ME-766 ASSIGNMENT1

Roll Number: 170010036

1. Convergence Study:

a. Trapezoidal

I have used the No of division 10,100,1000,10000,100000,1000000 for showing the convergence. As we can see easily that only after the 1000 number of divisions the value is very nearly 2 and after that it becomes kind of constant and bit of fluctuation after 100000 number of divisions.

No of Divisions	Integral
10	1.98352349
100	1.99983561
1000	1.99999881
10000	2.00000358
100000	2.00002789
100000	2.00145435

b. **Montecarlo:**

Number of divisions used are the same as in the previous approach. Here also we can see we are getting better results as we increase sample points because the sample points generated are random so increasing the no of sampling points will increase the accuracy hence correctly calculating the value. Here we can see after 10000 points the accuracy starts saturating.

No of Divisions	Integral
10	1.80957721
100	1.96749706
1000	2.06119714
10000	1.99660577
100000	2.00194666
1000000	2.00199484

2. Timing Study:

For Timing study I have chosen 2000000 sampling points. I have chosen a high number of sampling points so the difference is visible for serial and parallel codes. Also my local machine is dual core so for 6,8 threads we can not say that time taken will be decreased.

a. **Trapezoidal:** Here in the table below we can see when increasing threads the time taken is decreasing. After 4 threads the difference is not much due to cores available in the local machine.

Number of Threads	Time taken(in sec)
2	0.0468
4	0.0408
6	0.039
8	0.0368

b. Monte Carlo: Similar to above here also the time taken is decreasing when we are increasing the number of threads from 2 to 4 but for 4 to 6 it is increasing due to the same reason I have mentioned in the previous section.

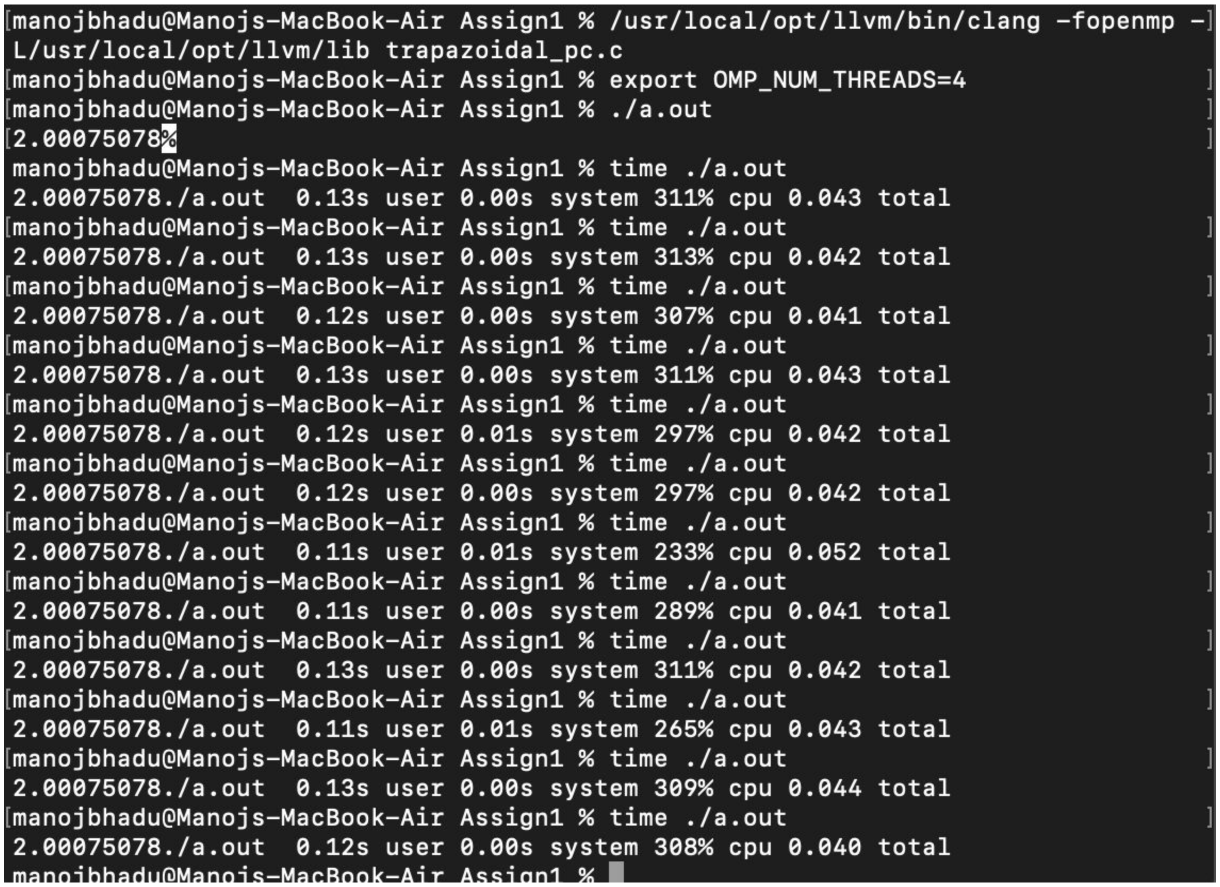
Number of Threads	Time taken(in sec)
2	0.1094
4	0.0774
6	0.0796
8	0.0782

3. Observations:

- Time taken for serial code using is ~0.075sec and parallel code is ~0.040sec using Trapezoidal Rule.So we can see around 45% decrease in time when we parallelize the code.
- Similarly time taken in serial code is ~0.097sec and parallel code is ~0.078sec using Monte Carlo Method. We can see here around 20% decrease in time when we parallelize the code
- Time taken in the Monte Carlo method is more than Trapezoidal rule due to time taken in generating the random numbers. In Monte Carlo there is an addition of time due to this.
- The % decrease is also less in monte carlo due to the same reason as previous.
- The trapezoidal rule is not as accurate as Simpson's Rule when the underlying function is smooth, because Simpson's rule uses quadratic approximations instead of linear approximations. Due to this trapezoidal rule is a bit fluctuating at high numbers of samples.

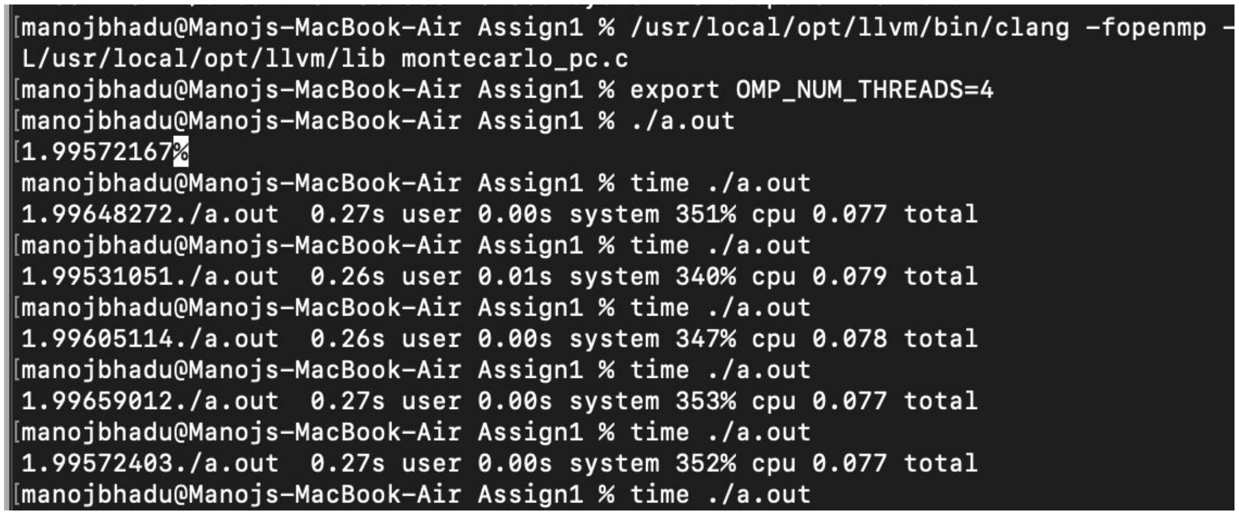
4. Some Results Screenshots:

a. Trapezoidal rule using 4 number of threads:



```
[manojbhadu@Manojs-MacBook-Air Assign1 % /usr/local/opt/llvm/bin/clang -fopenmp -]
L/usr/local/opt/llvm/lib trapazoidal_pc.c
[manojbhadu@Manojs-MacBook-Air Assign1 % export OMP_NUM_THREADS=4               ]
[manojbhadu@Manojs-MacBook-Air Assign1 % ./a.out                                ]
[2.00075078%                                                                    ]
manojbhadu@Manojs-MacBook-Air Assign1 % time ./a.out
2.00075078./a.out  0.13s user 0.00s system 311% cpu 0.043 total
[manojbhadu@Manojs-MacBook-Air Assign1 % time ./a.out                           ]
2.00075078./a.out  0.13s user 0.00s system 313% cpu 0.042 total
[manojbhadu@Manojs-MacBook-Air Assign1 % time ./a.out                           ]
2.00075078./a.out  0.12s user 0.00s system 307% cpu 0.041 total
[manojbhadu@Manojs-MacBook-Air Assign1 % time ./a.out                           ]
2.00075078./a.out  0.13s user 0.00s system 311% cpu 0.043 total
[manojbhadu@Manojs-MacBook-Air Assign1 % time ./a.out                           ]
2.00075078./a.out  0.12s user 0.01s system 297% cpu 0.042 total
[manojbhadu@Manojs-MacBook-Air Assign1 % time ./a.out                           ]
2.00075078./a.out  0.12s user 0.00s system 297% cpu 0.042 total
[manojbhadu@Manojs-MacBook-Air Assign1 % time ./a.out                           ]
2.00075078./a.out  0.11s user 0.01s system 233% cpu 0.052 total
[manojbhadu@Manojs-MacBook-Air Assign1 % time ./a.out                           ]
2.00075078./a.out  0.11s user 0.00s system 289% cpu 0.041 total
[manojbhadu@Manojs-MacBook-Air Assign1 % time ./a.out                           ]
2.00075078./a.out  0.13s user 0.00s system 311% cpu 0.042 total
[manojbhadu@Manojs-MacBook-Air Assign1 % time ./a.out                           ]
2.00075078./a.out  0.11s user 0.01s system 265% cpu 0.043 total
[manojbhadu@Manojs-MacBook-Air Assign1 % time ./a.out                           ]
2.00075078./a.out  0.13s user 0.00s system 309% cpu 0.044 total
[manojbhadu@Manojs-MacBook-Air Assign1 % time ./a.out                           ]
2.00075078./a.out  0.12s user 0.00s system 308% cpu 0.040 total
manojbhadu@Manojs-MacBook-Air Assign1 %
```

b. Monte Carlo Method using 4 number of threads:



```
[manojbhadu@Manojs-MacBook-Air Assign1 % /usr/local/opt/llvm/bin/clang -fopenmp -
L/usr/local/opt/llvm/lib montecarlo_pc.c
[manojbhadu@Manojs-MacBook-Air Assign1 % export OMP_NUM_THREADS=4
[manojbhadu@Manojs-MacBook-Air Assign1 % ./a.out
[1.99572167%
manojbhadu@Manojs-MacBook-Air Assign1 % time ./a.out
1.99648272./a.out  0.27s user 0.00s system 351% cpu 0.077 total
[manojbhadu@Manojs-MacBook-Air Assign1 % time ./a.out
1.99531051./a.out  0.26s user 0.01s system 340% cpu 0.079 total
[manojbhadu@Manojs-MacBook-Air Assign1 % time ./a.out
1.99605114./a.out  0.26s user 0.00s system 347% cpu 0.078 total
[manojbhadu@Manojs-MacBook-Air Assign1 % time ./a.out
1.99659012./a.out  0.27s user 0.00s system 353% cpu 0.077 total
[manojbhadu@Manojs-MacBook-Air Assign1 % time ./a.out
1.99572403./a.out  0.27s user 0.00s system 352% cpu 0.077 total
[manojbhadu@Manojs-MacBook-Air Assign1 % time ./a.out
```

5. Readme:

I have created four files two for serial and two for parallel using different methods of integration. Which are as follows:

1. For Monte carlo:
    - Serial code: montecarlo_sc.c
    - Parallelized code: montecarlo_pc.c
2. For Trapezoidal rule:
    - Serial code: trapazoidal_sc.c
    - Parallelized code:trapazoidal_pc.c

In these codes the number of samples is defined as N.
For running the code for different numbers of samples please change this N variable and run the c program.