***Analysis & Design of Algorithms***

**Assignment**: - Serial vs Parallel Implementation using

OpenMP

**Topic**: - Polynomial Evaluation – Evaluate a polynomial of

degree N at multiple points using Horner’s rule.

**To: -** Ms. Sahana

**By**: - Adarsh Devadiga (4MW23CS002),

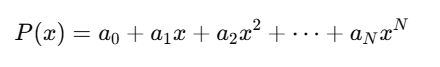
Adarsh Acharya (4MW23CS003),

Ashish Prasad (4MW23CS018),

Kunal K Prabhu(4MW233CS060)

***REPORT:***

Polynomial evaluation is the process of computing the value of a polynomial function at given input points. A polynomial of degree N is typically expressed as:



Directly computing this form involves a large number of multiplications, especially for higher degrees. To optimize this, Horner’s Rule is used. It rewrites the polynomial in a nested form that reduces the total number of operations:



This approach requires only N multiplications and N additions, making it very efficient for evaluating polynomials.

**Code Working:**

The program begins by prompting the user to enter the number of input points *(num\_points)* at which the polynomial should be evaluated. It then dynamically allocates memory for the polynomial coefficients, input x-values, and arrays to store the results of both serial and parallel evaluations. The polynomial coefficients and x-values are initialized with random numbers to simulate generic test cases.

The core computation is performed using **Horner’s Rule**, implemented in a function that efficiently evaluates the polynomial at a single point. First, the program executes a **serial loop**, where the polynomial is evaluated at each x-value one by one. The time taken for this operation is recorded using the *clock()* function. Next, the program performs the same evaluation in **parallel** using OpenMP, where multiple threads compute the values simultaneously, again measuring the execution time.

After both computations are completed, the results of the serial and parallel versions are compared element-by-element to verify correctness. If the results match within a small tolerance, the program prints a confirmation message. Finally, the execution times of both versions are displayed, allowing for performance comparison between serial and parallel execution.   
This setup provides a clear understanding of how parallelization can improve computational efficiency, especially for large input sizes.

1. ***Time complexity analysis:***

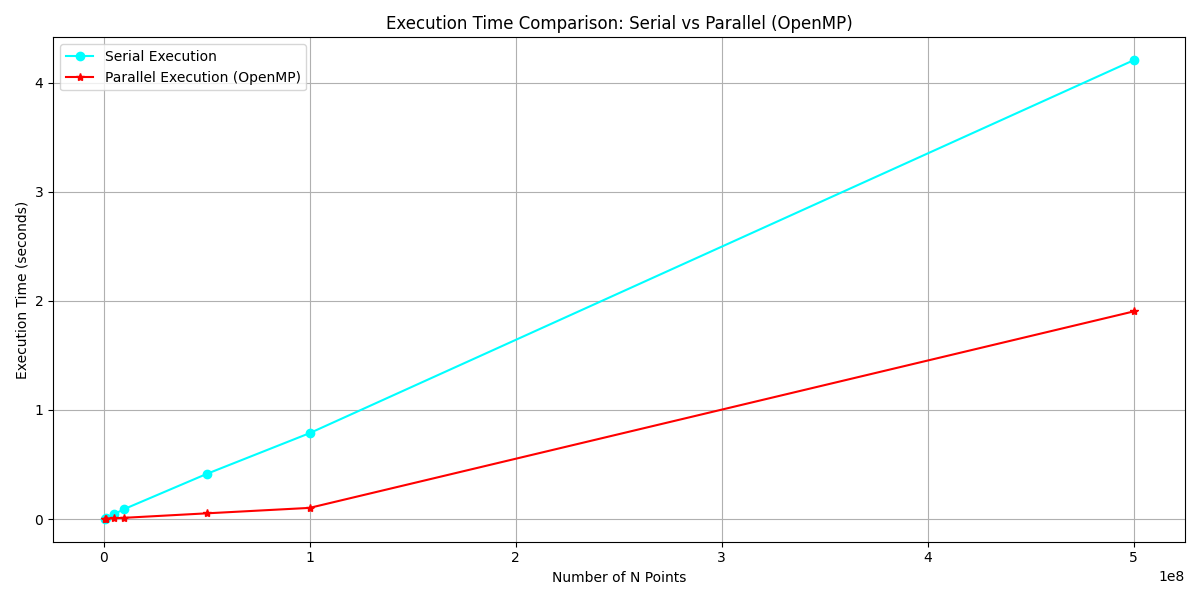
Horner's Rule Complexity:

* For evaluating a polynomial of degree *n*, Horner’s rule performs *n* multiplications and *n* additions per evaluation point.
* Thus, Time Complexity per point: *O(n)*
* If we evaluate the polynomial at *m* points:
  + Serial complexity: *O(m\* n)*
  + Parallel complexity (ideal): *O****(****(m\* n)/p****)*** where *p* is the number of threads

Since the polynomial degree *n* is constant (5), the overall complexity reduces to:

* Serial: *O(m)*
* Parallel: *O(m/p)* (ideal case)

1. ***Execution time comparison:***



1. ***Observations on speedup and scalability:***

* Speedup Trend:

Speedup is calculated as:

Speedup=Time(*serial)*/Time*(parallel)*

* Based on the data:

|  |  |  |  |
| --- | --- | --- | --- |
| Number of points | Serial Time (s) | Parallel Time (s) | Speedup (x) |
| 500,000 | 0.004000 | 0.002000 | 2.00 |
| 1,000,000 | 0.008000 | 0.003000 | 2.67 |
| 5,000,000 | 0.043000 | 0.008000 | 5.38 |
| 10,000,000 | 0.091000 | 0.011000 | 8.27 |

1. ***Scalability:***

* **Scalability** refers to how well the parallel implementation handles increasing workloads.
* The parallel version scales efficiently as input size grows, particularly beyond 1 million points.
* The use of OpenMP enables near-linear scaling up to a point, limited primarily by hardware constraints (number of CPU cores, memory bandwidth).

1. ***Diminishing Returns:***

* As input size increases, speedup starts to **plateau**—approaching but not surpassing a ceiling.
* This is due to:
  + Thread management overhead.
  + Memory access contention.
  + Non-parallelizable parts of the code (e.g., input, memory allocation).

--------------------------------------------X--------------------------------------