Integration of Trapezoidal Rule under Multiple Processes

The value that I used for n was 43,710,000. Please note that this number is different from the value that I've found in P2a. This number is more accurate due to the fact that in P2a, I did not declare literals correctly, which resulted in an incorrect tmin as well as an incorrect absolute relative error. For this program the absolute relative error for one process is 4.839292384035312719e-15. This number is indeed smaller than 5e-15, so the accuracy of the integration result is within the specs to 14 significant figures.

Table for Timings of All Runs with Minimum Times Highlighted:

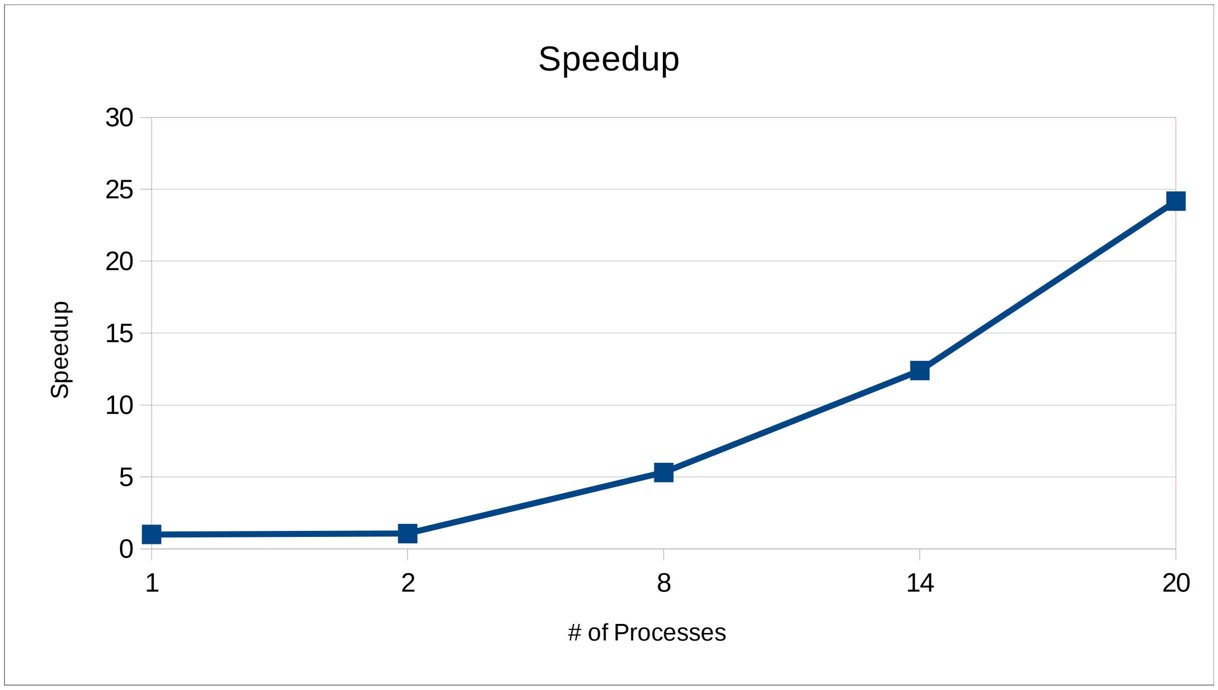
# of Processes	First Run (secs)	Second Run (secs)	Third Run (secs)
1	4.612659e01	4.664702e01	4.606836e01
2	4.39142e01	4.489546e01	4.327308e01
8	8.747091	8.669343	9.131057
14	3.761751	3.743794	3.718730
20	2.222776	2.123831	1.905913

Table of Minimum Time Runs Only with Other Information:

# of Processes	Min Time (secs)	Estimated Integral	Absolute Relative Error	Calculated Speedup	Calculated Efficiency
1	46.06836	4.7540192288588 e03	4.8939292384035312719 e-15	-	-
2	43.27308	4.7540192288588 e03	5.1974294501915265825 e-15	1.064596	0.532298
8	8.669343	4.7540192288588 e03	5.1420352902683873326 e-15	5.313939	0.664242
14	3.718730	4.7540177834746 e03	3.0403415984921945922 e-07	12.38819	0.884871
20	1.905913	4.7540192288588 e03	5.1216711640234557185 e-15	24.17128	1.208564

Looking at 20 processes, looks like we have perfect efficiency considering we only timed tasks that rank 0 was involved with.

Graph of Speedup:



Graph of Efficiency:

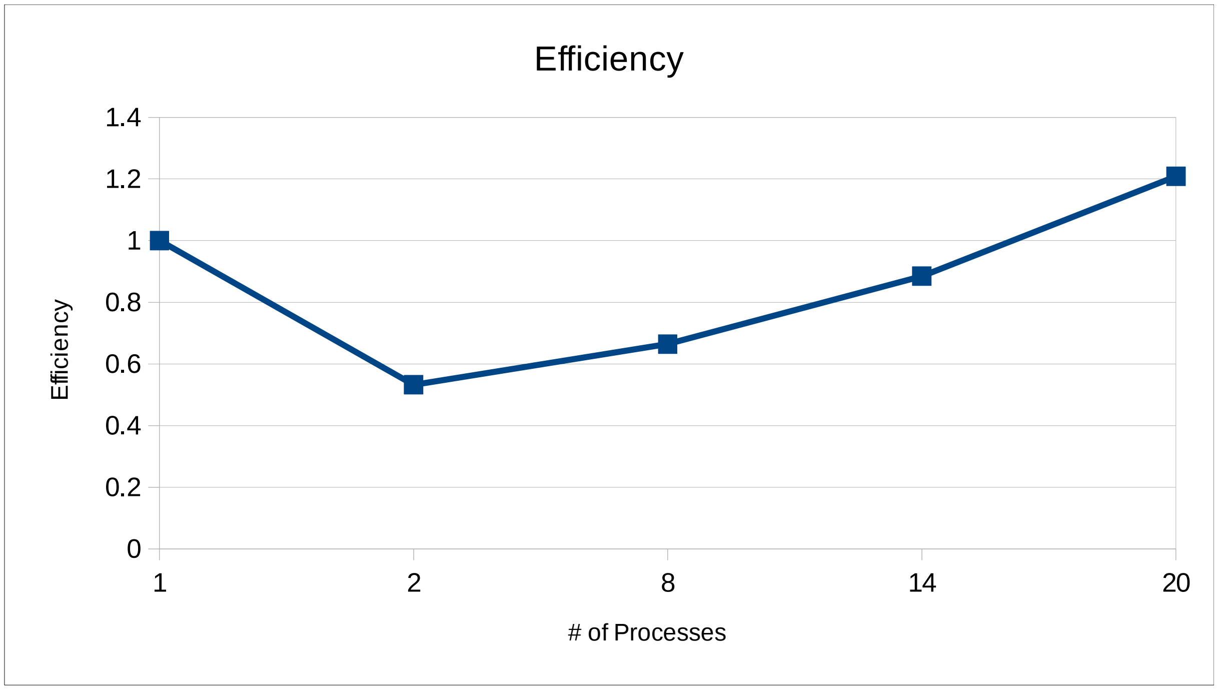
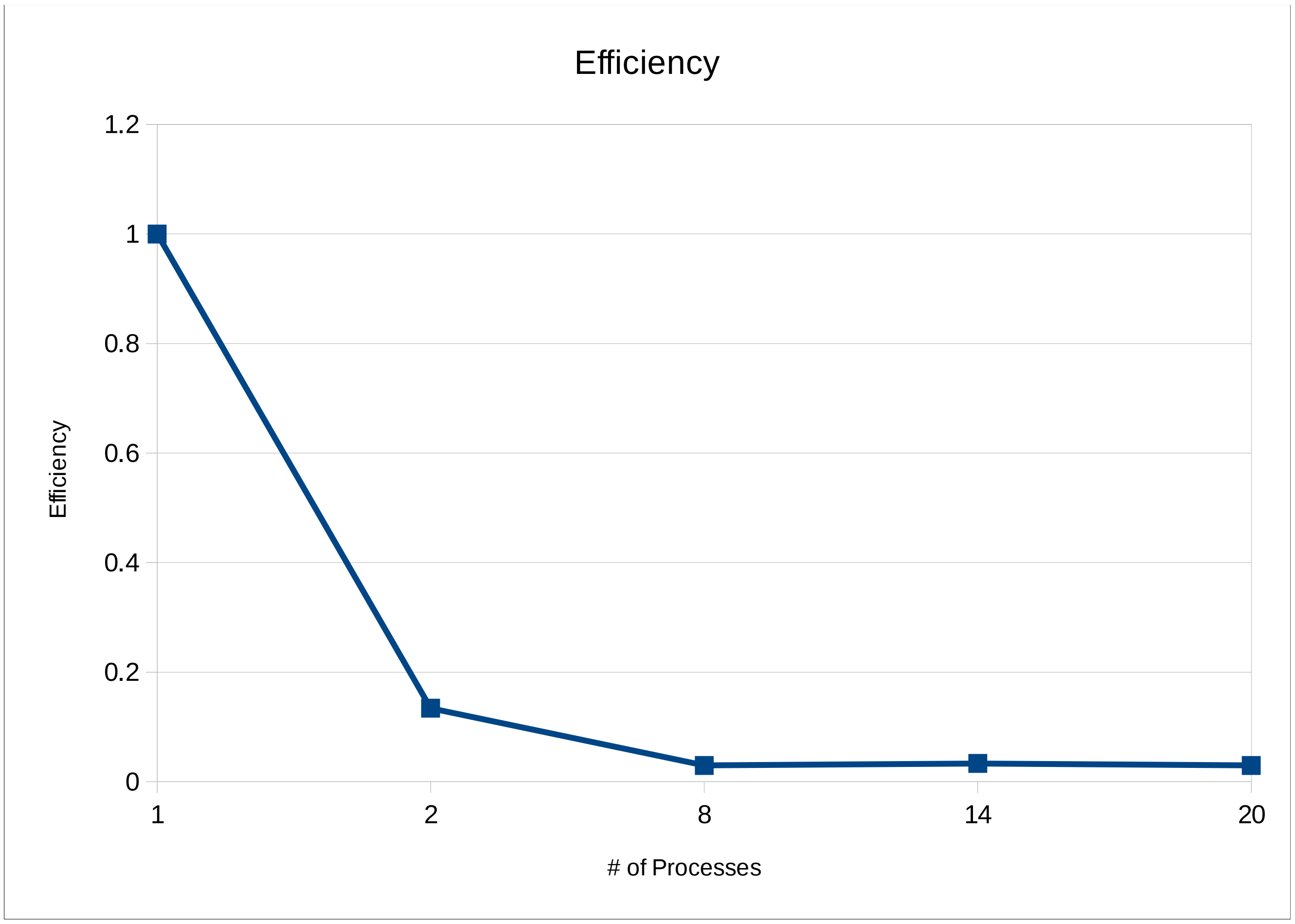


Table for Results with N Scaled with The Number of Processes:

# of Processes	N	First Run (secs)	Second Run (secs)	Third Run (secs)	Speedup	Efficiency
1	43,710,000	46.12659	46.64702	46.06836	-	-
2	87,420,000	171.8359	173.4400	171.7151	0.268284	0.134142
8	349,680,000	193.3661	194.9062	193.7199	0.23824	0.02978
14	611,940,000	99.15137	102.0937	98.73724	0.46658	0.03333
20	874,200,000	80.20819	82.27030	77.78890	0.59222	0.029611

Graph for Scaled Efficiency Resulting From Above:



Conclusion:

So for the case that the problem size stays the same, this program's speedup seems to jump up as we add the number of processes. According to the second table, the program seem to have achieved perfect efficiency with 20 processors. This was very surprising to me because this is not very likely to happen. However, I think that since I implemented the derived data structure to broadcast the datasets to the other processors only once, I have eliminated a lot of overhead. Each send and receive will take up that much more overhead and with 20 processors, thus this should have a lower efficiency if done without the derived data type. This seems to suggest that this program is strongly scalable.

Scaling the number of trapezoids to the number of processors, we notice that the efficiency decreases as n increases and the number of processors grows to 8. However, we also notice that the run time starts to come down as the problem size and processors starts to increase. However, the efficiency starts to decrease as well. I think with this problem is that the problem size is growing way too much for the processes to handle and that's why we see a decrease in efficiency but a lower run time. If we were to grow the number of size and allow the number of processes to grow much higher, we shall see a better efficiency result. Thus, everything seems to be hinting that if we increase the number of processes our run time will just get better. And if we have a fixed problem size, our efficiency result will also be that much greater. This leads me to conclude that this program is strongly scalable.