Min, Max, Sum and Average Parallel Reduction.

Title of the Assignment: Implement Min, Max, Sum and Average operations using Parallel Reduction.

Objective of the Assignment: To understand the concept of parallel reduction and how it can be used to perform basic mathematical operations on given data sets.

Prerequisite:

1. Parallel computing architectures
2. Parallel programming models
3. Proficiency in programming languages

Contents for Theory:

1. What is parallel reduction and its usefulness for mathematical operations on large data?
2. Concept of OpenMP
3. How do parallel reduction algorithms for Min, Max, Sum, and Average work, and what are their advantages and limitations?

Parallel Reduction.

Here's a **function-wise manual** on how to understand and run the sample C++ program that demonstrates how to implement Min, Max, Sum, and Average operations using parallel reduction.

1. **Min_Reduction function**
    - The function takes in a vector of integers as input and finds the minimum value in the vector using parallel reduction.
    - The OpenMP reduction clause is used with the "min" operator to find the minimum value across all threads.
    - The minimum value found by each thread is reduced to the overall minimum value of the entire array.
    - The final minimum value is printed to the console.

2. **Max_Reduction function**
    - The function takes in a vector of integers as input and finds the maximum value in the vector using parallel reduction.
    - The OpenMP reduction clause is used with the "max" operator to find the maximum value across all threads.
    - The maximum value found by each thread is reduced to the overall maximum value of the entire array.
    - The final maximum value is printed to the console.
3. **Sum_Reduction function**
    - The function takes in a vector of integers as input and finds the sum of all the values in the vector using parallel reduction.
    - The OpenMP reduction clause is used with the "+" operator to find the sum across all threads.
    - The sum found by each thread is reduced to the overall sum of the entire array.
    - The final sum is printed to the console.

4. **Average_Reduction function**
    - The function takes in a vector of integers as input and finds the average of all the values in the vector using parallel reduction.
    - The OpenMP reduction clause is used with the "+" operator to find the sum across all threads.
    - The sum found by each thread is reduced to the overall sum of the entire array.
    - The final sum is divided by the size of the array to find the average.
    - The final average value is printed to the console.

5. **Main Function**
    - The function initializes a vector of integers with some values.
    - The function calls the min_reduction, max_reduction, sum_reduction, and average_reduction functions on the input vector to find the corresponding values.

- The final minimum, maximum, sum, and average values are printed to the console.

6. **Compiling and running the program**

   Compile the program: You need to use a C++ compiler that supports OpenMP, such as g++ or clang. Open a terminal and navigate to the directory where your program is saved. Then, compile the program using the following command:

   ```
   $ g++ -fopenmp program.cpp -o program
   ```

   This command compiles your program and creates an executable file named "program". The "-fopenmp" flag tells the compiler to enable OpenMP.

   Run the program: To run the program, simply type the name of the executable file in the terminal and press Enter:

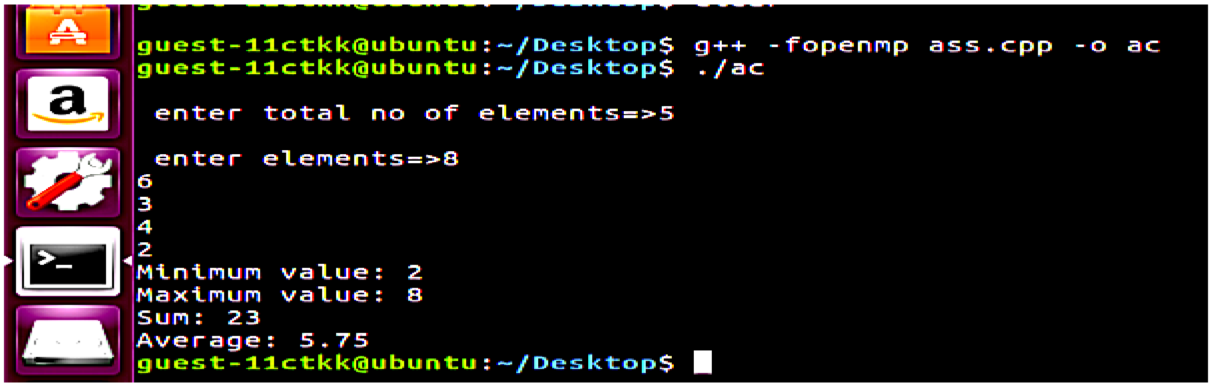
   ```
   $ ./program
   ```

Conclusion: We have implemented the Min, Max, Sum, and Average operations using parallel reduction in C++ with OpenMP. Parallel reduction is a powerful technique that allows us to perform these operations on large arrays more efficiently by dividing the work among multiple threads running in parallel. We presented a code example that demonstrates the implementation of these operations using parallel reduction in C++ with OpenMP. We also provided a manual for running OpenMP programs on the Ubuntu platform.

Assignment Question

1. What are the benefits of using parallel reduction for basic operations on large arrays?
2. How does OpenMP's "reduction" clause work in parallel reduction?
3. How do you set up a C++ program for parallel computation with OpenMP?
4. What are the performance characteristics of parallel reduction, and how do they vary based on input size?
5. How can you modify the provided code example for more complex operations using parallel reduction?

Output



```
guest-11ctkk@ubuntu:~/Desktop$ g++ -fopenmp ass.cpp -o ac
guest-11ctkk@ubuntu:~/Desktop$ ./ac

 enter total no of elements=>5

 enter elements=>8
6
3
4
2
Minimum value: 2
Maximum value: 8
Sum: 23
Average: 5.75
guest-11ctkk@ubuntu:~/Desktop$
```

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 ($-2^{15}+1$) or less*
INT_MAX :
Maximum value for an object of type int
Value of INT_MAX is 2147483647 (-2^{31} to $2^{31}-1$)