Assignment No: Group A-3

Aim: Implement parallel reduction using Min, Max, Sum and Average Operations.

Objective: To study and implementation of directive based parallel programming model.

Pre-requisites:

64-bit Open source Linux or its derivative

Programming Languages: C/C++

Theory:

OpenMP:

OpenMP is a set of C/C++ pragmas which provide the programmer a high-level front-end interface which get translated as calls to threads. The key phrase here is "higher-level"; the goal is to better enable the programmer to "think parallel" alleviating him/her of the burden and distraction of dealing with setting up and coordinating threads. For example, the OpenMP directive.

OpenMP Core Syntax:

Most of the constructs in OpenMP are compiler directives:

#pragma omp construct [clause [clause]...]

1. The min_reduction function finds the minimum value in the input array using the #pragma omp parallel for reduction (min: min_value) directive, which creates a parallel region and divides the loop iterations among the available threads. Each thread performs the comparison operation in parallel and updates the min_value variable if a smaller value is found.
2. Similarly, the max_reduction function finds the maximum value in the array, sum_reduction function finds thesum of the elements of array and average_reduction function finds the average of the elements of array by dividing the sum by the size of the array.
3. The reduction clause is used to combine the results of multiple threads into a single value, which is then returned by the function. The min and max operators are used for the min_reduction and max_reduction functions, respectively, and the + operator is used for the sum_reduction and average_reduction functions. Inthe main function, it creates a vector and calls the functions min_reduction, max_reduction, sum_reduction, andaverage_reduction to compute the values of min, max, sum and average respectively.

Example:

```
#include <iostream>
#include <vector>
 #include <omp.h>
using namespace std;
void min_reduction(vector<int>& arr) {
int min_value = INT_MAX;
#pragma omp parallel for reduction(min: min_value)for (int i = 0; i < arr.size(); i++) {
if (arr[i] < min_value) {min_value = arr[i];
}}
cout << "Minimum value: " << min_value << endl;
}
void max_reduction(vector<int>& arr) {int max_value = INT_MIN;
#pragma omp parallel for reduction(max: max_value) {
max_value = arr[i];
}}
cout << "Maximum value: " << max_value << endl;}
void sum_reduction(vector<int>& arr) {
int sum = 0;
#pragma omp parallel for reduction(+: sum)for (int i = 0; i < arr.size(); i++) {
sum += arr[i]; }
cout << "Sum: " << sum << endl; }
void average_reduction(vecto <int>& arr) {int sum = 0;
#pragma omp parallel for reduction(+: sum)for (int i = 0; i < arr.size(); i++) {
sum += arr[i];}
cout << "Average: " << (double)sum / arr.size() << endl;}
int main() {
vector<int> arr = {5, 2, 9, 1, 7, 6, 8, 3, 4};
min_reduction(arr);
max_reduction(arr);
sum_reduction(arr);
average_reduction(arr);
}
```

Conclusion: Thus, we have successfully implemented parallel reduction using Min, Max, Sum and Average Operations.