Assignment – 3

Code:

```
#include <bits/stdc++.h>
#include <chrono>
#include <omp.h>

using namespace std;

// Sequential function to find minimum value in a vector
int sequential_reduce_min(vector<int>& data) {
   return *min_element(data.begin(), data.end());
}

// Sequential function to find maximum value in a vector
int sequential_reduce_max(vector<int>& data) {
   return *max_element(data.begin(), data.end());
}

// Sequential function to calculate sum of all elements in a vector
int sequential_reduce_sum(vector<int>& data) {
   return accumulate(data.begin(), data.end(), 0);
}

// Sequential function to calculate average of all elements in a vector
double sequential_reduce_average(vector<int>& data) {
   int sum = sequential_reduce_sum(data);
   return static_cast<double>(sum) / data.size();
}

// Parallel function to find minimum value in a vector
int parallel_reduce_min(vector<int>& data) {
   int num_threads = omp_get_max_threads();
   int chunk_size = data.size() / num_threads;
   int result = INT_MAX;

   #pragma omp parallel for reduction(min:result)
   for (int i = 0; i < data.size(); i += chunk_size) {
      int local_result = *min_element(data.begin() + i, data.begin() + min(i + chunk_size,
static_cast<int>(data.size())));
      #pragma omp critical
```

```
        result = min(result, local_result);
    }

    return result;
}

// Parallel function to find maximum value in a vector
int parallel_reduce_max(vector<int>& data) {
    int num_threads = omp_get_max_threads();
    int chunk_size = data.size() / num_threads;
    int result = INT_MIN;

    #pragma omp parallel for reduction(max:result)
    for (int i = 0; i < data.size(); i += chunk_size) {
        int local_result = *max_element(data.begin() + i, data.begin() + min(i + chunk_size,
static_cast<int>(data.size())));
        #pragma omp critical
        result = max(result, local_result);
    }

    return result;
}

// Parallel function to calculate sum of all elements in a vector
int parallel_reduce_sum(vector<int>& data) {
    int num_threads = omp_get_max_threads();
    int chunk_size = data.size() / num_threads;
    int result = 0;

    #pragma omp parallel for reduction(+:result)
    for (int i = 0; i < data.size(); i += chunk_size) {
        int local_result = accumulate(data.begin() + i, data.begin() + min(i + chunk_size,
static_cast<int>(data.size())), 0);
        #pragma omp critical
        result += local_result;
    }

    return result;
}
```

```
// Parallel function to calculate average of all elements in a vector
double parallel_reduce_average(vector<int>& data) {
   int num_threads = omp_get_max_threads();
   int chunk_size = data.size() / num_threads;
   double sum = 0;

   #pragma omp parallel for reduction(+:sum)
   for (int i = 0; i < data.size(); i += chunk_size) {
      int local_sum = accumulate(data.begin() + i, data.begin() + min(i + chunk_size,
static_cast<int>(data.size())), 0);
      #pragma omp critical
      sum += local_sum;
   }

   return sum / data.size();
}

//Main function
signed main() {
   int array_size;
   int min_range, max_range;

   cout << "Enter the size of the array: ";
   cin >> array_size;
   cout << "Enter the minimum range for random numbers: ";
   cin >> min_range;
   cout << "Enter the maximum range for random numbers: ";
   cin >> max_range;

   vector<int> data(array_size);

   srand(time(0));
   //taking random values between the given range by user
   for (int i = 0; i < array_size; ++i) {
      data[i] = rand() % (max_range - min_range + 1) + min_range;
   }

   bool flag = true;
```

```
 while (flag) {
      int ch;
      cout << "\n<-----------MENU----------->" << endl;
      cout << "1. For Maximum value in array" << endl;
      cout << "2. For Minimum value in array" << endl;
      cout << "3. For Sum of all values in array" << endl;
      cout << "4. For Average of all values in array" << endl;
      cout << "5. For Exit" << endl;

      cout << "Enter your choice: ";
      cin >> ch;

      switch (ch) {
         case 1: {
            auto start = chrono::steady_clock::now();
            int max_val_parallel = parallel_reduce_max(data);
            auto end = chrono::steady_clock::now();
            cout << "Parallel Maximum value in array: " << max_val_parallel << endl;
            cout << "Time taken (Parallel): " << chrono::duration_cast<chrono::microseconds>(end
- start).count() << " microseconds" << endl;

            start = chrono::steady_clock::now();
            int max_val_sequential = sequential_reduce_max(data);
            end = chrono::steady_clock::now();
            cout << "Sequential Maximum value in array: " << max_val_sequential << endl;
            cout << "Time taken (Sequential): " <<
chrono::duration_cast<chrono::microseconds>(end - start).count() << " microseconds" << endl;
            break;
         }

         case 2: {
            auto start = chrono::steady_clock::now();
            int min_val_parallel = parallel_reduce_min(data);
            auto end = chrono::steady_clock::now();
            cout << "Parallel Minimum value in array: " << min_val_parallel << endl;
            cout << "Time taken (Parallel): " << chrono::duration_cast<chrono::microseconds>(end
- start).count() << " microseconds" << endl;

            start = chrono::steady_clock::now();
            int min_val_sequential = sequential_reduce_min(data);
            end = chrono::steady_clock::now();
```

```
            cout << "Sequential Minimum value in array: " << min_val_sequential << endl;
            cout << "Time taken (Sequential): " <<
chrono::duration_cast<chrono::microseconds>(end - start).count() << " microseconds" << endl;
            break;
        }

        case 3: {
            auto start = chrono::steady_clock::now();
            int sum_val_parallel = parallel_reduce_sum(data);
            auto end = chrono::steady_clock::now();
            cout << "Parallel Sum of all values in array: " << sum_val_parallel << endl;
            cout << "Time taken (Parallel): " << chrono::duration_cast<chrono::microseconds>(end
- start).count() << " microseconds" << endl;

            start = chrono::steady_clock::now();
            int sum_val_sequential = sequential_reduce_sum(data);
            end = chrono::steady_clock::now();
            cout << "Sequential Sum of all values in array: " << sum_val_sequential << endl;
            cout << "Time taken (Sequential): " <<
chrono::duration_cast<chrono::microseconds>(end - start).count() << " microseconds" << endl;
            break;
        }

        case 4: {
            auto start = chrono::steady_clock::now();
            double avg_val_parallel = parallel_reduce_average(data);
            auto end = chrono::steady_clock::now();
            cout << "Parallel Average of all values in array: " << avg_val_parallel << endl;
            cout << "Time taken (Parallel): " << chrono::duration_cast<chrono::microseconds>(end
- start).count() << " microseconds" << endl;

            start = chrono::steady_clock::now();
            double avg_val_sequential = sequential_reduce_average(data);
            end = chrono::steady_clock::now();
            cout << "Sequential Average of all values in array: " << avg_val_sequential << endl;
            cout << "Time taken (Sequential): " <<
chrono::duration_cast<chrono::microseconds>(end - start).count() << " microseconds" << endl;
            break;
        }

        case 5: {
```

```
            cout << "Thank You!!" << endl;
            flag = false;
            break;
        }

        default: {
            cout << "Invalid choice, Try Again";
            break;
        }
      }
   }
   return 0;
}
```

Output:

```
PS E:\AIT\4th Year Sem 2> g++ 7446_Assignment_3.cpp -lgomp -o as
PS E:\AIT\4th Year Sem 2> ./as
Enter the size of the array: 10
Enter the minimum range for random numbers: 1
Enter the maximum range for random numbers: 1000

<----------MENU---------->
1. For Maximum value in array
2. For Minimum value in array
3. For Sum of all values in array
4. For Average of all values in array
5. For Exit
Enter your choice: 1
Parallel Maximum value in array: 961
Time taken (Parallel): 0 microseconds
Sequential Maximum value in array: 961
Time taken (Sequential): 0 microseconds

<----------MENU---------->
1. For Maximum value in array
2. For Minimum value in array
3. For Sum of all values in array
4. For Average of all values in array
5. For Exit
Enter your choice: 2
Parallel Minimum value in array: 120
Time taken (Parallel): 0 microseconds
Sequential Minimum value in array: 120
Time taken (Sequential): 0 microseconds

<----------MENU---------->
1. For Maximum value in array
2. For Minimum value in array
3. For Sum of all values in array
4. For Average of all values in array
5. For Exit
Enter your choice: 3
Parallel Sum of all values in array: 5047
Time taken (Parallel): 0 microseconds
```

```
Sequential Sum of all values in array: 5047
Time taken (Sequential): 0 microseconds

<----------MENU---------->
1. For Maximum value in array
2. For Minimum value in array
3. For Sum of all values in array
4. For Average of all values in array
5. For Exit
Enter your choice: 4
Parallel Average of all values in array: 504.7
Time taken (Parallel): 0 microseconds
Sequential Average of all values in array: 504.7
Time taken (Sequential): 0 microseconds

<----------MENU---------->
1. For Maximum value in array
2. For Minimum value in array
3. For Sum of all values in array
4. For Average of all values in array
5. For Exit
Enter your choice: 5
Thank You!!
PS E:\AIT\4th Year Sem 2>
```