Yaseen Aniss
CSCI 551

Assignment 2 Parallel Numerical Integration

1.) Value of n used, and a discussion of how you arrived at this value for n

Value for n = 2032143

The method I used to get the minimum value for 'n' is to first hardcode some values for 'n' and examine the relative true error and the precision in digits. First I tried some smaller numbers like 100000, 500000, 750000 and then I slowly increased these numbers until I noticed that I was getting close to a possible value that may give me 14 significant digits. Soon I began to notice a pattern; when I increased the value for 'n', my resulting calculation would decrease to a result that is closer to the actual value. Once I tried the value 2000000 for n, I got a result that had a precision of 13 sig figs and was off by +.0000000005. So from there I knew I was close. Next, I set my initial value for num_traps as 2000000 and tried a brute force approach by creating an infinite while loop and incrementing by 1 over and over and testing until the resulting value until we can be sure we have reached a precision of 14 sig figs.

My method for making sure the value for 'n' is within the precision of 14 sig figs is to use a function called 'is_goal_reached' that checks if (1)The absolute relative true error is less than the minimum accepted error AND (2)the approximate value that we calculated minus the true value is equal to 0 in the first 14 digits.

For example :

If the approximate value is 1234.56789012349090 and the true value is 1234.56789012348979

Than the true error is 9.2086339383283e-16 which is less than 5.0e-15
AND
(1234.56789012349090) – (1234.56789012348979) = 0000.00000000000000XXXX
Which has 14 leading zeros.

This is a safe indicator that we have found a value for 'n' that will give us an answer that is precise for the first 14 significant figures.

Yaseen Aniss
CSCI 551

2.) Absolute relative true error using n trapezoids

Absolute relative true error: 1.8172989066017e-15

To find the absolute relative true error, I have a function defined in project.c called ‘calc_abs_true_error’

It returns a double (the calculated error) and it also takes two arguments which are the true value for the integral and the approximate value we are using to calculate the error. The result of the following calculation

Absolute value of (TRUE_VALUE of the integral - APPROX_VALUE / TRUE_VALUE of the integral)

Is the absolute relative true error.

Yaseen Aniss
CSCI 551

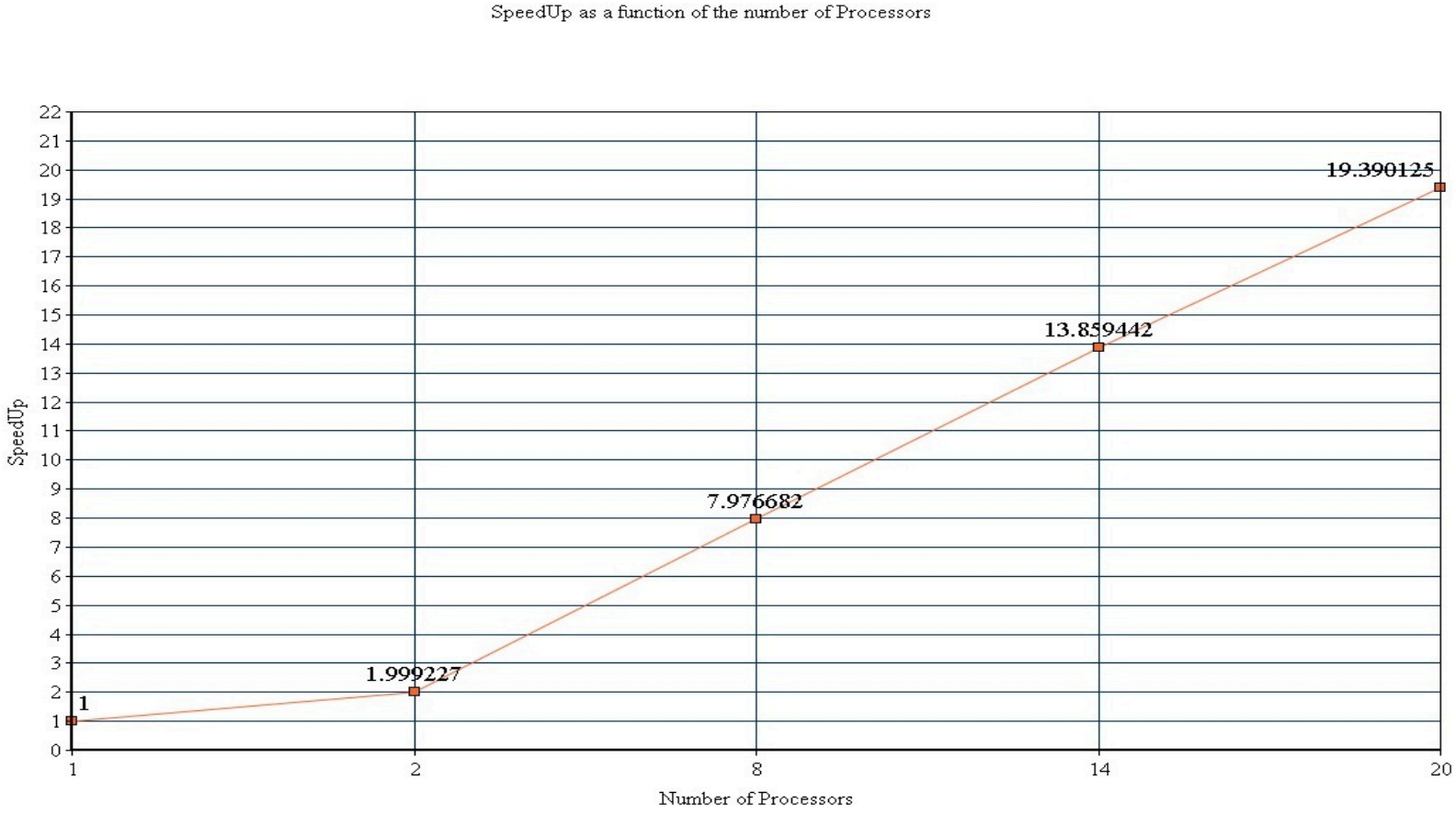
3.) Tables of timings from **all** runs

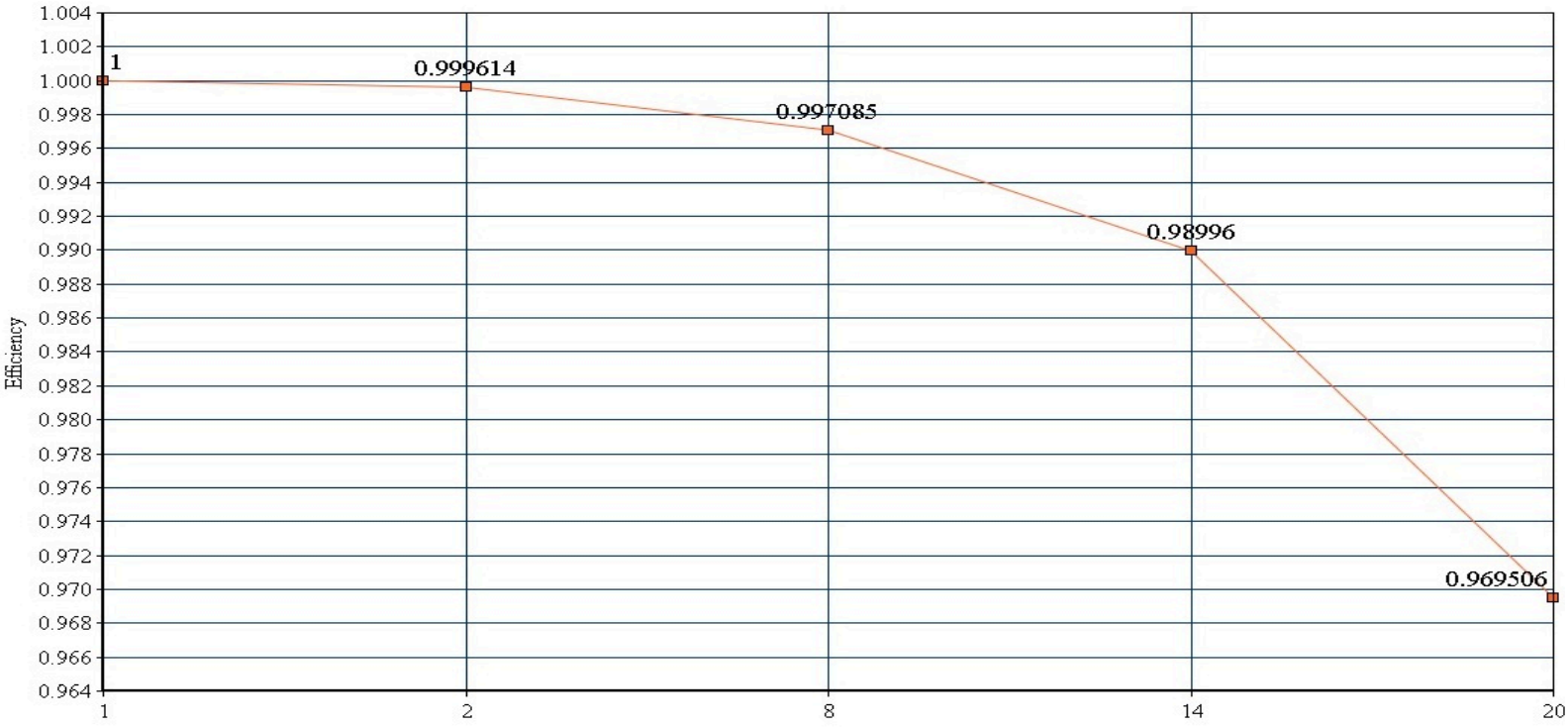
Number of Trapezoids	Number of Processors (Times are in Seconds)				
	1	2	8	14	20
n /16 = 127008	0.024782896	0.012411833	0.0031621456	0.0019240379	0.0016508102
	0.024756193	0.012367964	0.0032148361	0.001967907	-> 0.001502037 <-
	0.024775982	-> 0.012353897 <-	0.0031881332	0.0020000935	0.0016181469
	0.024770021	0.012356997	0.0032069683	-> 0.0018820763 <-	0.0015308857
	-> 0.024723053 <-	0.012377977	-> 0.0031561852 <-	0.0019640923	0.0015149117
n /8 = 254017	0.049298048	-> 0.024646044 <-	0.0062861443	0.0037519932	0.0028400421
	0.049288034	0.024665117	0.0062460899	0.0038979053	0.0028529167
	0.049466848	0.025751114	0.0062339306	0.003921032	-> 0.0026330948 <
	-> 0.049259186 <-	0.024761915	-> 0.0062170029 <-	-> 0.0036840439 <-	0.0027871132
	0.049611807	0.024711132	0.0062258244	0.003813982	0.0027520657
n/4 = 508035	0.098493099	0.049347162	-> 0.012435913 <-	0.0073430538	0.0053658485
	0.098505974	0.049386978	0.012657881	-> 0.0073111057 <-	0.0053639412
	0.09858799	0.051515102	0.012702942	0.0074529648	-> 0.0052301884 <
	0.099032879	0.049380064	0.012506008	0.0073490143	0.0056140423
	-> 0.098491192 <-	-> 0.04927206 <-	0.012554169	0.0080051422	0.0052578449
n/2 = 1016071	0.20180917	0.098497868	-> 0.024767876 <-	0.015098095	0.010476112
	-> 0.19681001 <-	0.098582983	0.02530694	0.014441013	0.010531187
	0.19681406	-> 0.098464966 <-	0.024781942	0.015310049	0.010373116
	0.19692492	0.09849596	0.024804115	0.014906883	-> 0.010200977 <-
	0.19685698	0.098667145	0.025561094	-> 0.014256001 <-	0.010349035
CALCULATED n = 2032143	0.39356399	0.19724107	0.049399853	0.028875113	-> 0.020290852 <-
	0.39449215	0.20594406	0.04954505	0.030387878	0.020814896
	-> 0.39344215 <-	-> 0.19679713 <-	-> 0.049324036 <-	0.028561831	0.02038002
	0.39351892	0.196841	0.049368858	-> 0.028388023 <-	0.021991014
	0.39347506	0.19683504	0.050026894	0.02896595	0.02136302

n*2 = 4064286	0.78690195	-> 0.39360809 <-	0.1058619	0.056811094	0.040181875
	0.78682399	0.39367318	0.099101782	0.057843924	-> 0.039764881 <-
	0.78689194	0.39476204	0.099011898	0.0567348	0.042313099
	-> 0.78672409 <-	0.40171599	0.10174298	-> 0.056645155 <-	0.040555
	0.7868309	0.39369392	-> 0.098541975 <-	0.060836077	0.041640997
n*4 = 8128572	-> 1.5734391 <-	0.7870481	-> 0.19694614 <-	0.11286497	0.084768057
	1.5735722	0.78702211	0.21723795	0.11796403	-> 0.079207897 <-
	1.5735822	0.79139996	0.19770002	0.11709309	0.084671021
	1.5734951	-> 0.78700089 <-	0.20693493	-> 0.11277795 <-	0.079342127
	1.5735221	0.78957891	0.19802403	0.11307597	0.08686018
n*8 = 16257144	-> 3.1466088 <-	-> 1.5740242 <-	-> 0.39454484 <-	0.23023605	0.16934395
	3.146714	1.5828049	0.39600611	0.23051715	0.16127205
	3.1470079	1.576839	0.40425205	0.22545886	-> 0.15857601 <-
	3.1467249	1.5743301	0.39540982	0.22545695	0.16593218
	3.1467869	1.5741072	0.42348504	-> 0.22542095 <-	0.16185713
n*16 = 32514288	6.309798	3.164609	0.84563184	0.48514295	-> 0.31652904 <-
	-> 6.2931221 <-	3.1481771	0.78745508	0.45785093	0.31933212
	6.3275859	-> 3.1478159 <-	0.79092193	0.45698094	0.33886194
	6.305414	3.1576841	0.82901096	0.4840889	0.32238007
	6.2936442	3.3823991	-> 0.78732705 <-	-> 0.45287609 <-	0.32049203

Yaseen Aniss
CSCI 551

4.)Graphs of speedup and efficiency





Yaseen Aniss
CSCI 551

5.) Any conclusions you can make.

The conclusions I can make are as follows.

a.) The system does NOT exhibit perfectly linear speedup because as we added more processors, the speedup value was never equal to the number of processors (Except when proccessors == 1).

b.) The system did NOT exhibit perfect parallelization because the system only had an efficiency value of 1 when we used 1 procesor. As we added more, the efficiency went down.

c.) An important aspect to note was that speed up and efficiency were directly related in this system. As speedup went up, efficiency went down and vise versa.

d.) As the number of processors was increased, the precision of the calculated result was diminished. I believe this was a result of the minuscule rounding errors that occurred on each separate core before the values were sent to the main processor.

Yaseen Aniss
CSCI 551

6.) Discuss experiments in scaled speedup, results, conclusions

To experiment with scaled speedup, I took my calculated value of n (2032143) and I ran the parallel code for the integration using n/16, n/8, n/4, n/2, n*2, n*4, n*8 and n*16. I used all these values of n on 1, 4, 8, 14 and 20 processes while calculating elapsed time, integration result, absolute true error, speed up and efficiency (ALL this data is in "output.txt" which was created by running the ruby script "create_output.rb"). After examining the results, a very interesting realization I came across that I would have never guessed beforehand, was that increasing the number of trapezoids (n) does not ALWAYS increase the precision in the result. In fact, our result was only precise to at least 14 significant figures when we used the calculated value of n, and n*2 for the trapezoid numbers. For every other value of n, we did not get 14 significant figures of precision. Even when we used n*4, n*8, and n*16 trapezoids, the result did not get any more precise. However, it is important to note that OVERALL, increasing the number of trapezoids DID increase the precision because if you look at the absolute true error values in "output.txt" for the lower values of n (n/16, n/8, n/4, etc) , the error generally GREW as the number of trapezoids was decreased and the error LESSENED as the number of trapezoids was increased. This is the result we would normally expect. Also, after examining the efficiency values when using 20 processors, I noticed that as the value of n was increased, the efficiency also increased. This indicates that the program is weakly scalable since decreasing the value for n (the problem size) lowers efficiency and increasing the value of n increases the efficiency.