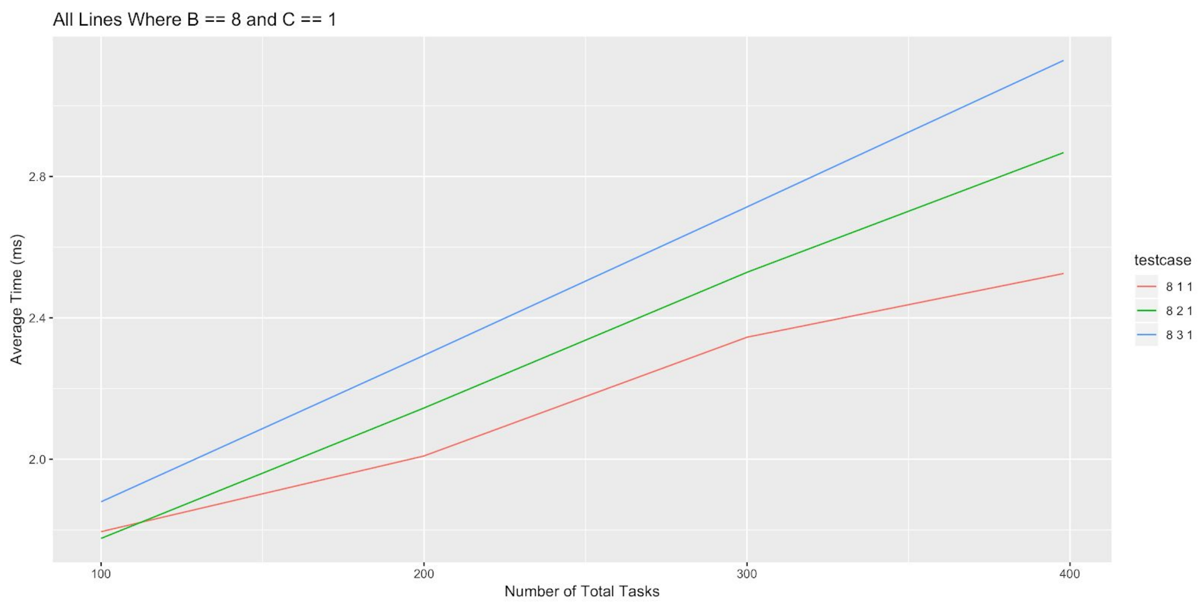

Figure 8: The effect of changing P on process implementation execution time

As can be seen in Figure 7 and Figure 8, increasing the amount of producers increases the execution time of the program. Presumably, this is because there is the additional overhead of context switching between the producer threads/processes by the operating system. In the single producer single consumer case, there is less context switching and so the code runs faster.

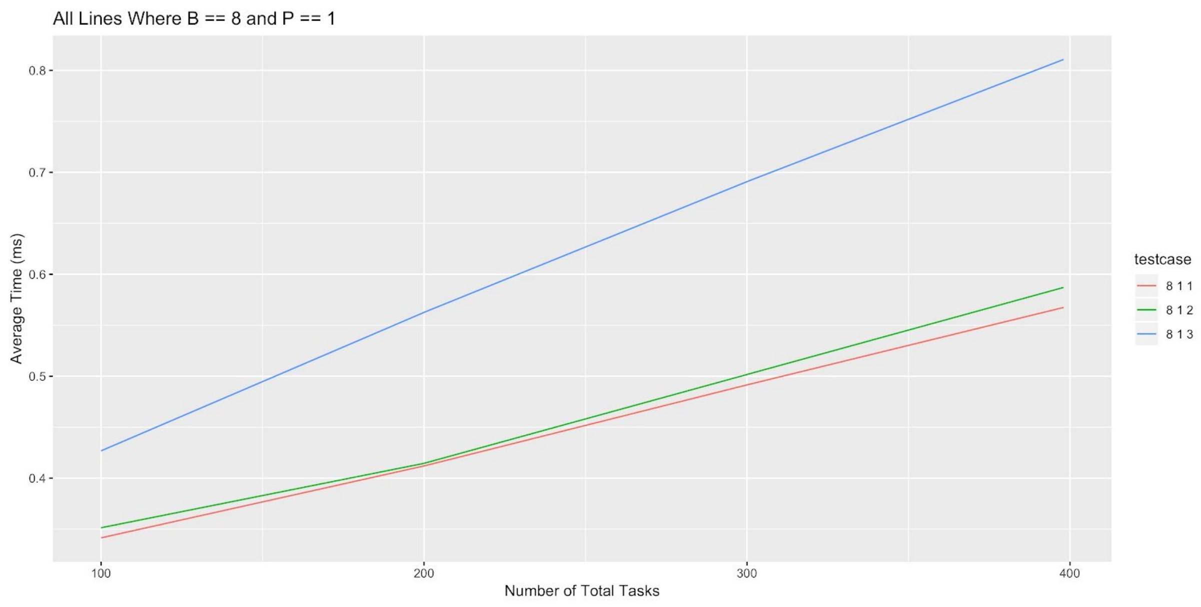

Figure 9: The effect of changing C on thread implementation execution time

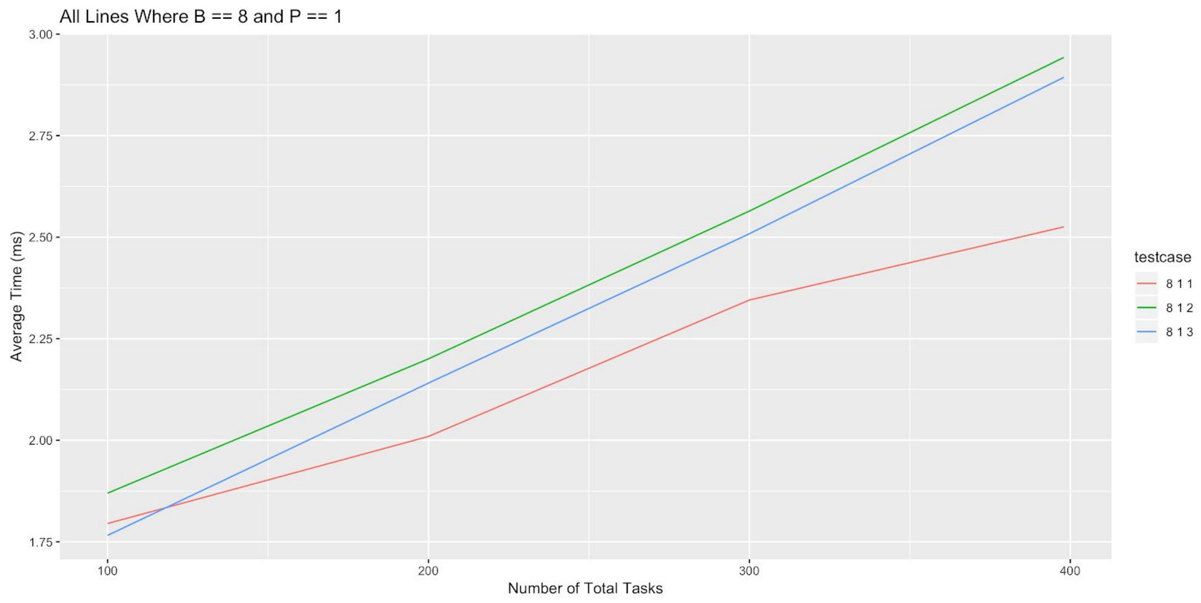

Figure 10: The effect of changing C on process implementation execution time

Similarly to the effects of changing P, increasing the amount of consumers also increases the execution time, as shown in Figure 9 and Figure 10. Again, this is assumed to be because of the additional context switching that must be done.