Write a program to implement Parallel Merge sort using OpenMP. Use existing algorithms and measure the performance of sequential and parallel algorithms.

```
#include <iostream>
#include <vector>
#include <omp.h>
#include <chrono>

void merge(std::vector<int>& arr, int left, int mid, int right) {
   int n1 = mid - left + 1;
   int n2 = right - mid;

   std::vector<int> leftArr(n1);
   std::vector<int> rightArr(n2);

   for (int i = 0; i < n1; ++i) {
      leftArr[i] = arr[left + i];
   }
   for (int j = 0; j < n2; ++j) {
      rightArr[j] = arr[mid + 1 + j];
   }

   int i = 0;
   int j = 0;
   int k = left;

   while (i < n1 && j < n2) {
      if (leftArr[i] <= rightArr[j]) {
         arr[k] = leftArr[i];
         ++i;
      } else {
         arr[k] = rightArr[j];
         ++j;
      }
      ++k;
   }

   while (i < n1) {
      arr[k] = leftArr[i];
      ++i;
      ++k;
   }

   while (j < n2) {
```

```
        arr[k] = rightArr[j];
        ++j;
        ++k;
    }
}

void sequentialMergeSort(std::vector<int>& arr, int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;

        sequentialMergeSort(arr, left, mid);
        sequentialMergeSort(arr, mid + 1, right);

        merge(arr, left, mid, right);
    }
}

void parallelMergeSort(std::vector<int>& arr, int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;

        #pragma omp parallel sections
        {
            #pragma omp section
            {
                parallelMergeSort(arr, left, mid);
            }
            #pragma omp section
            {
                parallelMergeSort(arr, mid + 1, right);
            }
        }

        merge(arr, left, mid, right);
    }
}

int main() {
    std::vector<int> arr = {9, 4, 2, 7, 5, 1, 8, 3, 6};

    std::cout << "Sequential Merge Sort:" << std::endl;
    std::vector<int> arrSeq = arr;
    auto startSeq = std::chrono::steady_clock::now();
    sequentialMergeSort(arrSeq, 0, arrSeq.size() - 1);
    auto endSeq = std::chrono::steady_clock::now();
    std::chrono::duration<double> durationSeq = endSeq - startSeq;
```

```
    for (int num : arrSeq) {
        std::cout << num << " ";
    }

    std::cout << "\n\nParallel Merge Sort:" << std::endl;
    std::vector<int> arrPar = arr;
    auto startPar = std::chrono::steady_clock::now();
    parallelMergeSort(arrPar, 0, arrPar.size() - 1);
    auto endPar = std::chrono::steady_clock::now();
    std::chrono::duration<double> durationPar = endPar - startPar;

    for (int num : arrPar) {
        std::cout << num << " ";
    }

    std::cout << "\n\nSequential Merge Sort Duration: " << durationSeq.count() << " seconds";
    std::cout << "\nParallel Merge Sort Duration: " << durationPar.count() << " seconds";

    return 0;
}
```