

Fibonacci Sequence: Single vs Multithreaded

Introduction:

Through this lab I experimented with multithreaded programming implementing the calculation of the fibonacci sequence. Two programs were implemented in C, the first being a traditional procedural program without multithreading and the second using the pthread library in C. The goal of this lab was to experiment and witness firsthand how multithreading would impact performance. Since the numbers in the fibonacci sequence increase very rapidly I set a limit of only calculating and printing the first one hundred numbers to prevent any overflows from happening in the data types I used.

Analysis:

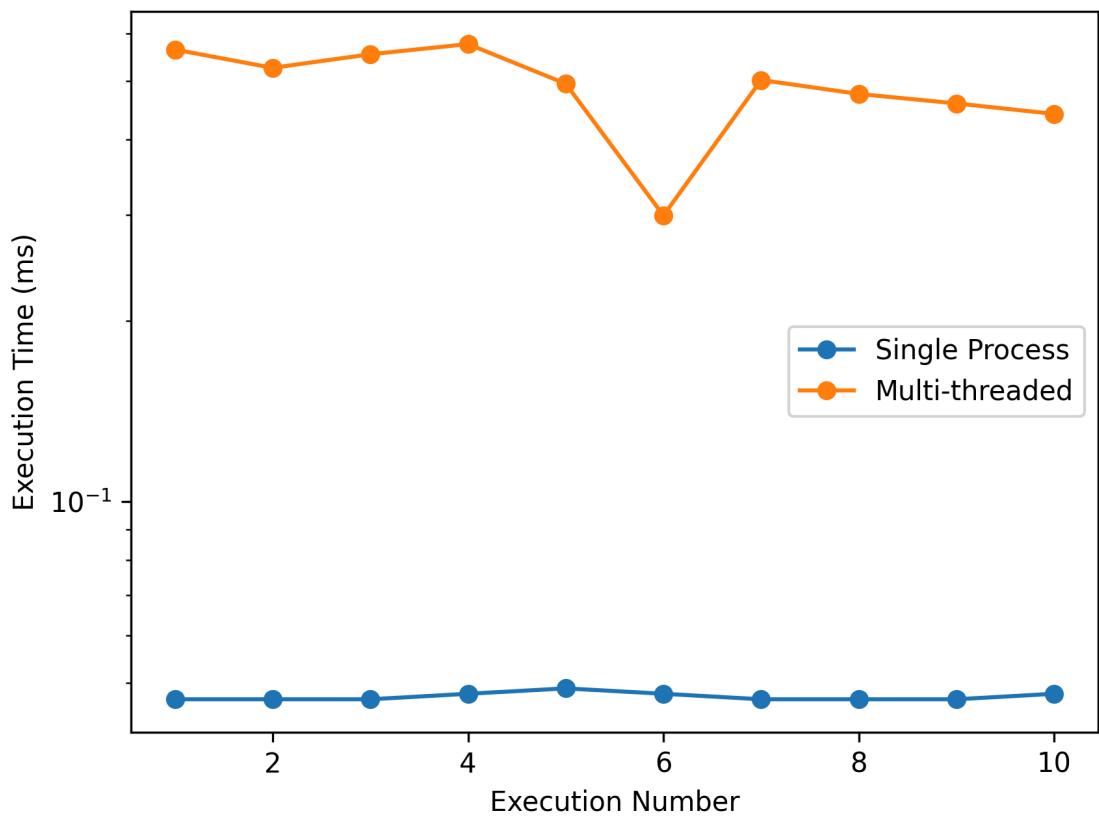
From the data that I was able to gather it is easy to see that the non multithreaded program significantly outperformed the multithreaded program. This was contrary to what I had hypothesized at the beginning of this experiment. Upon the further research I conducted to try and gain some understanding of why I ended up with these results I discovered a few things that lead me to the results I got. Firstly, I found that the type of problem I tried to solve was not suited for multithreading. This is because the fibonacci series is a problem that is sequential in nature. The next result in the sequence is entirely dependent on the previous two results $[(n-1)+(n-2)]$. Due to this when I was using a multithreaded approach all that I was doing was creating two threads waiting for one to finish calculating the sequence up to one hundred then letting the second thread do the same thing. After each thread was done calculating the sequence to one hundred they were then merged but this whole process that I did of creating threads and merging them was not free there was an overhead to all of it which can be seen when comparing the times of the multithreaded program against the non multithreaded program. The lack of overhead that the non multithreaded program had is why its performance was so much better. Perhaps this approach could work on the fibonacci sequence if I were calculating much larger numbers and if I was using a different algorithm like the matrix exponentiation approach which would give me a more efficient time complexity but there would still be overhead with merging the threads since I would need the previous results of the sequence to calculate the next result.

Conclusion:

To summarize, I found out that for sequential problems like the fibonacci series, especially for a small number of the series like one hundred in my case, multithreading is not the best approach. I found out that multithreading really shines when you are dealing with a problem that has tasks that can run in parallel with each other and where the overhead of thread creation and merging is justified. For problems like the fibonacci series the non multithreaded approach was the better performing one because of the lack of overhead that its counterpart had.

Run	Single(ms)	Threaded(ms)
1	0.047	0.564
2	0.047	0.526
3	0.047	0.554
4	0.048	0.576
5	0.049	0.495
6	0.048	0.299
7	0.047	0.502
8	0.047	0.476
9	0.047	0.459
10	0.048	0.441

Fibonacci Execution Times Across Runs



Average Performance: Single vs Multi-threaded

