CS342-Project #1 Report

Integral.c

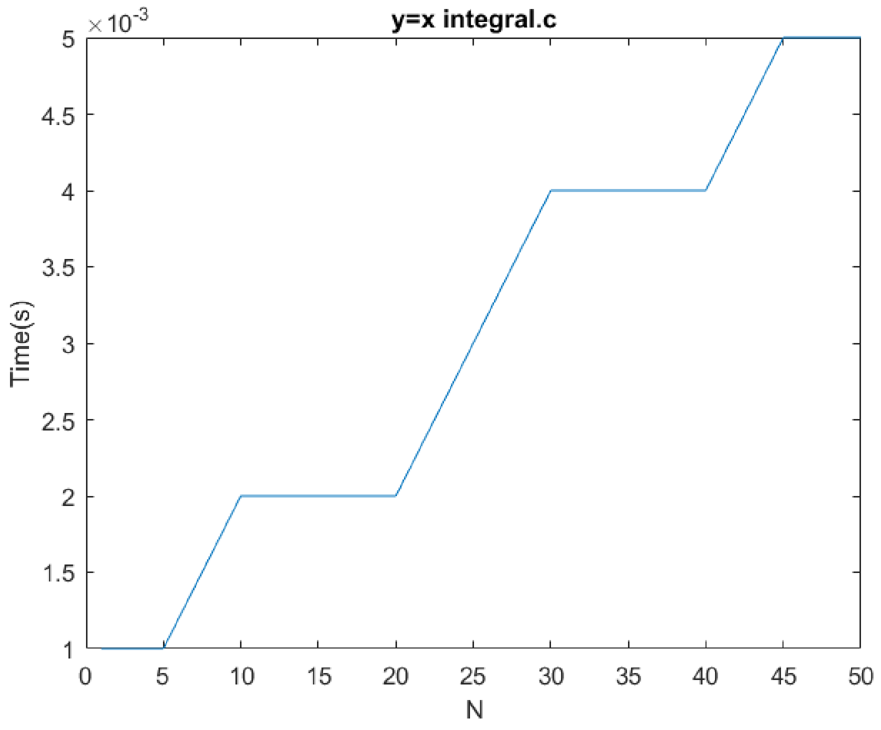
For the multi-process application I chose N values as (1,5,10,15,...,50). Since it was reported in the homework document that the biggest value N could take is 50 I stick with the number and having 10 experiments in total I divided this range of values to 10 and tried the program with these values. Since the variables are only N and functions lower and upper bound and K value are fixed and are valued as below:

L=0 , U =10 , K=20

In each experiment the output values for each N value is reported as values of y and then the resulting plot is included.

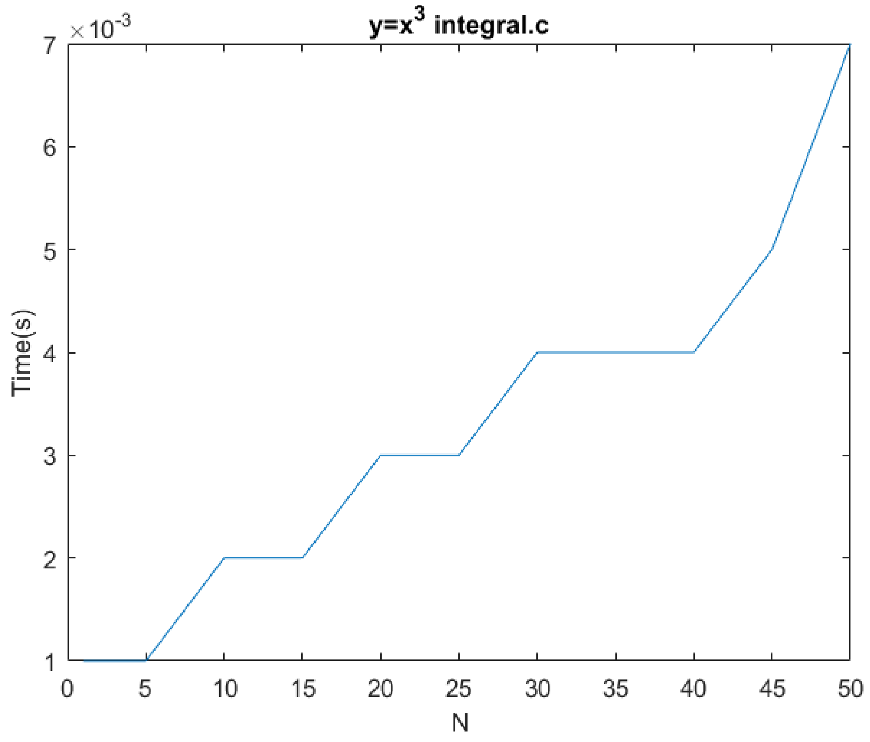
Y=X

y = (0,001 0,001 0,002 0,002 0,002 0,003 0,003 0,004 0,004 0,004 0,005 0,005) seconds.



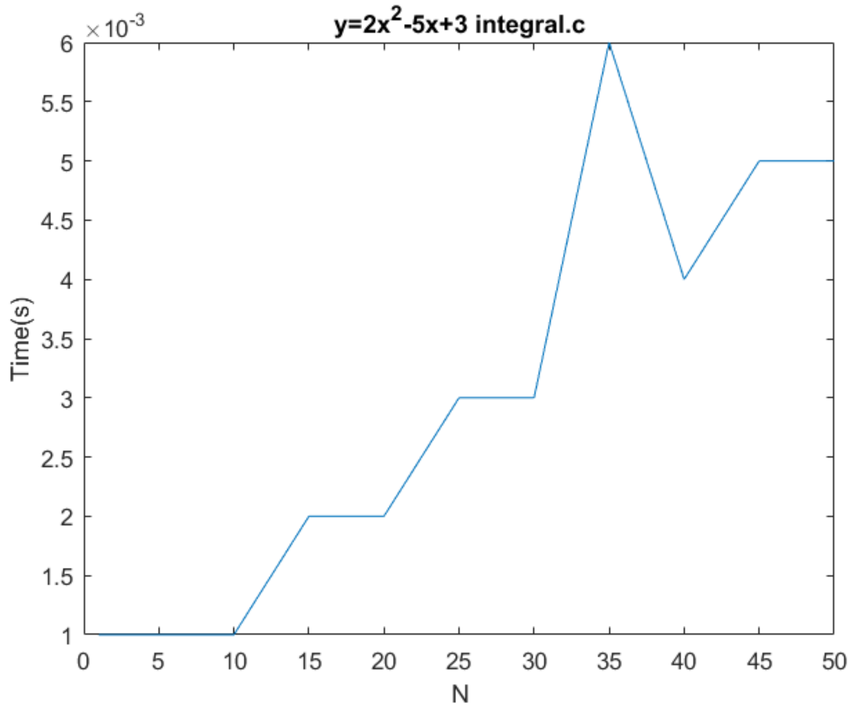
Y=X^3

y = (0,001 0,001 0,002 0,002 0,003 0,003 0,003 0,004 0,004 0,004 0,005 0,007) seconds.



Y=2X^2-5X+3

y =(0,001 0,001 0,001 0,002 0,002 0,003 0,003 0,006 0,004 0,005 0,005) seconds.



Tintegral.c

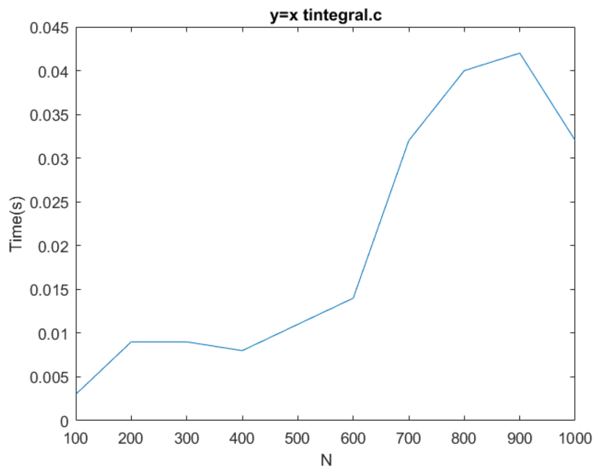
For the multi-threaded solution I chose N values as (100,200,300,...,1000). Since it was reported in the homework document that the biggest value N could take is 1000 I stick with the number and having 10 experiments in total I divided this range of values to 10 and tried the program with these values. Since the variables are only N and functions lower and upper bound and K value are fixed and are valued as below:

L=0 , U =10 , K=20

In each experiment the output values for each N value is reported as values of y and then the resulting plot is included.

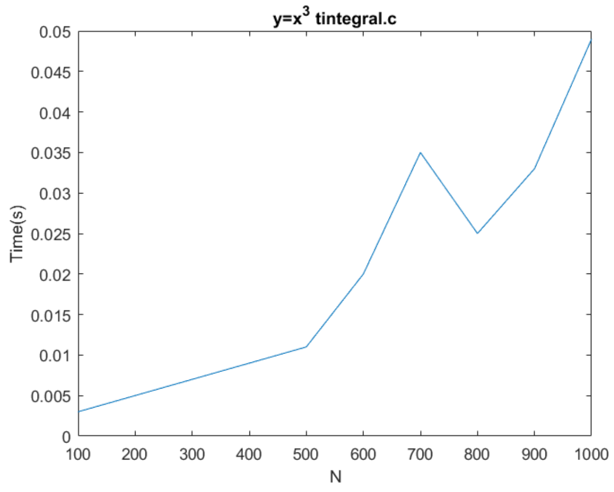
Y=X

y=(0,003 0,009 0,008 0,011 0,014 0,032 0,040 0,042 0,032) seconds.



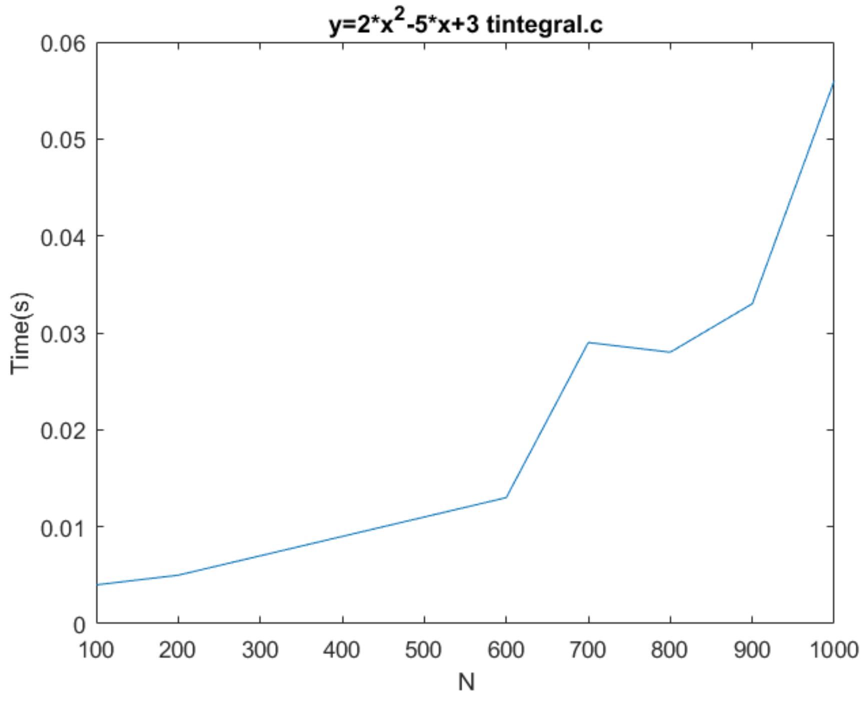
Y=X^3

y=(0,003 0,005 0,007 0,009 0,011 0,020 0,035 0,025 0,033 0,049) seconds.



Y=2X^2-5X+3

y=(0,004 0,005 0,007 0,009 0,011 0,013 0,029 0,028 0,033 0,056) seconds.



Conclusion

According to results observed for the multi-processed program one would not be wrong to say that as the number of child processes increase, that is N in this case, the execution time of the program increases as well. This seems pretty reasonable since there is not enough processors for each process, the time spent on context switch causes a lot of overhead resulting in waste of time eventually.

The occasion explained above is also true for multi-threaded program thus as N increases it slows down.

For both programs though, the precision increases with the increasing N which is a result of the theorem being used in the implementation. As one divides the area under a function more and more the less space he/she would lose due to sharp corners of the trapezoids that cannot fit the function properly.

If we were to compare two solutions that are implemented we could easily say that the one implemented with threads is much more efficient in terms of time. We can see that from the results above too. The biggest N we use for multi-processed programs is 50 and their average time on that N is around 0,006 seconds. For multi-threaded programs we start the experiment with N=100 that is twice as the multi-processed program's biggest N value. Even though the N value is twice we see that the average time response of the program to this N value is around 0.003 that is less than multi-processed ones. Moreover even though not included in the report just to observe it I tried multi-processed program with N=500 which took about 0.095 seconds which is 9 times greater than the multi-threaded program with the same N value. Thus we could say multi threading at least for the environment I am using right now is a more efficient way to implement this program.