ME766: HPSC
Assignment 1: OpenMP

Gopalan Iyengar Roll No.-19D170009 13/03/2021

Overview

Please see OpenMP code attached, explicit coding for serial code (number of threads=1) is not included. Merely changing the value of number of threads results in a serial implementation.

- **Composite Trapezoidal Rule:**
  - https://en.wikipedia.org/wiki/Trapezoidal_rule
  - Essentially adding up the areas of trapeziums made by two consecutive evenly spaced points within the interval, over the interval.
  - $$\int_a^b f(x)\,dx \approx \sum_{k=1}^{N} \frac{f(x_{k-1})+f(x_k)}{2}\Delta x_k = \frac{\Delta x}{2}\left(f(x_0)+2f(x_1)+2f(x_2)+2f(x_3)+2f(x_4)+\cdots+2f(x_{N-1})+f(x_N)\right)$$
- **Monte-Carlo Integration Method:**
  - https://en.wikipedia.org/wiki/Monte_Carlo_integration
  - Essentially taking n+1 random samples within the interval, averaging the value of integrand at those samples, and integrate that value over the interval.
- Please find the OpenMP implementation of the question in the file- 'hw1_19D170009.pdf'.

Convergence Study

True value of integral = **2**

Following table shows the value of integral reported by C++, for different number of subdivisions, here, 10^k, for k in {1,2,3,4,5,6,7}, averaged over 5 repeated executions. Needless to say, the value calculated will be the same for any number of threads utilized.

k---->	1	2	3	4	5	6	7
Trapezoidal Method	1.98352	1.99984	2	2	2	2	2
Monte Carlo Method	2.44944	1.99674	2.05914	2.0097	1.98245	1.99469	1.99214

From the table, we can observe that the trapezoidal integral converges way faster (~100-1000 subdivisions) than the Monte Carlo integral (10^5-10^7 subdivisions), owing to the fact that the Monte Carlo integral utilizes random samples, which give unpredictable results at lower values of divisions of the interval.

Timing Study

The following tables shows the value of execution time (in ms) of the part of the program where the integral is being calculated, for different number of subdivisions, here, 10^k, for k in {3,4,5,6,7,8,9}, and nuber of threads utilized for parallelization n for n in {2,4,6,8}, averaged over 5 repeated executions.

TRAPEZOIDAL METHOD

k ↓, n--->	2	4	6	8
3	0.201	0.114	0.097	0.122
4	0.165	0.123	0.133	0.117
5	0.675	0.488	0.453	0.417
6	6.133	3.948	3.675	2.971
7	62.537	40.185	32.577	26.965
8	611.669	384.854	312.405	274.321
9	6106.95	3791.4	3230.64	3100

MONTE CARLO METHOD

k ↓, n--->	2	4	6	8
3	0.089	0.05	0.046	0.06
4	0.295	0.23	0.191	0.207
5	2.419	1.839	1.524	1.284
6	24.031	17.874	15.758	13.222
7	240.195	177.77	146.144	121.056
8	2394.8	1739.08	1440.34	1206.56
9	24010	17370.7	14987.6	12894.2

From the above tables, we can conclude that,

1. For the most part, execution time of Monte Carlo method is higher than that of the Trapezoidal method, given all other variables(k,n) are the same.
2. As k increases, execution time increases almost proportionally, given an n.
3. There is a general trend, given a k, as n increases, execution time decreases, but not proportionally, implying that the process is not completely parallelizable with the threads we can utilize, and contains some serial character.
4. There is random fluctuation and increase in execution time while highly parallelizing(n=8) the process for very low k(2,3,4), which implies that the system is introducing certain time error into the execution due to thread scheduling, serial processes, etc., which is of the same order as the execution time for low n(~10-6 s).