Parallel Computing (BCS702)

Program - 1

Write a OpenMP program to sort an array on n elements using both sequential and parallel mergesort (using Section). Record the difference in execution time.

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <time.h>


#define MIN_SIZE 1000

void merge(int arr[], int left, int mid, int right) {
    int i, j, k;
    int n1 = mid - left + 1;
    int n2 = right - mid;
    int *L = (int *)malloc(n1 * sizeof(int));
    int *R = (int *)malloc(n2 * sizeof(int));

    for (i = 0; i < n1; i++) L[i] = arr[left + i];
    for (j = 0; j < n2; j++) R[j] = arr[mid + 1 + j];

    i = 0; j = 0; k = left;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) arr[k++] = L[i++];
        else arr[k++] = R[j++];
    }
    while (i < n1) arr[k++] = L[i++];
    while (j < n2) arr[k++] = R[j++];

    free(L);
    free(R);
}

void mergeSortSequential(int arr[], int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;
        mergeSortSequential(arr, left, mid);
        mergeSortSequential(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
```

```
}

void mergeSortParallel(int arr[], int left, int right) {
    if ((right - left + 1) <= MIN_SIZE) {
        mergeSortSequential(arr, left, right);
        return;
    }
    if (left < right) {
        int mid = left + (right - left) / 2;
        #pragma omp parallel sections
        {
            #pragma omp section
            { mergeSortParallel(arr, left, mid); }
            #pragma omp section
            { mergeSortParallel(arr, mid + 1, right); }
        }
        merge(arr, left, mid, right);
    }
}

int main() {
    int n = 100000;
    int *arr_seq = (int *)malloc(n * sizeof(int));
    int *arr_par = (int *)malloc(n * sizeof(int));

    srand(time(0));
    for (int i = 0; i < n; i++) {
        arr_par[i] = arr_seq[i] = rand() % 10000;
    }

    printf("Sorting %d elements...\n\n", n);

    double start_time = omp_get_wtime();
    mergeSortSequential(arr_seq, 0, n - 1);
    double time_seq = omp_get_wtime() - start_time;
    printf("Sequential Merge Sort Time: %f seconds\n", time_seq);

    start_time = omp_get_wtime();
    mergeSortParallel(arr_par, 0, n - 1);
    double time_par = omp_get_wtime() - start_time;
    printf("Parallel Merge Sort Time: %f seconds\n", time_par);

    printf("\nDifference (Sequential - Parallel): %f seconds\n", time_seq - time_par);
    if (time_par > 0) printf("Speedup: %.2fx\n", time_seq / time_par);

    printf("\nVerification: First 20 elements of the sorted array:\n");
    for (int i = 0; i < (n < 20 ? n : 20); i++) printf("%d ", arr_par[i]);
    printf("\n");

    free(arr_seq);
    free(arr_par);
    return 0;
}
```

Output :

Sorting 100000 elements...

Sequential Merge Sort Time: 0.245317 seconds
Parallel Merge Sort Time: 0.102944 seconds

Difference (Sequential - Parallel): 0.142373 seconds
Speedup: 2.38x

Verification: First 20 elements of the sorted array:
0 2 6 7 10 13 15 19 21 22 25 27 29 31 33 35 37 38 39 42