Oriah Ulrich
CSCI 551
Assignment 2

Using the find_min_trapezoids program, i found the minimum trapezoids necessary to approximate the integral of the function to within the accepted relative error 2.4976e-14. The approximation is within 1e-14.

find_min_trapezoids works by finding a sub-interval that probably has the root value N where the error is close to the accepted error. Then it uses a binary search to find the minimum N where the error is as close to the accepted error as possible.

**N** = 4709890 (which accurately approximates the integral within the following relative error)

| | | |
|---|---|---|
| **True Error** | = | 0.000 000 000 09992695560 |
| **Relative True Error** | = | 0.00000000000002495852 |

After finding this minimum N value, I ran the integration in parallel using the "parallel_integration" program. This program is run with the "run" program and can also be run with the "run_test" script which automates running the parallel program by using lists specified in the script. The processor list consist of constant values used to represent the number of processors used in each iteration. The trapezoid list contains the constant values used to represent the number of trapezoids to use in each iteration. To test the parallel program it sends each combination of processor counts and trapezoid numbers to the parallel program. The parallel program is programmed to output data to a file called "points.out".

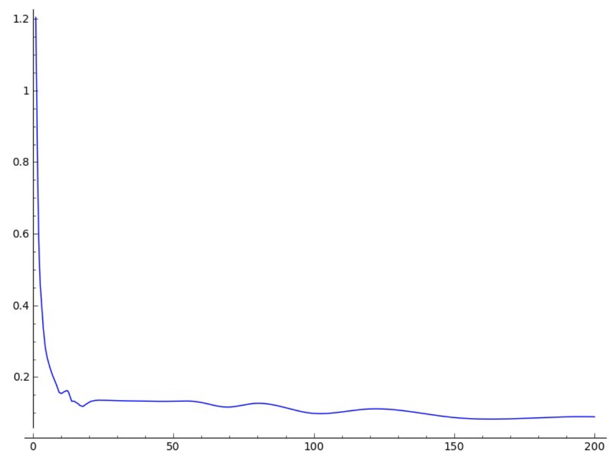
After running the series of tests as described, I decided to take a look at several graphical representations of the data. First i chose a line graph that plots the time vs processors and time vs trapezoids. They turned out as expected. I describe the graphs in captions.

## Timing Table Graphs:

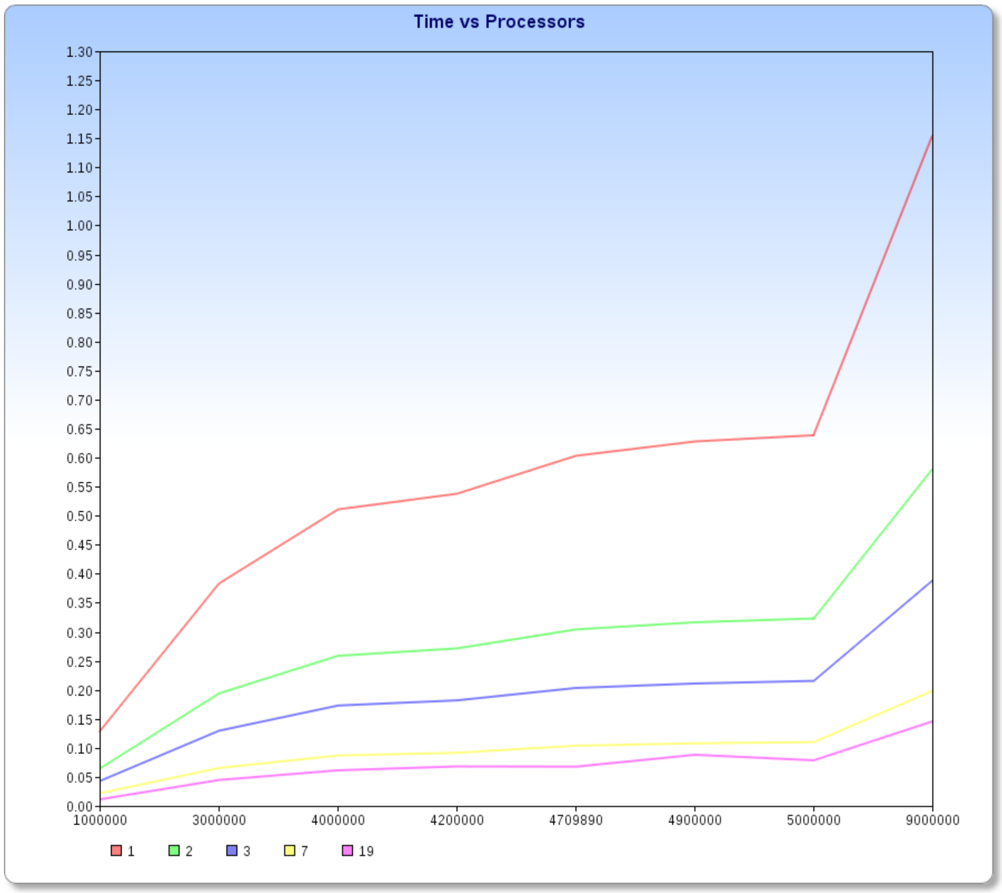
(Shows the total average time it took to execute the approximation)

### Time [s] vs #Processors:

This shows a possible asymptotic limit of the time it takes to compute an approximation of an integral with a constant number of trapezoids on variable number of processors. The height of the graph is time. The horizontal axis is the number of processes used.



## Time [s]  vs  #Trapazoids:



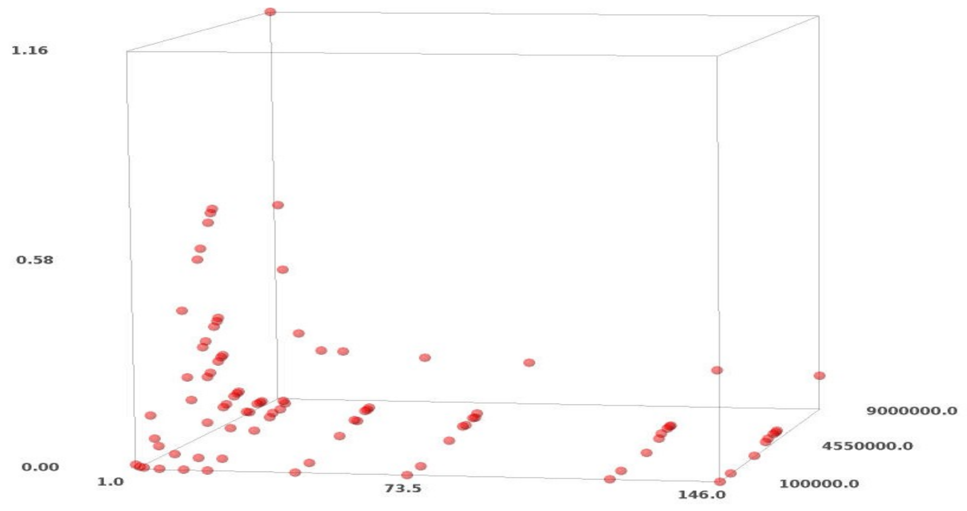
This is pretty interesting. It shows a sort of diminishing returns from the number of processors used. The number of processors happens to be increased as the time it takes to run the approximation decreases. However it seems from this graph that the rate at which the time decreases as processors increases is not linear and is decreases (t``(p) < 0) even though (t`(p) > 0). This means there might be a limit to how much time is saved regardless of how many processors are thrown at this problem.

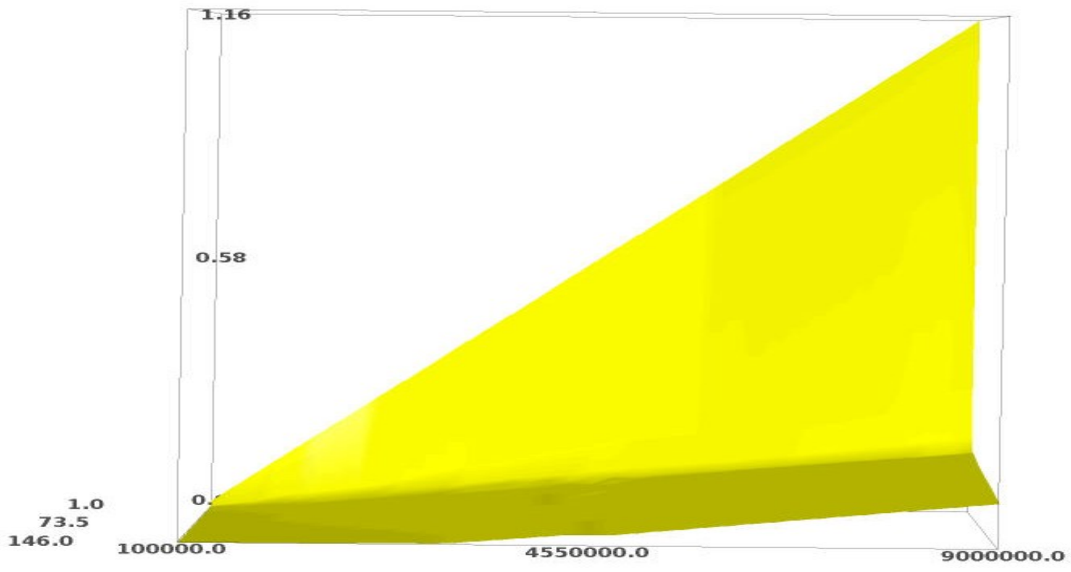
**Note**: 9e6 trapezoids is a bad example in this graph. I need to plot more points in between 5e6 and 9e6 to show the trend.

**Note**: In this graph, the individual curves are not good visualizations of the rate of change of time vs number of trapezoids used, since the subintervals for the x-axis are not visually equidistant. You really cannot tell how it changes based on the visual representation since the numbers are bunched.

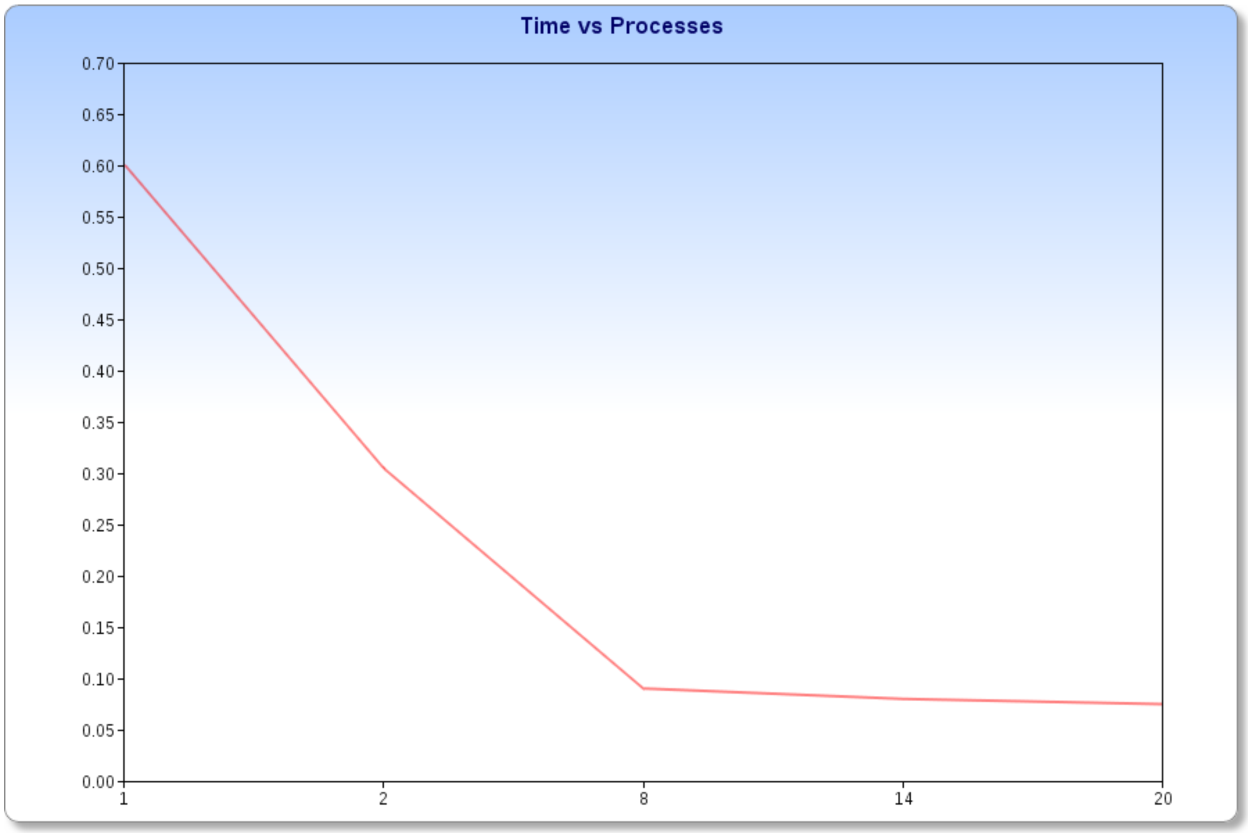
## 3d interpolation graphs:

The height of these graphs represents the time it took to execute at different parameters. The varying parameters that i had cared about were the number of trapezoids used to approximate the integral and also the number of processors used. In the graph below the bottom horizontal (x-axis) represents the number of processors used in the approximation. The depth of the graph represents the number of trapezoids used to approximate the integral.





## Speed up and Efficiency:



There were 5 configurations. In each configuration I ran 5 trials for each. In each trial I found the minimum time and sent it to the master process to take the average minimum time. The master process would calculate the average minimum time and this would be the time you see on the graph for each configuration.  So with 1 processor used to calculate the integral, the time it took one processor was 0.6 seconds. With 2 processors/processes it took roughly have of that time. The more processors used the less the time it took.

## Speedup:

T1 = 0.601292

**Tn =** {0.304892, 0.090201, 0.080029, 0.074960} for processors **p** = {2, 8, 14, 20}

**Sn =** {1.972147515, 6.666208426, 7.514271432, 8.021504803}

## Efficiency:

**En = Sn/p**

**En = {0.986073757, 0.833276053, 0.536733674, 0.40107524}**

**Conclusions:**

With what had been said so far and the data given from the efficiency, it is reasonable to determine that there is definitely a sweet spot number of processors to use where we gain the most benefit. As a user i enjoyed running this with at least 4 to 10 processes or so because the integration was much faster than was with 1 processor. Even with two processes the difference was significant. However, as i tested the program with more than 50 or even 100s of processes, i didn’t really see much benefit as a result. I wonder what it would look like to examine the minimum number of processors needed to provide a decent user experience in extreme circumstances.