✅ Problem Statement (Simple Version)

Write a program in C++ using OpenMP to perform parallel reduction operations for:

- **Minimum**
- **Maximum**
- **Sum**
- **Average**

on a list (or array) of numbers. Use **OpenMP reduction clauses** to compute these efficiently using multiple threads.

🎯 Objectives

1. Input: A list of n numbers (array or vector).
2. Use **OpenMP reduction** to compute:
   - Minimum value
   - Maximum value
   - Sum
   - Average (computed as sum / n)
3. Compare **parallel execution** with a sequential version (optional for demo).

🧠 What is Parallel Reduction?

In OpenMP, **reduction** means:

Each thread does part of the work and then **combines the results**.

You use #pragma omp parallel for reduction(op : variable) to:

- Run the loop in **parallel**
- Combine each thread's result using the operation op (+, min, max, etc.)

How to Compile and Run (Linux or local compiler):

```
g++ -fopenmp reduction_ops.cpp -o reduction_ops
./reduction_ops
```

🟢 Sample Input:

```
Enter number of elements: 6
Enter elements: 4 8 1 10 5 2
```

✅ Output:

```
Minimum value: 1
Maximum value: 10
Sum: 30
Average: 5
```

What to Say in Viva:

"I used OpenMP's reduction clause to parallelize min, max, and sum operations across multiple threads. Each thread processes a part of the array, and OpenMP automatically combines the results. This improves performance for large arrays."

🧾 Code-by-Code Explanation

✅ parallelMin(vector<int> vec)

cpp

CopyEdit

```cpp
int min_val = vec[0];
#pragma omp parallel for
for (int i = 1; i < vec.size(); i++) {
    if (vec[i] < min_val) {
        min_val = vec[i];
    }
}
```

🔴 **Problem**: This is *not thread-safe* because min_val is a **shared variable**, and multiple threads could write to it at once.

✅ **Fix it using reduction**:

cpp

CopyEdit

```
#pragma omp parallel for reduction(min:min_val)
```

✅ **parallelMax(vector<int> vec)**

Same issue as min. Should be:

cpp

CopyEdit

```
#pragma omp parallel for reduction(max:max_val)
```

✅ **parallelSum(vector<int> vec)**

cpp

CopyEdit

```
int sum = 0;
#pragma omp parallel for
for (int i = 0; i < vec.size(); i++) {
    sum += vec[i];
}
```

🔴 Not thread-safe. Needs a **reduction** clause:

cpp

CopyEdit

```
#pragma omp parallel for reduction(+:sum)
```

✅ parallelAverage(vector<int> vec)

Uses parallelSum(vec) and divides by number of elements — this is correct.

🖥️ Main Function

cpp

CopyEdit

```
int n;
cin >> n;
vector<int> vec(n);
```

Takes input from user, stores in vector.

Then calls all four parallel functions and prints results.

✅ Output:

- Min
- Max
- Sum
- Average

🟢 Summary of Fixes

Function	Problem	Fix
parallelMin	Not thread-safe	Use reduction(min:min_val)
parallelMax	Not thread-safe	Use reduction(max:max_val)
parallelSum	Not thread-safe	Use reduction(+:sum)
parallelAverage	Relies on parallelSum	Fine if parallelSum is fixed

---

✅ Final Fixed Example (for parallelSum):

cpp

CopyEdit

```
int parallelSum(vector<int> vec) {
    int sum = 0;
    #pragma omp parallel for reduction(+:sum)
    for (int i = 0; i < vec.size(); i++) {
        sum += vec[i];
    }
    return sum;
}
```

🗣️ What to Say in Viva

“I implemented min, max, sum, and average using OpenMP. I used **parallel reduction** to safely perform operations like sum and max across multiple threads. Reduction helps avoid race conditions by combining results after each thread finishes its part.”