National University of Computer and Emerging Sciences FAST

Project Report

Course: **Parallel and Distributed Computing**

Instructor: **Syed Faisal Ali**

Semester: Fall 2023

Project Title: **Numerical Computing Methods Analysis with OpenMP**

Group Members

Sno	ID	Name
1	21K3323	Muhammad Shaheer
2	21K3161	Muhammad Ahmed
3	21K4556	Muhammad Anas

Introduction

Problem

While numerical computing methods are powerful tools, the computational demands of large-scale problems can be time-consuming. The need for efficient solutions becomes evident, prompting the exploration of parallel implementations. The goal is to execute numerical computing methods concurrently on multiple processing units or cores, leading to improved efficiency and reduced computation time.

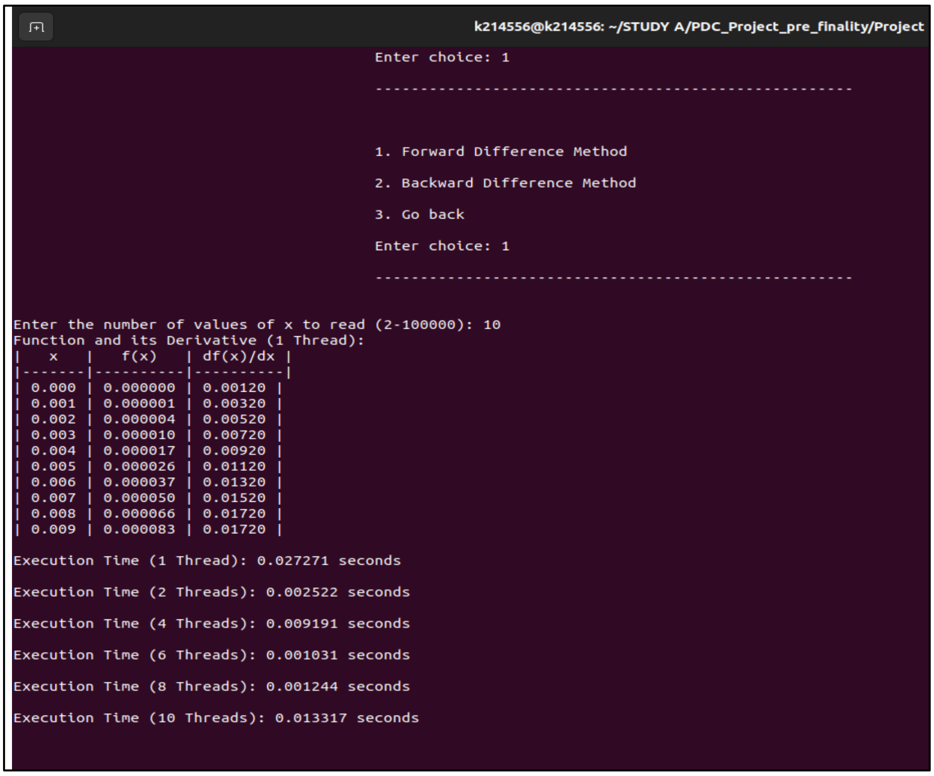
Algorithm

The project focuses on implementing the following numerical computing methods

Numerical Differentiation

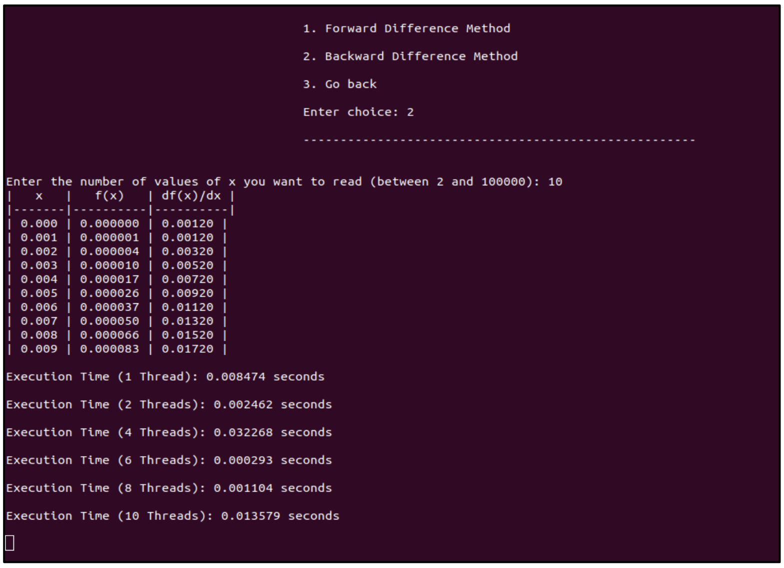
Forward Difference Method

The forward difference method is a numerical technique used to approximate the derivative of a function at a particular point. It computes the derivative by considering the function values at the given point and a small increment h. In a parallel setting using OpenMP, this method involves dividing the range of function evaluations among threads to simultaneously calculate the differences and derivatives across different intervals.



Backward Difference Method

Similar to the forward difference method, the backward difference method approximates the derivative of a function but uses backward-pointing intervals. In parallelization with OpenMP, threads can be utilized to compute backward differences concurrently across distinct intervals, enhancing computational efficiency.

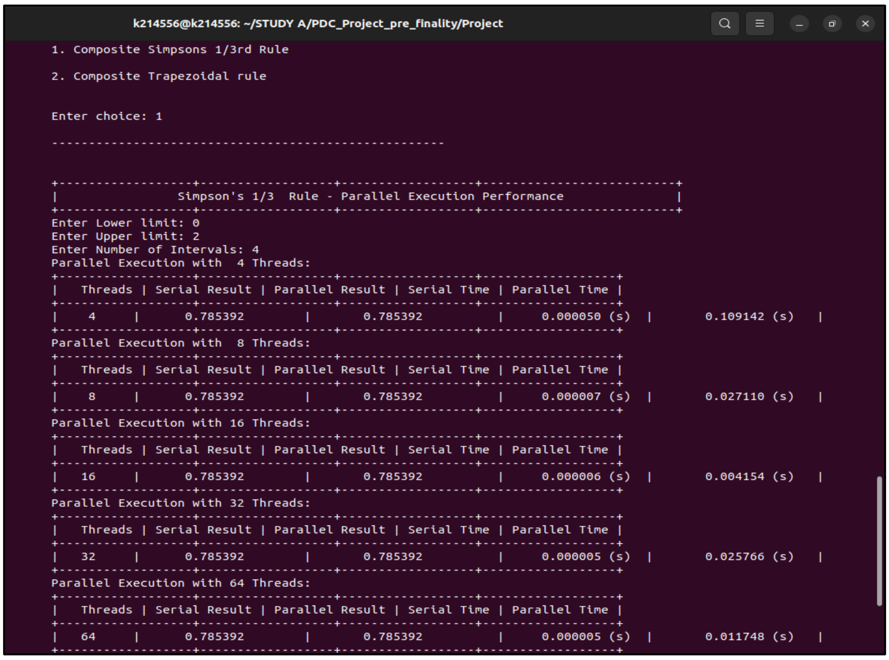


```
1. Forward Difference Method

2. Backward Difference Method

3. Go back

Enter choice: 2

--------------------------------------------------

Enter the number of values of x you want to read (between 2 and 100000): 10
|   x   |   f(x)   | df(x)/dx |
|-------|----------|----------|
| 0.000 | 0.000000 | 0.00120 |
| 0.001 | 0.000001 | 0.00120 |
| 0.002 | 0.000004 | 0.00320 |
| 0.003 | 0.000010 | 0.00520 |
| 0.004 | 0.000017 | 0.00720 |
| 0.005 | 0.000026 | 0.00920 |
| 0.006 | 0.000037 | 0.01120 |
| 0.007 | 0.000050 | 0.01320 |
| 0.008 | 0.000066 | 0.01520 |
| 0.009 | 0.000083 | 0.01720 |

Execution Time (1 Thread): 0.008474 seconds

Execution Time (2 Threads): 0.002462 seconds

Execution Time (4 Threads): 0.032268 seconds

Execution Time (6 Threads): 0.000293 seconds

Execution Time (8 Threads): 0.001104 seconds

Execution Time (10 Threads): 0.013579 seconds
```

Numerical Integration

Simpson's Method

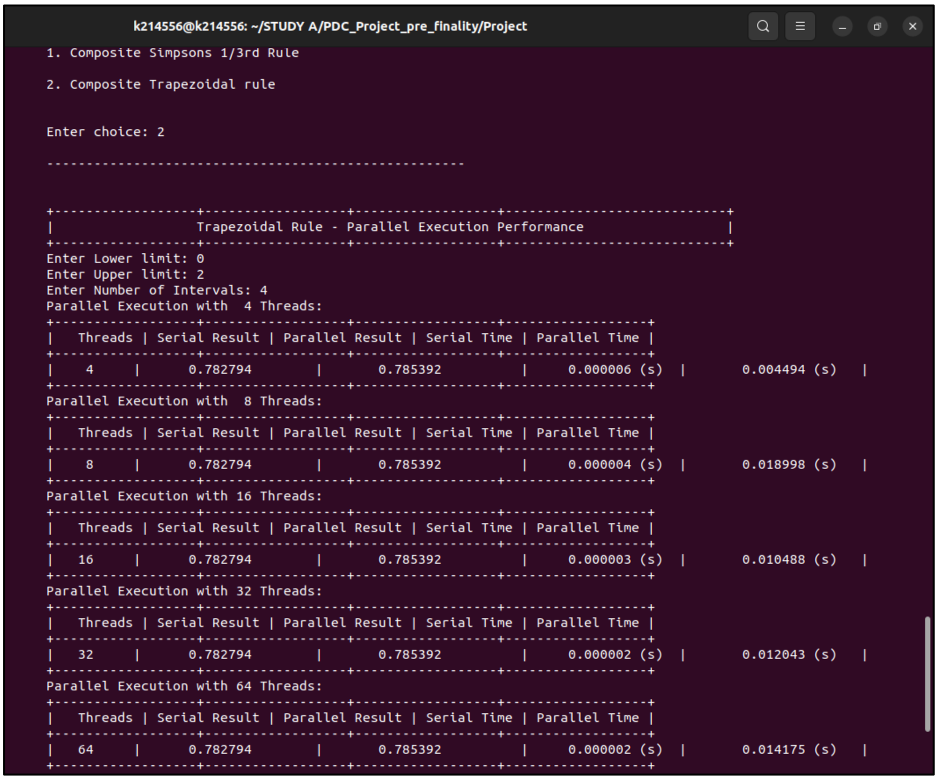
Simpson's method is a numerical integration technique that approximates the definite integral of a function by using quadratic polynomials. It divides the interval into smaller segments and employs weighted averages of function values at these points. In OpenMP, this method can benefit from parallelism by assigning segments to different threads for simultaneous computation, accelerating the integration process.



```
k214556@k214556: ~/STUDY A/PDC_Project_pre_finality/Project
1. Composite Simpsons 1/3rd Rule

2. Composite Trapezoidal rule


Enter choice: 1

--------------------------------------------------


+----------------+----------------+----------------+------------------------+
|                Simpson's 1/3  Rule - Parallel Execution Performance       |
+----------------+----------------+----------------+------------------------+
Enter Lower limit: 0
Enter Upper limit: 2
Enter Number of Intervals: 4
Parallel Execution with  4 Threads:
+----------------+----------------+----------------+----------------+
|   Threads | Serial Result | Parallel Result | Serial Time | Parallel Time |
+----------------+----------------+----------------+----------------+
|    4     |      0.785392        |       0.785392          |     0.000050 (s)  |       0.109142 (s)   |
+----------------+----------------+----------------+----------------+
Parallel Execution with  8 Threads:
+----------------+----------------+----------------+----------------+
|   Threads | Serial Result | Parallel Result | Serial Time | Parallel Time |
+----------------+----------------+----------------+----------------+
|    8     |      0.785392        |       0.785392          |     0.000007 (s)  |       0.027110 (s)   |
+----------------+----------------+----------------+----------------+
Parallel Execution with 16 Threads:
+----------------+----------------+----------------+----------------+
|   Threads | Serial Result | Parallel Result | Serial Time | Parallel Time |
+----------------+----------------+----------------+----------------+
|   16     |      0.785392        |       0.785392          |     0.000006 (s)  |       0.004154 (s)   |
+----------------+----------------+----------------+----------------+
Parallel Execution with 32 Threads:
+----------------+----------------+----------------+----------------+
|   Threads | Serial Result | Parallel Result | Serial Time | Parallel Time |
+----------------+----------------+----------------+----------------+
|   32     |      0.785392        |       0.785392          |     0.000005 (s)  |       0.025766 (s)   |
+----------------+----------------+----------------+----------------+
Parallel Execution with 64 Threads:
+----------------+----------------+----------------+----------------+
|   Threads | Serial Result | Parallel Result | Serial Time | Parallel Time |
+----------------+----------------+----------------+----------------+
|   64     |      0.785392        |       0.785392          |     0.000005 (s)  |       0.011748 (s)   |
+----------------+----------------+----------------+----------------+
```

Trapezoidal Method

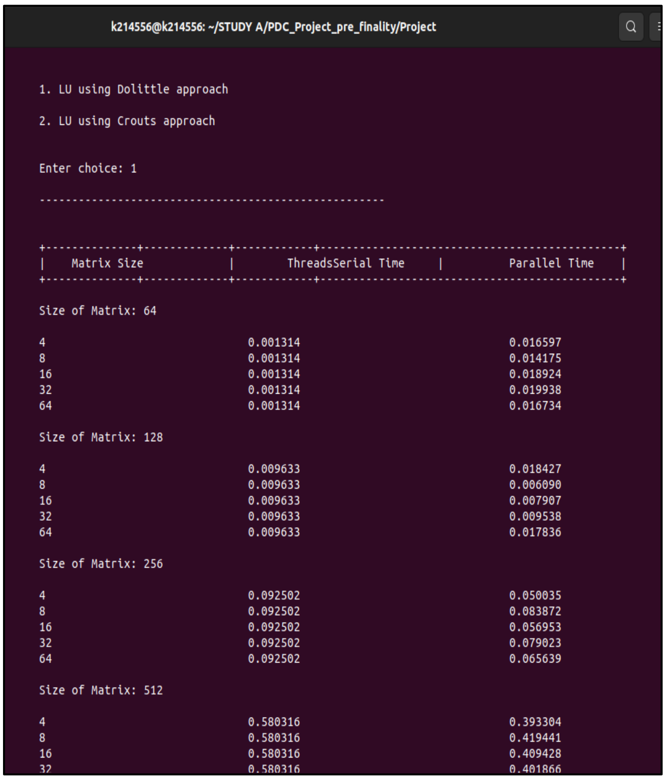
The trapezoidal method estimates the definite integral by approximating the area under the curve using trapezoids. It divides the interval into smaller subintervals and computes the area for each trapezoid. With OpenMP, this method can be parallelized by assigning segments to multiple threads, allowing for concurrent calculation of trapezoidal areas, thereby speeding up the integration process.

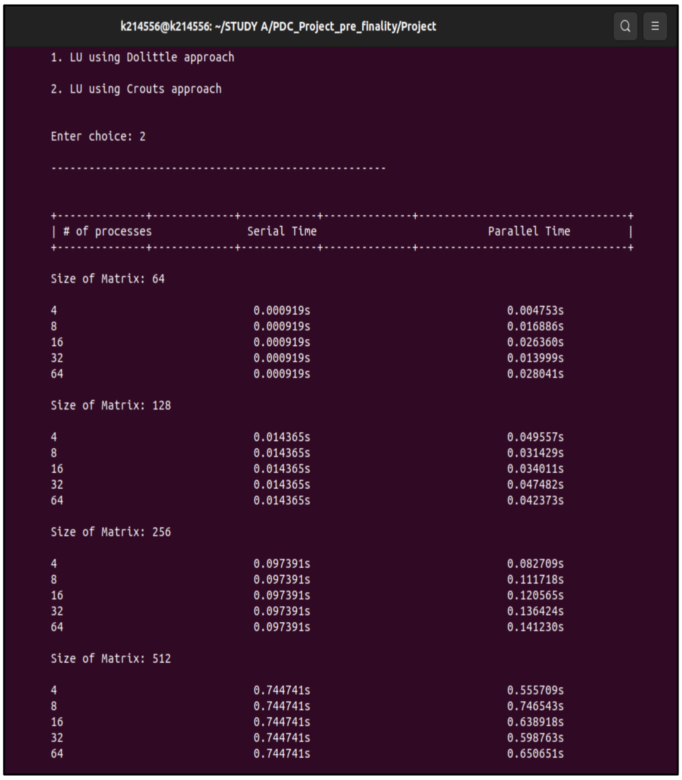


Solving Linear Systems

LU Decomposition (Doolittle and Crout)

LU decomposition is a method used to solve linear systems of equations by decomposing the coefficient matrix into lower and upper triangular matrices. Doolittle and Crout are two approaches to achieve LU decomposition. In a parallel setting with OpenMP, decomposition involves breaking down the matrix operations into parallel tasks, distributing the computation of matrix factors among threads for efficient computation.

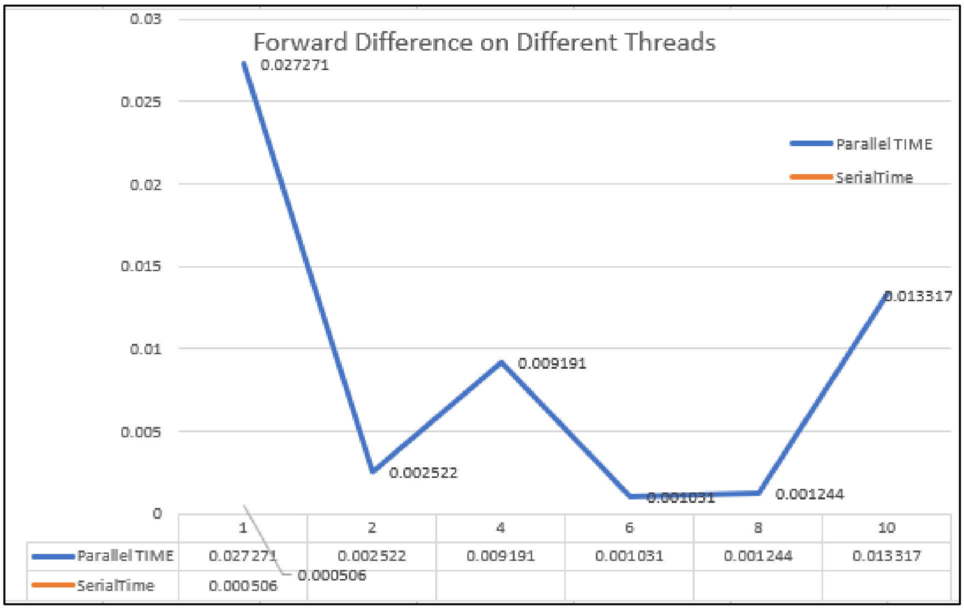




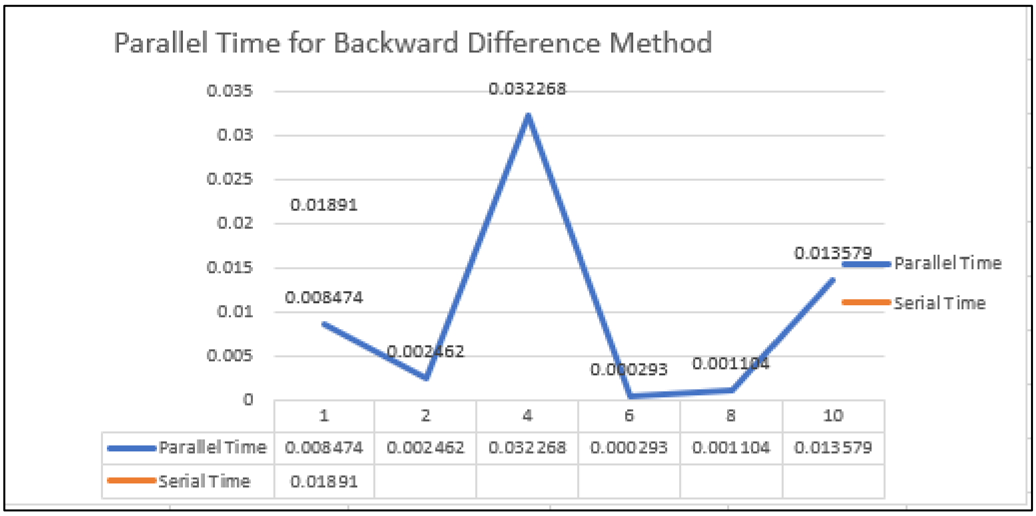
Each of these methods demonstrates varying computational complexities and advantages when parallelized using OpenMP in C on the Ubuntu operating system. Parallelization aims to optimize execution time by leveraging multiple cores or threads for simultaneous computations, thereby enhancing the overall performance of these numerical algorithm

Serial and Parallel Execution Visualization through Graphs:

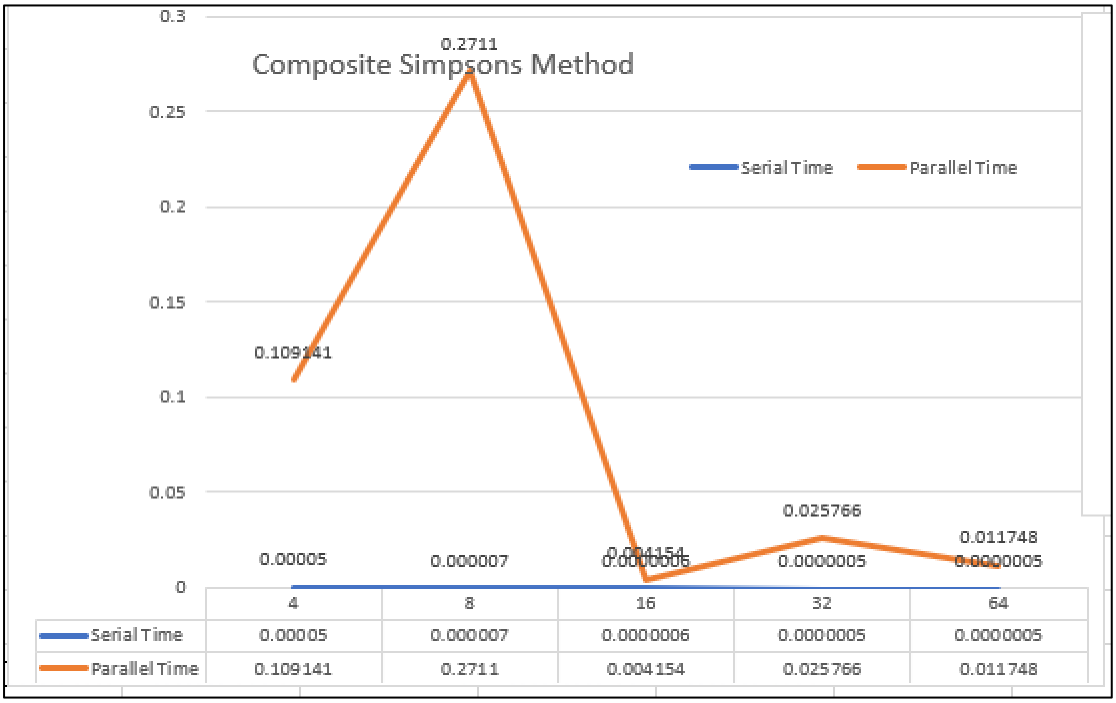
Forward Difference:



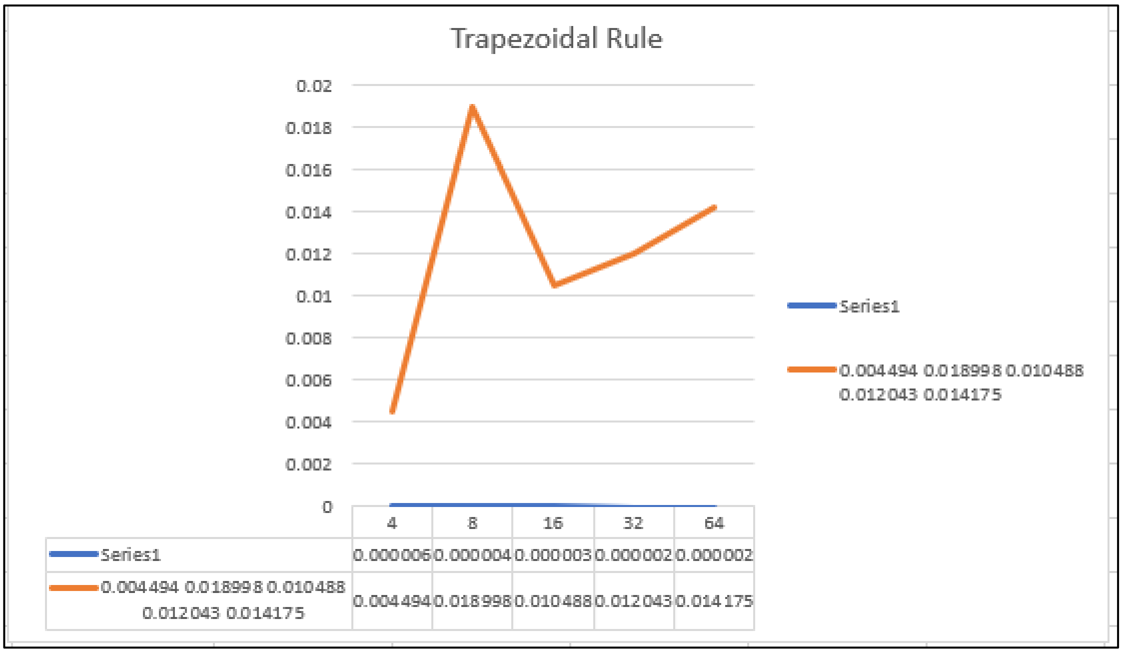
Backward Difference:



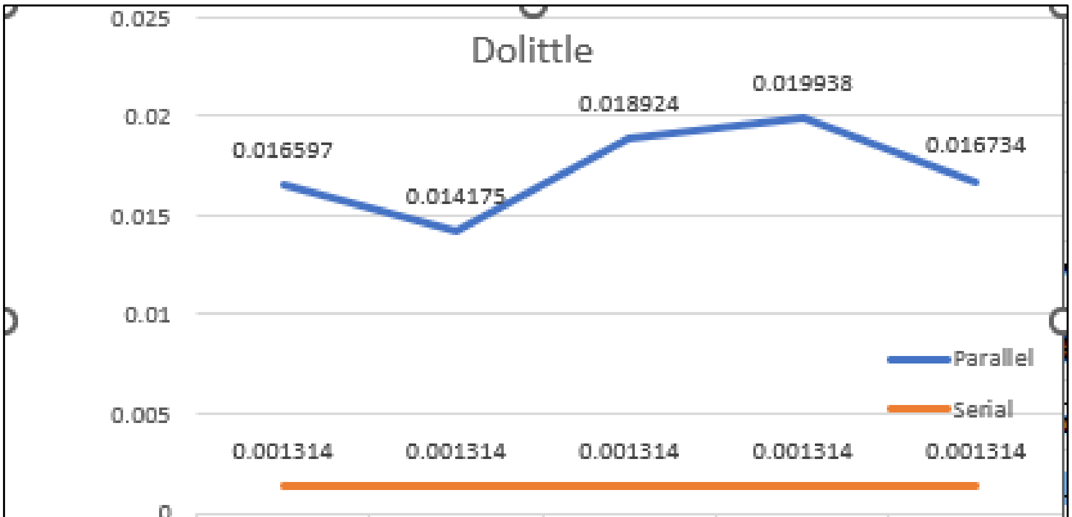
Simpsons Method:



Trapezoidal Method:

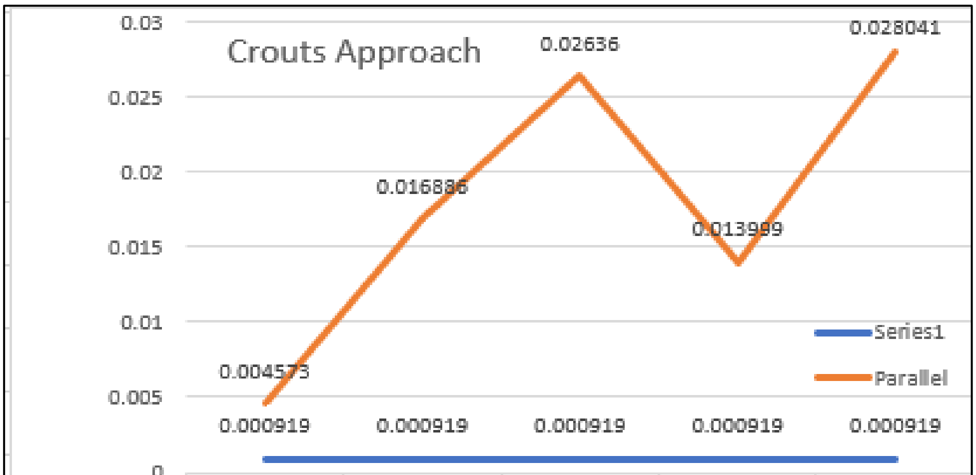


Dolittle Method:



	4	8	16	32	64
Parallel	0.016597	0.014175	0.018924	0.019938	0.016734
Serial	0.001314	0.001314	0.001314	0.001314	0.001314

Crout Method:



	4	8	16	32	64
Series1	0.000919	0.000919	0.000919	0.000919	0.000919
Parallel	0.004573	0.016886	0.02636	0.013999	0.028041

Specifications of the System Performed:

Core	I7
Generation	11TH Generation
Number of Cores	4
Threads	8
Speed	3.8ghz

Conclusion:

In summary, the project's exploration of serial and parallel methods using OpenMP has not only demonstrated the advantages of parallelization in numerical computing but has also provided valuable insights into optimizing computational tasks for improved efficiency and scalability. The findings underscore the practical relevance of leveraging parallel processing techniques to address increasingly complex computational challenges in various domains.