```
#include <iostream>
#include <vector>
#include <algorithm>
#include <numeric>
#include <thread>

// Define the number of threads to use
const int NUM_THREADS = 4;
// Define the data size
const int DATA_SIZE = 100;
// Define the data type
using DataType = double;
// Define the data vector type
using DataVec = std::vector<DataType>;
// Define the reduction result type
struct ReductionResult {
    DataType min;
    DataType max;
    DataType sum;
    DataType avg;
};

// Define the parallel reduction function
ReductionResult parallel_reduction(const DataVec& data) {
    // Calculate the number of elements per thread
    const int elements_per_thread = data.size() / NUM_THREADS;

    // Define the vector to store the results
    std::vector<ReductionResult> results(NUM_THREADS);

    // Create the threads and perform the reduction
```

```
    std::vector<std::thread> threads(NUM_THREADS);
    for (int i = 0; i < NUM_THREADS; ++i) {
        threads[i] = std::thread([&data, &results, elements_per_thread, i] {
            // Calculate the range of elements for this thread
            const int start_index = i * elements_per_thread;
            const int end_index = start_index + elements_per_thread;

            // Perform the reduction for this range of elements
            ReductionResult& result = results[i];
            result.min = *std::min_element(data.begin() + start_index, data.begin() + end_index);
            result.max = *std::max_element(data.begin() + start_index, data.begin() + end_index);
            result.sum = std::accumulate(data.begin() + start_index, data.begin() + end_index,
static_cast<DataType>(0));
            result.avg = result.sum / static_cast<DataType>(elements_per_thread);
        });
    }

    // Join the threads
    for (auto& thread : threads) {
        thread.join();
    }

    // Merge the results
    ReductionResult result;
    result.min = results[0].min;
    result.max = results[0].max;
    result.sum = results[0].sum;
    result.avg = results[0].avg;
    for (int i = 1; i < NUM_THREADS; ++i) {
        result.min = std::min(result.min, results[i].min);
        result.max = std::max(result.max, results[i].max);
```

```
        result.sum += results[i].sum;
        result.avg += results[i].avg;
    }
    result.avg /= static_cast<DataType>(data.size());

    // Return the result
    return result;
}

int main() {
    // Generate some random data
    DataVec data(DATA_SIZE);
    std::generate(data.begin(), data.end(), [] { return static_cast<DataType>(rand()) / RAND_MAX; });

    // Perform the parallel reduction
    ReductionResult result = parallel_reduction(data);

    // Print the result
    std::cout << "Min: " << result.min << std::endl;
    std::cout << "Max: " << result.max << std::endl;
    std::cout << "Sum: " << result.sum << std::endl;
    std::cout << "Average: " << result.avg << std::endl;

    return 0;
}
```

Output :

Min: 0.0163006

Max: 0.998925

Sum: 54.6825

Average: 0.021873