Design & Analysis of Algorithms Mini Project

Title:
Implement Merge Sort and Multithreaded Merge Sort. Compare Time Required by Both the Algorithms. Also, Analyze the Performance of Each Algorithm for the Best Case and the Worst Case.

Group Members:

- Tanay Jadhav(29)
- Ritesh Shevare(30)
- Om Taskar(31)
- Pratik Kotkar(32)

Problem Statement:
Write a program to implement the merge sort algorithm. Additionally, implement a multithreaded version of merge sort. Compare and analyze the time complexity and performance of both algorithms in the best-case and worst-case scenarios.

Objectives:

- To implement a single-threaded merge sort algorithm.
- To develop a multithreaded merge sort algorithm where different parts of the array are sorted concurrently.
- To analyze and compare the execution time and efficiency of the single-threaded and multithreaded merge sort implementations.
- To evaluate the impact of input size and structure (best and worst cases) on the performance of both implementations.
- To assess the computational overhead introduced by thread management and synchronization in multithreaded merge sort.

Introduction:
Merge sort is a classic divide-and-conquer sorting algorithm with a time complexity of O(n log n) for both the best-case and worst-case scenarios. It is widely used for its efficiency with large datasets, especially when stable sorting is required. However, the inherent sequential nature of merge sort limits its speed-up in single-threaded environments as the array is divided and merged in a strictly linear sequence. By leveraging multithreading, the sorting process can be parallelized, allowing different parts of the array to be sorted

simultaneously, which can lead to significant improvements in execution time, particularly on modern multi-core processors.

In this project, we implement both the traditional single-threaded merge sort and a multithreaded version where multiple threads are used to sort different segments of the array concurrently. The project aims to analyze and compare the time required for both algorithms under different input conditions (best and worst cases), highlighting the advantages and challenges of using multithreading for this problem.

Algorithm/Program:

1. **Merge Sort (Single-threaded)**

```
void merge(vector<int>& arr, int left, int mid, int right) {
   // Merging logic for single-threaded merge sort
}

void mergeSort(vector<int>& arr, int left, int right) {
   if (left < right) {
      int mid = left + (right - left) / 2;
      mergeSort(arr, left, mid);
      mergeSort(arr, mid + 1, right);
      merge(arr, left, mid, right);
    }
 }
```

2. Multithreaded Merge Sort

```
void mergeSortMultiThreaded(vector<int>& arr, int left, int right) {
   if (right - left <= THRESHOLD) {
      // Use single-threaded merge sort when the size is small enough
      mergeSort(arr, left, right);
   } else {
      int mid = left + (right - left) / 2;
      std::thread leftThread(mergeSortMultiThreaded, std::ref(arr), left, mid);
      std::thread rightThread(mergeSortMultiThreaded, std::ref(arr), mid + 1, right);
      leftThread.join();
      rightThread.join();
      merge(arr, left, mid, right);
   }
}
```

Analysis:

Aspect	Single-Threaded Merge Sort	Multithreaded Merge Sort
Worst-case Time Complexity	O(n log n)	O((n log n) / P)
Best-case Time Complexity	O(n log n)	O((n log n) / P)
Average-case Time Complexity	O(n log n)	O((n log n) / P)
Space Complexity	O(n)	O(n) + O(P log n) (for thread overhead)

- Where P is the number of threads.

Conclusion:
Both the single-threaded and multithreaded merge sort algorithms have a time complexity of O(n log n) for the best-case and worst-case scenarios. However, the multithreaded approach can provide a significant speedup in execution time, especially on large datasets, by utilizing multiple CPU cores. The degree of improvement depends on the number of available cores and the overhead of thread management. While multithreading improves performance in general, excessive thread creation can lead to diminishing returns due to synchronization overhead and resource contention.