Assignment #2

Tags	Assignment / Exam, Parallel & Dist. Systems

Information

Name: Vladimir W. Schmadlak

Student ID: st20283948

Programme: BSc (Hons) Software Engineering

Github: https://github.com/VladimorCodebreaker/Parallel-Systems-Assignment2.git

Structure

Overview

The program is designed to simulate a parallel painting task where a team of workers paints a set of circles. Developed in C#, the application leverages multi-threading and concurrency control to efficiently handle parallel processing.

Main Components

`Program Class`: Serves as the entry point. Manages user input and coordinates the execution of performance analysis.

`Circle Class`: Represents the circles to be painted.

`Worker Class`: Represents workers who paint circles. Each worker maintains its count of painted circles and checks shared data to avoid duplicate painting.

`PaintCoordinator Class`: Orchestrates the distribution of painting tasks among workers in parallel.

`Visualization Class`: Provides real-time visual feedback on the painting task's progress.

`PerformanceAnalyzer Class`: Measures and reports the time taken for the painting task to be completed by different numbers of workers.

Evaluation

Is this problem able to be parallelized?

The task of painting circles is inherently parallelisable as each circle can be painted independently without affecting the others. This independence allows for efficient distribution of tasks across multiple workers.

How would the problem be partitioned?

The problem is partitioned by dividing the set of circles among available workers. Each worker is responsible for painting a subset of the total circles, minimizing interdependencies.

Are communications needed?

Communication among workers is required to ensure a circle is not painted more than once. This is achieved using a shared concurrent dictionary that tracks the painted status of each circle.

Are there any data dependencies?

Data dependency is minimal and is primarily related to the status of whether a circle has been painted. The shared concurrent dictionary manages this dependency effectively.

Are there synchronization needs?

Synchronization is crucial to prevent concurrent modifications of shared data (painted circles). The use of thread-safe collections and synchronization mechanisms like `lock` ensures data integrity.

Will load balancing be a concern?

Load balancing is a consideration as some workers might finish their tasks earlier than others. The program manages this by dynamically assigning circles to workers until all are painted.

Test Results

Testing Scenarios

The application's performance was evaluated under three distinct scenarios, each varying in the number of worker threads employed for the painting task. The scenarios included tests with 5, 20, and 100 workers, respectively.

```
Threads: 5, Time: 4316 ms
Threads: 20, Time: 1706 ms
```

```
Threads: 100, Time: 1591 ms
```

Performance Metrics / Observations

`5 Workers`: When the task was executed with 5 workers, the total time taken to paint all the circles was **4316 milliseconds (ms)**. This scenario represents a baseline for performance measurement.

`20 Workers`: Increasing the number of workers to 20 significantly improved the performance. The task was completed in **1706 ms**, showcasing a noticeable reduction in time compared to the 5-worker scenario.

`100 Workers`: The most significant improvement was observed when the worker count was raised to 100. The painting task was completed in just **1591 ms**, slightly faster than the 20-worker scenario, indicating diminishing returns on adding more workers beyond a certain point.