Explanation of Parallel Reduction Practical

1 Overview of the Practical

This practical involves computing the **minimum**, **maximum**, **sum**, and **average** of an array using **sequential** and **parallel reduction** with **OpenMP**. The objectives are:

- Implement sequential and parallel reduction algorithms.
- Measure performance on a user-defined array.
- Compute **speedup** (sequential time / parallel time) and **efficiency** (speedup / threads).
- Verify that parallel results match sequential results.
- Understand OpenMP directives for parallelization.

2 Code Explanation

The code implements **Sequential Reduction** and **Parallel Reduction** to process an array.

2.1 Sequential Reduction

Purpose: Computes the minimum, maximum, sum, and average of an array.

How it Works:

- Loops through the array once.
- Updates the minimum (smallest value), maximum (largest value), and sum (total).
- Calculates the average as sum divided by array size.
- Runs on a single thread.

Code:

```
int seqMin = INT_MAX;
int seqMax = INT_MIN;
long long seqSum = 0;
for (int i = 0; i < n; i++) {
    seqMin = min(seqMin, data[i]);
    seqMax = max(seqMax, data[i]);
    seqSum += data[i];
}
double seqAvg = static_cast<double>(seqSum) / n;
```

2.2 Parallel Reduction

Purpose: Splits the array across threads to compute local results, then combines them.

How it Works:

- Uses `#pragma omp parallel` to create threads.
- Each thread computes local minimum, maximum, and sum for its portion of the array.
- Uses `#pragma omp for schedule(static)` to divide array elements evenly.
- Combines local results into global results using `#pragma omp critical`.
- Calculates the global average.

Code:

```
int globalMin = INT_MAX;
int globalMax = INT_MIN;
long long globalSum = 0;
#pragma omp parallel
{
    int localMin = INT_MAX;
    int localMax = INT_MIN;
    long long localSum = 0;
    #pragma omp for schedule(static)
    for (int i = 0; i < n; i++) {
        localMin = min(localMin, data[i]);
        localMax = max(localMax, data[i]);
        localSum += data[i];
    }
    #pragma omp critical
    {
        globalMin = min(globalMin, localMin);
        globalMax = max(globalMax, localMax);
        globalSum += localSum;
```

```
    }
}
double globalAvg = static_cast<double>(globalSum) / n;
```

2.3 Main Function

Purpose: Sets up the experiment and measures performance.

How it Works:

- Takes input for array size and elements.
- Runs Sequential Reduction, measures time, and stores results.
- Runs Parallel Reduction, measures time, and stores results.
- Verifies correctness by comparing results.
- Prints times, speedup, efficiency, correctness, and results.

3 Output Explanation

Input: Array size 6, elements [10, 5, 8, 3, 12, 7]

```
Sequential Reduction Time: 0.0009 ms
Parallel Reduction Time: 2.6318 ms
Speedup: 0.000341971
Threads Used: 8
Efficiency: 4.27464e-05
Correctness: Pass
Minimum: 3
Maximum: 12
Sum: 45
Average: 7.5
```

Key Points:

- **Sequential Time**: 0.0009 ms (fast).
- **Parallel Time**: 2.6318 ms (slower).
- **Speedup**: 0.000341971 (< 1, parallel is slower).
- **Efficiency**: 4.27464e-05 (low, threads underutilized).
- **Correctness**: Pass (results match).
- **Results**: Min 3, Max 12, Sum 45, Avg 7.5.

4 How to Explain to the Examiner

1. **Introduce the Practical**:
    - "This practical computes the minimum, maximum, sum, and average of an array using sequential and parallel reduction with OpenMP. We measure times, compute speedup and efficiency, and verify results match."
2. **Explain Code**:
    - **Sequential**: "Loops through the array to find min, max, sum, and average, on one thread."
    - **Parallel**: "Splits the array across threads. Each thread computes local min, max, sum, then combines them using a critical section."
    - **Main**: "Inputs array, runs both reductions, measures times, checks correctness, and prints metrics."
3. **Explain Output**:
    - "For 6 elements and 8 threads, Sequential took 0.0009 ms, Parallel took 2.6318 ms. Speedup was 0.000341971, efficiency was low. Correctness passed with min 3, max 12, sum 45, avg 7.5."
    - "Parallel was slower due to thread overhead for a small array."
4. **Explain Why Sequential Was Faster**:
    - "The array had only 6 elements, so thread creation and synchronization overhead was higher than the computation time."

5 Tips for Explanation

- Simplify: Reduction finds min, max, sum; parallel splits work across threads.
- Highlight OpenMP: `#pragma omp parallel` creates threads, `#pragma omp critical` combines results.
- Explain overhead: Small arrays make parallel slower.
- Be ready for questions: Why parallel is slower, how correctness is verified, or how reduction is parallelized.