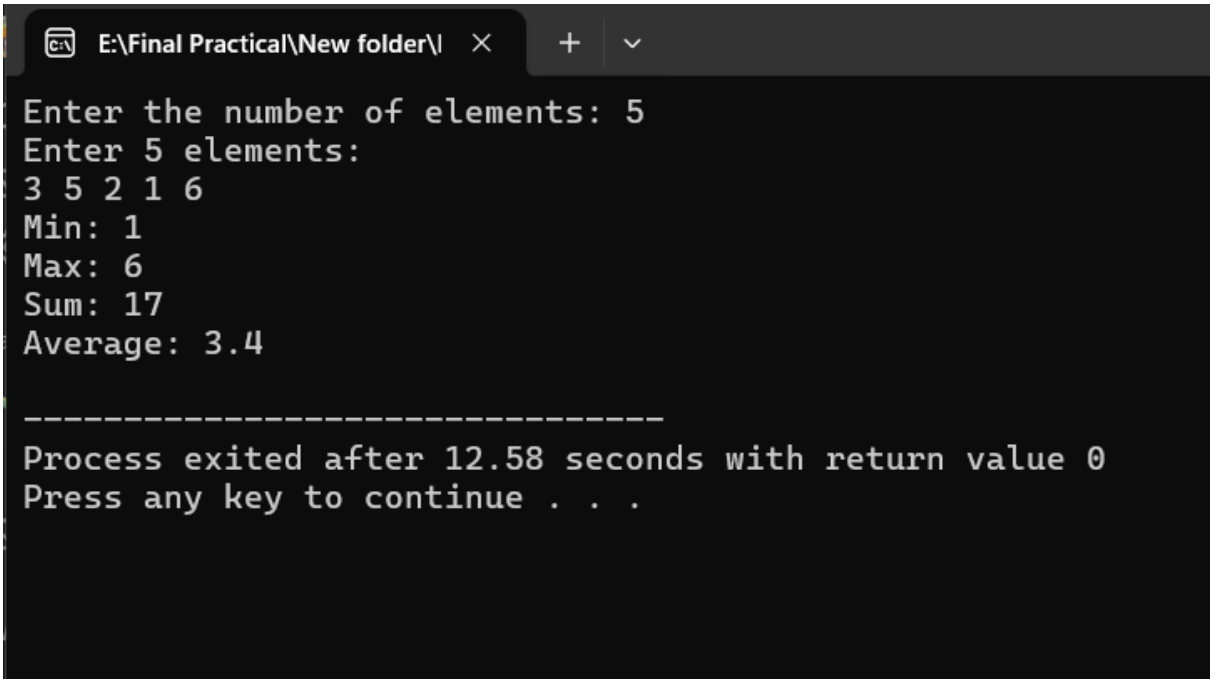
HPC Practical 5

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

using namespace std;

struct Result {
    int min;
    int max;
    int sum;
    double average;
};

Result parallel_reduction(const vector<int>& data) {
    int num_elements = data.size();
    int min_val = data[0];
    int max_val = data[0];
    int sum = 0;

    #pragma omp parallel for reduction(min:min_val) reduction
    for (int i = 0; i < num_elements; ++i) {
        int value = data[i];
        sum += value;

        if (value < min_val) {
            min_val = value;
        }

        if (value > max_val) {
            max_val = value;
        }
    }

    double average = static_cast<double>(sum) / num_elements;
```

```
    return { min_val, max_val, sum, average };
}

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

    vector<int> data(num_elements);
    cout << "Enter " << num_elements << " elements:\n";
    for (int i = 0; i < num_elements; ++i) {
        cin >> data[i];
    }

    Result result = parallel_reduction(data);

    cout << "Min: " << result.min << endl;
    cout << "Max: " << result.max << endl;
    cout << "Sum: " << result.sum << endl;
    cout << "Average: " << result.average << endl;

    return 0;
}
```



Let's break down the Merge Sort step by step for the given example:

Let's consider the example where the user enters 5 elements: 3, 5, 2, 1, and 6.

1. **Initialization**:
   - `num_elements` is set to 5.
   - `data` vector is created with 5 elements: [3, 5, 2, 1, 6].
   - `min_val` and `max_val` are initialized to the first element of `data`, which is 3.
   - `sum` is initialized to 0.
2. **Parallel Reduction Loop**:
   - The loop is parallelized using OpenMP with `#pragma omp parallel for`.
   - Each thread has its own `min_val`, `max_val`, and `sum` variables.
   - Each thread iterates over a portion of the `data` vector and updates its local `min_val`, `max_val`, and `sum`.
   - At the end of the loop, the local values from each thread are combined to calculate the final `min_val`, `max_val`, and `sum`.
3. **Thread 1 (for the sake of example)**:

- Thread 1 starts with `min_val` = 3, `max_val` = 3, and `sum` = 0.
- Iterates over its assigned portion of `data` [3, 5] and updates `min_val` to 3, `max_val` to 5, and adds 3 + 5 = 8 to `sum`.

4. **Thread 2 (continuing the example)**:
   - Thread 2 starts with `min_val` = 3, `max_val` = 3, and `sum` = 0.
   - Iterates over its assigned portion of `data` [2, 1] and updates `min_val` to 1, `max_val` to 3, and adds 2 + 1 = 3 to `sum`.
5. **Thread 3, 4, 5**:
   - Similar to Thread 1 and Thread 2, each updating their local `min_val`, `max_val`, and `sum`.
6. **Combining Results**:
   - After all threads finish, their local `min_val`, `max_val`, and `sum` are combined.
   - The final `min_val` is the minimum of all local `min_val` (1).
   - The final `max_val` is the maximum of all local `max_val` (5).
   - The final `sum` is the sum of all local `sum` (3 + 5 + 2 + 1 + 6 = 17).
7. **Calculating Average**:
   - The average is calculated as the `sum` divided by the number of elements (17 / 5 = 3.4).
8. **Output**:
   - The program prints:
     - Min: 1
     - Max: 5
     - Sum: 17
     - Average: 3.4

This demonstrates how parallel reduction works by breaking down the computation into smaller parts that can be executed concurrently by multiple threads, improving performance for large datasets.

Theory for Practical

📌 What is Parallel Reduction

Parallel reduction is a technique used in parallel programming to efficiently combine values from multiple threads or processes into a single result. It is commonly used for operations like finding the minimum, maximum, sum, or average of a set of values.

Here's how parallel reduction typically works:

1. **Initialization**: Each thread or process starts with its own local copy of the result variable (e.g., `min_val`, `max_val`, `sum`).
2. **Partial Computation**: Each thread computes a partial result using its portion of the data. For example, in a sum reduction, each thread would add up a subset of the values.
3. **Combining Results**: The partial results from all threads are combined to produce a single result. This step is often done hierarchically, where pairs of partial results are combined, then those results are combined, and so on, until a single result is obtained.
4. **Final Result**: The final combined result is used as the output of the reduction operation.

Parallel reduction is useful for improving the performance of operations that can be expressed as associative and commutative, meaning the order in which operations are performed does not affect the result (e.g., addition, multiplication). It allows for the efficient use of multiple processors or cores to speed up computations.

📌 What is OpenMP

OpenMP is a programming interface that helps you write programs that can run faster on computers with multiple processors or cores. It does this by allowing

you to divide your program into parts that can be done at the same time by different processors. This can make your program run faster and more efficiently.

📌 Explain Parallel BFS to a 5-year-old kid

Imagine you and your friends have a bunch of candies, and you want to know how many candies you all have together. Instead of each of you counting your candies separately and then adding them up, you decide to work together to count them faster.

Here's how it works:

1. **Divide the candies**: You and your friends divide the candies equally among yourselves. Each of you counts your own candies.
2. **Counting**: While each of you is counting your candies, you also keep track of the total number of candies counted so far. For example, if you have 5 candies and your friend has 3, the total counted so far is 8.
3. **Combining**: After everyone finishes counting their candies, you all gather together and add up the total number of candies each of you counted. This gives you the total number of candies you all have together.
4. **Final Count**: The final count is the total number of candies you all have together, which you found by working together and adding up the counts from each person.

In this way, you were able to find the total number of candies faster by working together and combining your counts, which is similar to how parallel reduction works in programming.

Viva Questions ❓ ❓

1. What is parallel reduction?

- **Answer:** Parallel reduction is a technique used in parallel programming to efficiently combine values from multiple threads or processes into a single

result. It is commonly used for operations like finding the minimum, maximum, sum, or average of a set of values.

2. How does parallel reduction work?

- **Answer:** Parallel reduction typically works by dividing the data into segments, computing partial results for each segment in parallel, and then combining these partial results to obtain the final result.

3. What are the benefits of parallel reduction?

- **Answer:** Parallel reduction allows for the efficient use of multiple processors or cores to speed up computations, especially for operations that can be expressed as associative and commutative.

4. Can you give an example of a parallel reduction operation?

- **Answer:** One example is finding the sum of an array of numbers. Each thread can compute the sum of a subset of the array, and then these partial sums can be combined to find the total sum.

5. What are some operations that can be performed using parallel reduction?

- **Answer:** Operations such as finding the minimum, maximum, sum, or average of a set of values can be performed using parallel reduction.

6. How is data divided and combined in parallel reduction?

- **Answer:** Data is divided into segments, and each segment is processed independently to compute a partial result. These partial results are then combined hierarchically to produce the final result.

7. What is the difference between parallel reduction and sequential reduction?

- **Answer:** In sequential reduction, the reduction operation is performed sequentially, one after the other. In parallel reduction, the reduction operation is performed concurrently by multiple threads or processes, which can lead to faster computation times.

8. When is parallel reduction typically used?

- **Answer:** Parallel reduction is typically used when dealing with large datasets or when performance optimization is required in parallel programming.