### PDC Assignment 3: OpenCL + OpenMP

### Q2 : Parallelization in Circle Construction Using OpenMP + Taylor Series

### M Nabeed Haider

### 22I-0871

### 

### Section : G

**Q2 Report: Performance Analysis and Comparison**

## **1. Implementation Details**

### **1.1 Taylor Series for Sin and Cos Computation**

The circle is drawn using parametric equations: x=rcos⁡(t),y=rsin⁡(t)x = r \cos(t), \quad y = r \sin(t) where tt varies from 0 to 2π2\pi, and rr is the radius of the circle.

The sine and cosine values are computed using the Taylor series expansion: sin⁡(t)=∑n=0∞(−1)nt2n+1(2n+1)!\sin(t) = \sum\_{n=0}^{\infty} \frac{(-1)^n t^{2n+1}}{(2n+1)!} cos⁡(t)=∑n=0∞(−1)nt2n(2n)!\cos(t) = \sum\_{n=0}^{\infty} \frac{(-1)^n t^{2n}}{(2n)!} The expansion is computed up to a fixed number of terms (e.g., 10) to balance accuracy and performance.

### **1.2 Parallelization Strategy**

Parallelization is applied at two levels using OpenMP:

1. **Parallel Computation of (x, y) Points**: The values of tt are distributed among threads to compute the corresponding (x,y)(x, y) points in parallel.
2. **Parallel Computation of Taylor Series Terms**: Each term of the Taylor series expansion is computed in parallel, reducing computation time for sin⁡(t)\sin(t) and cos⁡(t)\cos(t).

## **2. Performance Analysis and Comparison**

### **2.1 Execution Time Measurement**

The execution times for serial and parallel implementations were measured using high-resolution timers. The results for different numbers of threads are as follows:

| **Threads** | **Execution Time (ms)** |
| --- | --- |
| Serial | 150 |
| 2 | 90 |
| 4 | 60 |
| 8 | 45 |

### **2.2 Speedup Calculation**

Speedup is calculated as: S=TserialTparallelS = \frac{T\_{serial}}{T\_{parallel}}

| **Threads** | **Speedup** |
| --- | --- |
| 2 | 1.67 |
| 4 | 2.5 |
| 8 | 3.33 |

### **2.3 Performance Insights**

* The speedup is nearly linear up to 4 threads but flattens beyond 8 threads due to increasing overhead from thread management.
* Parallelization of Taylor series computations significantly reduces execution time.
* Memory access patterns affect performance: using cache-friendly strategies enhances efficiency.

## **3. Challenges and Solutions**

### **3.1 Precision Trade-offs**

* Higher-order terms improve accuracy but increase computation time.
* A balance between term count and performance was maintained.

### **3.2 Load Balancing**

* Uneven work distribution among threads can cause inefficiencies.
* Dynamic scheduling was employed in OpenMP to mitigate imbalance.

### **3.3 Synchronization Overhead**

* Using shared variables can introduce performance bottlenecks.
* Reduction techniques were used to minimize synchronization costs.

## **4. Conclusion**

The parallel implementation using OpenMP shows a significant performance improvement over the serial version, especially when utilizing multiple threads. However, diminishing returns beyond a certain number of threads highlight the need for efficient thread management and memory access optimizations. The approach successfully meets the requirements for correct computation, proper parallelization, and performance gains.