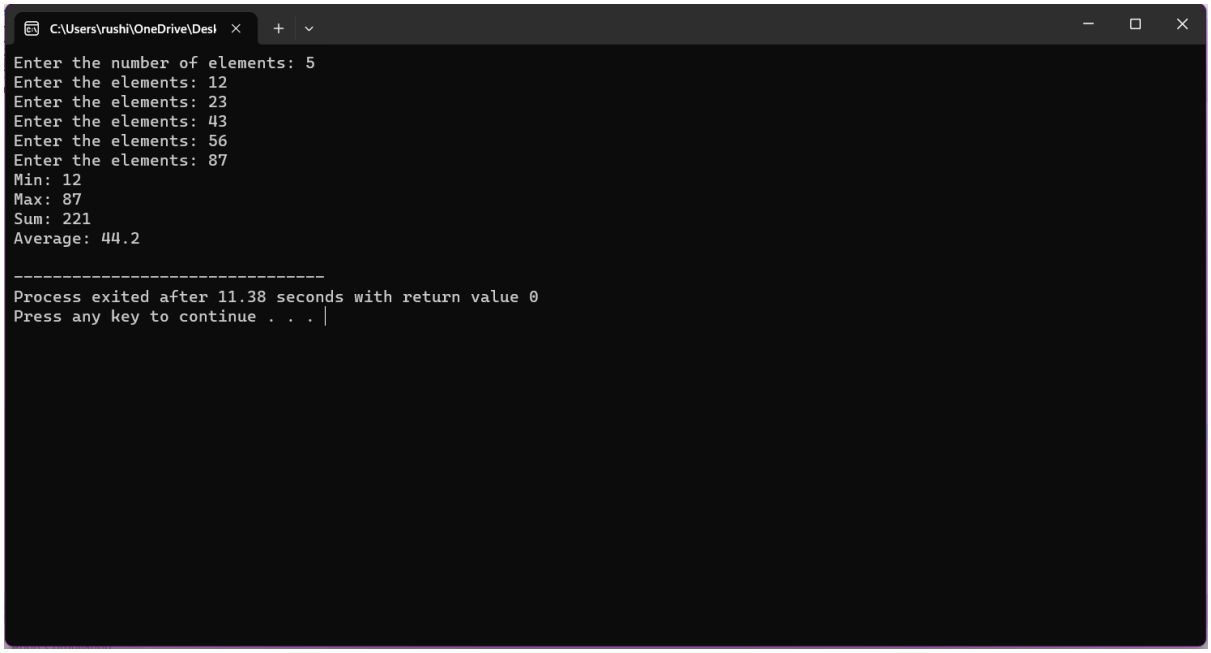
Title :- Implement Min, Max, Sum and Average operations using Parallel Reduction.

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

template <typename T>
T parallel_min(const std::vector<T>& data) {
    T result = std::numeric_limits<T>::max();
    #pragma omp parallel for reduction(min: result)
    for (size_t i = 0; i < data.size(); ++i) {
        result = std::min(result, data[i]);
    }
    return result;
}

template <typename T>
T parallel_max(const std::vector<T>& data) {
    T result = std::numeric_limits<T>::min();
    #pragma omp parallel for reduction(max: result)
    for (size_t i = 0; i < data.size(); ++i) {
        result = std::max(result, data[i]);
    }
    return result;
}

template <typename T>
T parallel_sum(const std::vector<T>& data) {
    T result = 0;
    #pragma omp parallel for reduction(+: result)
```

```
    for (size_t i = 0; i < data.size(); ++i) {
        result += data[i];
    }
    return result;
}

template <typename T>
double parallel_average(const std::vector<T>& data) {
    T sum = parallel_sum(data);
    return static_cast<double>(sum) / data.size();
}

int main() {
    std::cout << "Enter the number of elements: ";
    size_t num_elements;
    std::cin >> num_elements;
    std::vector<int> data(num_elements);

    for (size_t i = 0; i < num_elements; ++i) {
            std::cout << "Enter the elements: ";
        std::cin >> data[i];
    }
    std::cout << "Min: " << parallel_min(data) << std::endl;
    std::cout << "Max: " << parallel_max(data) << std::endl;
    std::cout << "Sum: " << parallel_sum(data) << std::endl;
    std::cout << "Average: " << parallel_average(data) << std::endl;

    return 0;
}
```

Output :-


C:\Users\rushi\OneDrive\Desk
Enter the number of elements: 5
Enter the elements: 12
Enter the elements: 23
Enter the elements: 43
Enter the elements: 56
Enter the elements: 87
Min: 12
Max: 87
Sum: 221
Average: 44.2

Process exited after 11.38 seconds with return value 0
Press any key to continue . . .