**Implement Min, Max, Sum and Average operations using Parallel Reduction.**

#include <iostream>

//#include <vector>

#include <omp.h>

#include <climits>

using namespace std;

void min\_reduction(int arr[], int n) {

int min\_value = INT\_MAX;

#pragma omp parallel for reduction (min: min\_value)

for (int i = 0; i < n; i++) {

if (arr[i] < min\_value) {

min\_value = arr[i];

}

}

cout << "Minimum value: " << min\_value << endl;

}

void max\_reduction(int arr[], int n) {

int max\_value = INT\_MIN;

#pragma omp parallel for reduction (max: max\_value)

for (int i = 0; i < n; i++) {

if (arr[i] > max\_value) {

max\_value = arr[i];

}

}

cout << "Maximum value: " << max\_value << endl;

}

void sum\_reduction(int arr[], int n) {

int sum = 0;

#pragma omp parallel for reduction (+: sum)

for (int i = 0; i < n; i++) {

sum += arr[i];

}

cout << "Sum: " << sum << endl;

}

void average\_reduction(int arr[], int n) {

int sum = 0;

#pragma omp parallel for reduction (+: sum)

for (int i = 0; i < n; i++) {

sum += arr[i];

}

cout << "Average: " << (double)sum / (n-1) << endl;

}

int main () {

int \*arr,n;

cout<<"\n enter total no of elements=>";

cin>>n;

arr=new int[n];

cout<<"\n enter elements=>";

for (int i=0; i<n;i++)

{

cin>>arr[i];

}

// int arr[] = {5, 2, 9, 1, 7, 6, 8, 3, 4};

// int n = size(arr);

min\_reduction (arr, n);

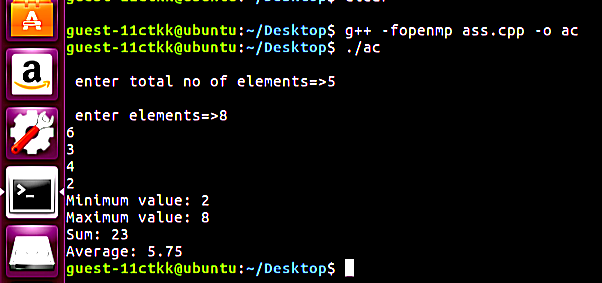
max\_reduction (arr, n);

sum\_reduction (arr, n);

average\_reduction (arr, n);

}

Output



**Void Min\_reduction()**

* void min\_reduction(vector<int>& arr) declares a void function that takes a reference to an integer vector as its argument.
* int min\_value = INT\_MAX; initializes an integer variable min\_value to the largest possible integer value using the INT\_MAX constant from the <climits> header file. This is done to ensure that min\_value is initially greater than any element in arr.
* #pragma omp parallel for reduction (min: min\_value) is an OpenMP directive that specifies that the following loop should be executed in parallel using multiple threads. The reduction (min: min\_value) clause indicates that each thread should maintain a private copy of min\_value and update it with the minimum value it finds in its portion of the loop. Once the loop is complete, OpenMP will combine all the private copies of min\_value into a single shared value that represents the minimum value in arr.
* for (int i = 0; i < arr.size(); i++) { is a loop that iterates over each element of arr.
* if (arr[i] < min\_value) {min\_value = arr[i]; } checks if the current element of arr is less than min\_value. If so, it updates min\_value to be the current element.
* cout << "Minimum value: " << min\_value << endl; prints out the minimum value found in arr.

**void max\_reduction ()**

* void max\_reduction(vector<int>& arr) declares a void function that takes a reference to an integer vector as its argument.
* int max\_value = INT\_MIN; initializes an integer variable max\_value to the smallest possible integer value using the INT\_MIN constant from the <climits> header file. This is done to ensure that max\_value is initially smaller than any element in arr.
* #pragma omp parallel for reduction (max: max\_value) is an OpenMP directive that specifies that the following loop should be executed in parallel using multiple threads. The reduction (max: max\_value) clause indicates that each thread should maintain a private copy of max\_value and update it with the maximum value it finds in its portion of the loop. Once the loop is complete, OpenMP will combine all the private copies of max\_value into a single shared value that represents the maximum value in arr.
* for (int i = 0; i < arr.size(); i++) { is a loop that iterates over each element of arr.
* if (arr[i] > max\_value) {max\_value = arr[i];} checks if the current element of arr is greater than max\_value. If so, it updates max\_value to be the current element.
* cout << "Maximum value: " << max\_value << endl; prints out the maximum value found in arr.

**#include <climits>**

<climits> is a header file in C++ that contains constants related to integer types. This header file provides implementation-defined constants for minimum and maximum values of integral types, such as INT\_MAX (maximum value of int) and INT\_MIN (minimum value of int).

Using these constants instead of hardcoding the values of the minimum and maximum integer values is a good practice because it makes the code more readable and avoids the possibility of introducing errors in the code. The use of these constants also ensures that the code will work correctly across different platforms and compilers.

**INT\_MIN:**

Minimum value for an object of type int

Value of INT\_MIN is -32767 (-215+1) or less\*

**INT\_MAX:**

Maximum value for an object of type int

Value of INT\_MAX is 2147483647 (-231 to 231-1)